Available online at www.sciencedirect.com
ScienceDirect
Computers & Operations Research 36 (2009) 655–673
computers & operations research
www.elsevier.com/locate/cor

Ant colony optimization for the two-dimensional loading vehicle routing problem

Guenther Fuellerer[a], Karl F. Doerner[a,*], Richard F. Hartl[a], Manuel Iori[b]

[a] *Department of Business Administration, University of Vienna, Bruenner Strasse 72, 1210 Vienna, Austria*
[b] *DISMI, University of Modena and Reggio Emilia, Via Amendola 2, 42100 Reggio Emilia, Italy*

Available online 30 October 2007

Abstract

In this paper a combination of the two most important problems in distribution logistics is considered, known as the two-dimensional loading vehicle routing problem. This problem combines the loading of the freight into the vehicles, and the successive routing of the vehicles along the road network, with the aim of satisfying the demands of the customers.

The problem is solved by different heuristics for the loading part, and by an ant colony optimization (ACO) algorithm for the overall optimization. The excellent behavior of the algorithm is proven through extensive computational results.

The contribution of the paper is threefold: first, on small-size instances the proposed algorithm reaches a high number of proven optimal solutions, while on large-size instances it clearly outperforms previous heuristics from the literature. Second, due to its flexibility in handling different loading constraints, including items rotation and rear loading, it allows us to draw qualitative conclusions of practical interest in transportation, such as evaluating the potential savings by permitting more flexible loading configurations. Third, in ACO a combination of different heuristic information usually did not turn out to be successful in the past. Our approach provides an example where an ACO algorithm successfully combines two completely different heuristic measures (with respect to loading and routing) within one pheromone matrix.

Keywords: Vehicle routing; Ant colony optimization; Two-dimensional packing

1. Introduction

Two of the most important problems in distribution logistics concern the loading of the freight into the vehicles, and the successive routing of the vehicles along the road network, with the aim of satisfying the demands of the customers.

In the field of combinatorial optimization, loading and routing problems have been studied intensively but separately. Only in recent years some attention has been devoted to their combined optimization. The obvious advantage is that, by considering the information on the freight to be loaded, one can construct more appropriate routes for the vehicles. This comes at a price since the combinatorial difficulty of the problem increases considerably.

In this paper we address a particular combination of two-dimensional loading and vehicle routing, defined as the *two-dimensional loading capacitated vehicle routing problem* (2L-CVRP, see Iori [1,2]). In the 2L-CVRP customers demand sets of rectangular weighted items, while vehicles have a weight capacity and a rectangular two-dimensional

* Corresponding author.
E-mail addresses: Guenther.Fuellerer@univie.ac.at (G. Fuellerer), Karl.Doerner@univie.ac.at (K.F. Doerner), Richard.Hartl@univie.ac.at (R.F. Hartl), manuel.iori@unimore.it (M. Iori).

0305-0548/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2007.10.021

loading surface. The aim is to load the items into the vehicles and deliver them to the customers, through a road network, with minimum total cost. The problem is of interest in the transportation of three-dimensional box-shaped items that cannot be stacked one on top of each other, because of their fragility, weight or large dimensions. This happens, for example, when the transported items are kitchen appliances or large mechanical components. The 2L-CVRP generalizes the well known *capacitated vehicle routing problem* (CVRP), in which the demand of each customer is only represented by a positive integer, representing weight or volume.

We propose an effective heuristic based on Ant Colony Optimization (ACO). Starting point is the savings based ACO algorithm for the CVRP (see [3,4]), which is modified and extended by incorporating loading heuristics. The algorithm searches the space of routing solutions, while checking the feasibility of the two-dimensional loading of each route by means of lower bounds, fast heuristics and a truncated branch-and-bound. Our aim is to provide good solutions to large-size instances, leading to an algorithm of practical use in transportation.

The rest of the paper is organized as follows. In Section 2 we describe the 2L-CVRP and discuss possible loading configurations. In Section 3 we briefly review the previous literature on combined loading and routing problems. In Section 4 we present the algorithms used for the loading subproblem, while in Section 5 we give the implementation of the ACO for the combined loading and routing problem. In Section 6 we present extensive computational results for instances from the literature, showing a very good performance of our algorithm.

2. Problem description

In the 2L-CVRP we are given a complete undirected graph $G = (V_0, E)$, where $V_0 = V \cup \{0\}$ is a set of $n + 1$ vertices corresponding to the depot (0) and the customers ($V = \{1, \ldots, n\}$); E is the set of edges (i, j) between any pair of vertices, with associated cost c_{ij} $(i, j = 0, \ldots, n)$. Also given are K identical vehicles, characterized by a weight capacity D and a rectangular loading surface of width W and length L. Each vehicle has a single opening for loading and unloading items (rear loading). The opening is placed on the W edge and has width W. Let us also define $c_r(k)$ as the total cost of route k ($k = 1, \ldots, K$) and $S(k)$ as the set of customers belonging to route k.

Each customer i $(i = 1, \ldots, n)$ demands a set of m_i items of total weight d_i. Each item (i, p) $(i = 1, \ldots, n$, $p = 1, \ldots, m_i)$ has width w_{ip} and length l_{ip}. Let us also define $a_i = \sum_{p=1}^{m_i} w_{ip} l_{ip}$ as the total area of the items demanded by customer i. We assume that the set of items demanded by each customer can be loaded into a single vehicle and that all customers can be served by using no more than K vehicles. Split deliveries are not allowed, i.e., each customer is visited once. The aim is to find a partition of the customers into routes of minimal total cost, such that for each route there exists a *feasible* loading of the items into the vehicle loading surface.

Let us further specify the concept of feasible loading. In the 2L-CVRP the loading is restricted to be *orthogonal*, i.e., every item must be loaded with its edges parallel to the edges of the vehicles. For each vehicle all the items must be completely contained in the loading surface and cannot overlap, and the total sum of the weights cannot exceed the vehicle weight capacity. A usual and practical request in transportation is also that, when visiting a customer, his/her items can be unloaded from the vehicle by means of forklift trucks, without having to move items belonging to successive customers along the route. This implies that the portion of loading surface between each item of the customer being served and the opening of the vehicle must be empty (or filled by items of the same customer, that will be removed first). This constraint is denoted in the literature as either *sequential loading* or *rear loading* constraint (see [5]). In the remaining of the paper we will stick to rear loading. The term *unrestricted loading* (see [6]) is used instead when re-arrangements of the items in the vehicle at the customers' sites are allowed.

Finally, items can be rotated on their basis, although in some rare cases this cannot happen. An *oriented loading* is the one in which items cannot be rotated because, e.g., they are placed on old-fashioned pallets (accessible by forklift trucks only in one direction), or are very heavy, with the weight not evenly distributed. So far the 2L-CVRP has always been addressed in the literature by disregarding items rotation and imposing a fixed orientation. The loading in which items are allowed to be rotated by 90° is called *non-oriented*.

Let us address the resulting problem of loading the items of a given route into a single vehicle. We are given a route r, as an ordered sequence of customers, and an associated set of items $I_r = \{(i, p) : i \in r, p = 1, \ldots, m_i\}$, with the aim of checking if the items in I_r can be feasibly loaded into the vehicle surface. The problem can be expressed as the following minimization problem (2L): pack the items of set I_r into a rectangular strip of width W (i.e., the width of the vehicle) and infinite length, satisfying all operational constraints and with the aim of minimizing the length used. If the 2L solution value is lower or equal to L (i.e., the length of the vehicle), then a feasible loading of the items into

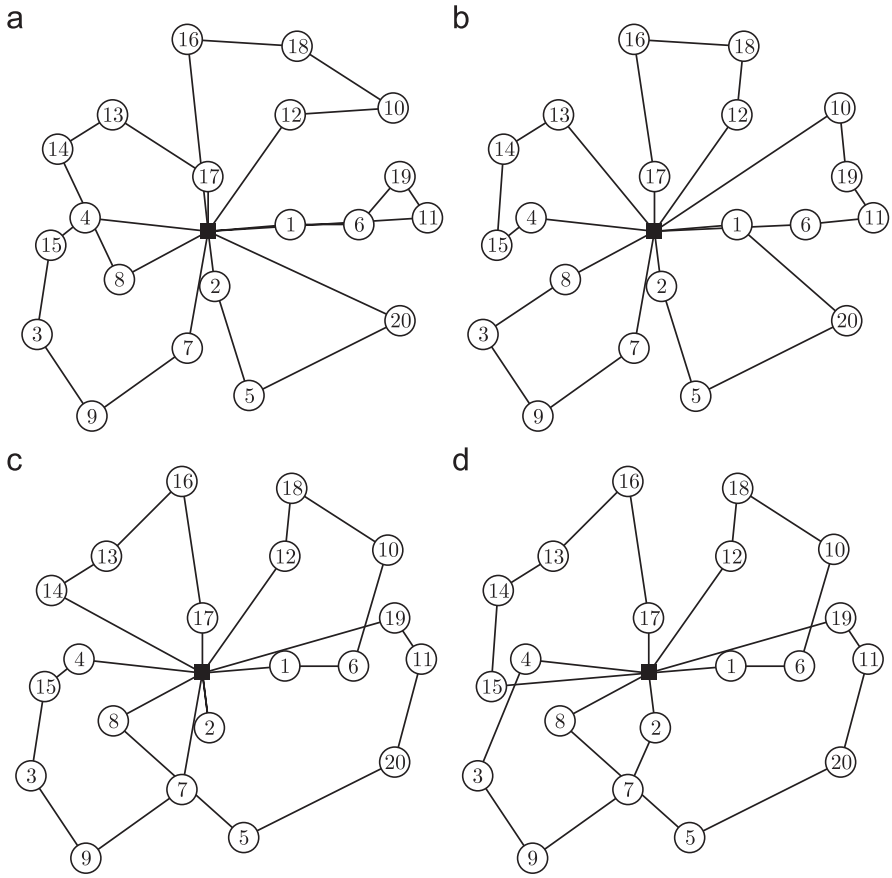

Fig. 1. Solutions to instance 2L-CVRP0*3*02 with different loading configurations: (a) 2|RO|L: $z = 403.93$, (b) 2|UO|L: $z = 387.70$, (c) 2|RN|L: $z = 384.93$, (d) 2|UN|L: $z = 380.35$.

the vehicle surface has been found. The 2L does not take into account the weight constraint on D, which is obviously easily checked in advance.

According to the classification presented above concerning the loading configurations, four different cases can be distinguished.

2|RO|L: two-dimensional *r*ear *o*riented loading;
2|UO|L: two-dimensional *u*nrestricted *o*riented loading;
2|RN|L: two-dimensional *r*ear *n*on-oriented loading;
2|UN|L: two-dimensional *u*nrestricted *n*on-oriented loading.

This is the first work in which the 2L-CVRP with non-oriented loading is addressed. The importance of taking into account different loading configurations is outlined in Figs. 1 and 2. These figures present solutions to instance 2L-CVRP0*3*02, derived from the classical Euclidean CVRP instance E021-04m (see Section 6 below for a description of the instances), and has $n = 20$, $K = 5$ and $D = 100$.

The value z of the heuristic solution found by our algorithm decreases from 403.93 for 2|RO|L (the tightest loading configuration) to 380.35 for 2|UN|L (the loosest loading configuration). The values for 2|RN|L and 2|UO|L are intermediate between these two extremes. The main reason for this improvement in the solution value lies in the increased number of feasible loadings for the vehicles. This fact is outlined in Fig. 2, where we present the four loadings associated with the solutions in Fig. 1. In Figs. 2(c) and 2(d) the grey items are those that have been rotated. In Section 6.4 we will evaluate the effects of the loading configurations on a large number of test instances.

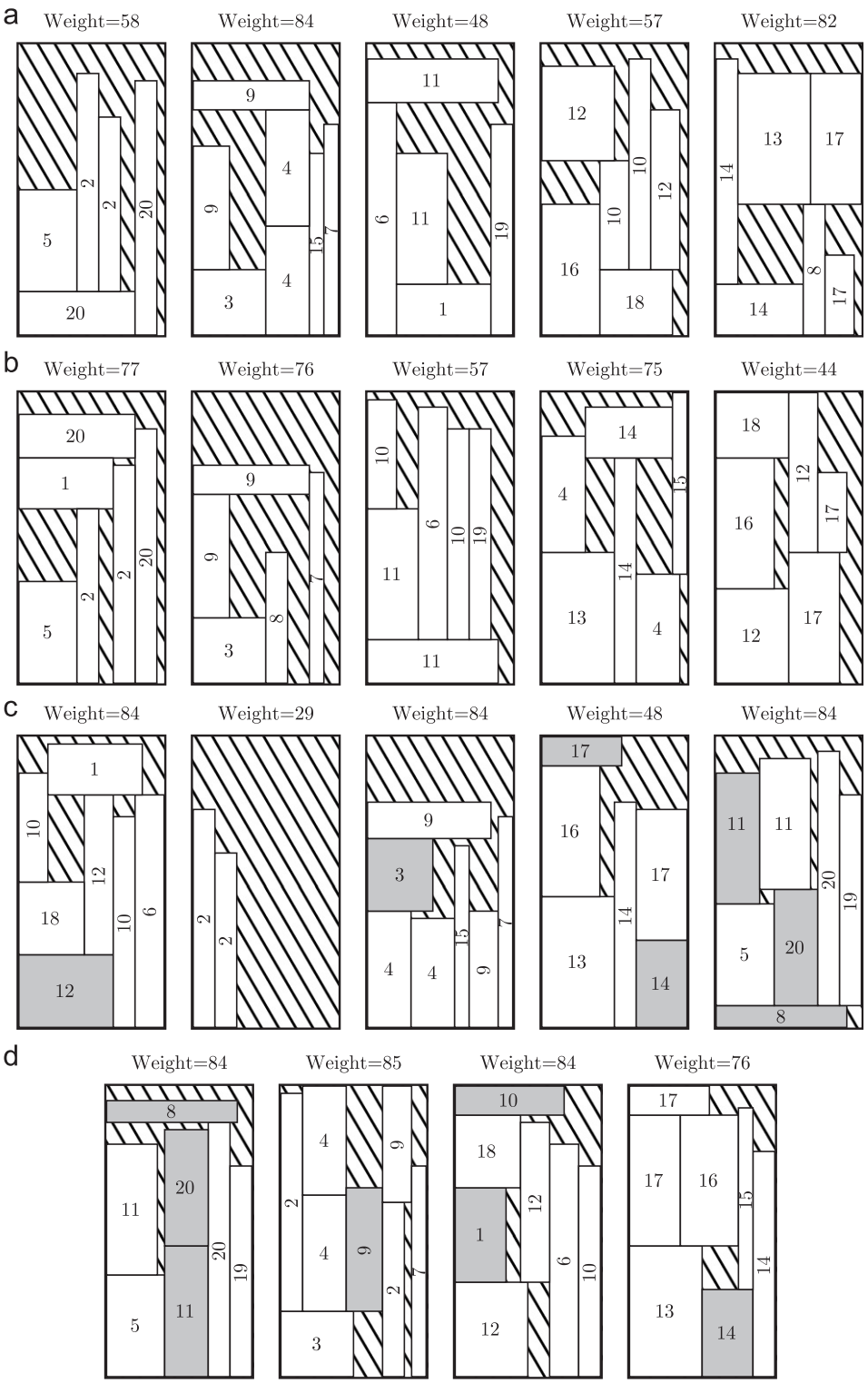

Fig. 2. Vehicle loadings corresponding to the routes in Fig. 1. The numbers in the items correspond to the customer IDs: (a) 2|RO|L, (b) 2|UO|L, (c) 2|RN|L, (d) 2|UN|L.

Another important observation concerns the loading configuration and the efficiency of the routing. The 2L-CVRP with rear loading can show crossings in the optimal solution. In the example, the third vehicle of the 2|RO|L configuration is associated to route (0, 11, 19, 6, 1, 0), with cost 70.53. This route indeed shows a crossing, and if this was erased, one could obtain the sequence (0, 19, 11, 6, 1, 0) with cost 68.42. This however would not be feasible, since the set of items of customer 11 contains a very large item, that cannot be placed side by side with items of other customers in the route. Thus customer 11 must be either the first, or the last in this route. Crossings cannot be optimal in unrestricted loading, and indeed in the solution associated to 2|UO|L and 2|UN|L, no crossing can be seen. Moreover, just four vehicles are needed for providing a feasible solution to 2|UN|L, as showed in Fig. 2(d).

3. Literature survey

The 2L-CVRP with 2|RO|L was solved by Iori et al. [5] by means of branch-and-cut. Their algorithm is based on the classical two-index vehicle flow formulation, with an additional separation procedure for checking eventual loading infeasibility. The loading check is done by means of lower bounds, a simple bottom-left heuristic and a branch-and-bound algorithm, in which the enumeration tree follows the principles proposed by Martello and Vigo [7] and Martello et al. [8]. Their approach could solve to optimality all 2L-CVRP instances with up to 25 customers and 91 items in one day of CPU time per instance.

To address larger-size 2L-CVRP instances Gendreau et al. [6] proposed a tabu search algorithm, derived from the well-known Taburoute Algorithm (see [9]) and working for both 2|RO|L and 2|UO|L. In their algorithm infeasible solutions, both in terms of weight and loading excess, are accepted but are assigned a penalty term in the objective function. The neighborhood consists in removing a customer from a route and inserting it into another route, optimizing both routes involved in the move. The check for a feasible loading is performed by means of lower bounds, heuristics, local search and a truncated branch-and-bound. The algorithm was tested on instances with up to 255 customers and 786 items.

The algorithm in [6] was generalized by Gendreau et al. [10] so as to address the three-dimensional case. In the resulting *three-dimensional loading capacitated vehicle routing problem* (3L-CVRP) the demands of the customers are sets of three-dimensional weighted items, and additional constraints have to be taken into account in the loading subproblem. The algorithm was tested on randomly created instances and on a real-world distribution test bed, obtained from a company operating in the Italian furniture market.

Another real-world loading and routing problem is the *multi-pile vehicle routing problem* (MP-VRP), addressed by Doerner et al. [11] and derived from a company operating in the Austrian timber market. The loading problem consists in placing timber chipboards into different piles available inside the loading space of particular vehicles. The MP-VRP was addressed by Doerner et al. in [11] by means of two different metaheuristics: tabu search and ACO. The tabu search accepts infeasible solutions, while ACO remains inside the space of feasible solutions. Both algorithms make use of the same loading heuristics, but ACO clearly outperforms the tabu search both in terms of solutions quality and speed. This work was the motivation that led us to apply ACO to the 2L-CVRP.

Finally Moura and Oliveira [12,13] addressed a particular combination of three-dimensional loading and CVRP with time windows. They proposed two classes of heuristic algorithms based, respectively, on hierarchical and sequential methods.

There is a huge amount of literature addressing loading and routing problems separately. For what concerns the CVRP, we refer the reader to the recent volume by Toth and Vigo [14] and to the surveys by Cordeau et al. [15] and Cordeau and Laporte [16]. For the two-dimensional loading, we refer instead the reader to the recent survey by Wäscher et al. [17].

4. Algorithms for the two-dimensional loading problem

As mentioned in Section 2 the check on the 2L must be done with respect to 2|RO|L, 2|UO|L, 2|RN|L and 2|UN|L. We are thus interested in providing flexible algorithms, capable of solving the four versions of the 2L. We note that 2|UO|L corresponds to the well-known *strip packing problem*, for which exact algorithms were proposed by Martello et al. [18] and effective heuristic algorithms by, e.g., Alvarez-Valdes et al. [19]. Problem 2|UN|L is known in the literature as the *orthogonal stock-cutting problem*, for which we refer to the work by Burke et al. [20]. To solve these problems, we implemented and tested lower bounds, heuristics, metaheuristics and a truncated branch-and-bound.

Lower bounds for the 2L were obtained by adapting algorithms for the *two-dimensional bin packing problem* (2BPP). In the 2BPP the aim is to pack a set of rectangular items into the minimum number of rectangular bins. If an instance of 2BPP with item set I_r and bin dimensions $[W, L]$ needs more than one bin, than route r is infeasible. We implemented the 2BPP lower bounds proposed by Martello and Vigo [7], which are based on a partitioning of the items into different subsets. Since these lower bounds apply to oriented items, we extended them to the case of rotation through the procedure described in Dell'Amico and Martello [21]. This simple procedure iteratively chooses an item and divides it into a list of smaller items, each of which has square dimensions.

If the lower bounds do not prove the infeasibility of route r, then we start looking for a feasible loading with simple heuristics. These heuristics receive in input the items sorted according to a given criteria and then pack them, one at a time, following a specific strategy. Four different criteria for the initial sorting of the items are applied, so as to consider the four versions of the 2L. For 2|RO|L items are sorted by scanning the customers assigned to route r in reverse order of visit and, for each customer, by listing the items in the corresponding set according to non-increasing width, breaking ties by non-increasing height. For 2|RN|L customers are again scanned as in the previous case, but then items of each customer's set are sorted according to non-increasing area. For 2|UO|L and 2|UN|L the order of visit of the customers is not relevant, thus for 2|UO|L items of all customers are sorted according to non-increasing width, breaking ties by non-increasing height, while for 2|UN|L they are sorted according to non-increasing area.

The first heuristic that we use is an adaptation of the classical *bottom-left fill* (see, e.g., [20]), obtained by generalizing the procedure in [5] for 2|RO|L to the four cases of the 2L. We maintain a list of location positions, to indicate where the items may be placed, in a bottom-left ordering. Each item in order is then placed in the lowest and leftmost position that satisfies all constraints. If non-oriented loading is considered, the placement of an item in a position is checked first with the item original orientation, and then (only if the attempted placement turned out to be infeasible) with a 90° item rotation. The second heuristic extends the *touching Perimeter Algorithm*, originally proposed by Lodi et al. [22] for the 2BPP. As done for the bottom-left fill, we maintain a list of location positions. Each item is then placed in the position being feasible and maximizing the fraction of the item perimeter touching other items or the borders of the vehicle. In the cases in which rotation is allowed (2|RN|L and 2|UN|L), we evaluate feasibility and touching perimeter fraction for each pair item-position.

If the simple heuristics fail to provide a feasible loading, then a local search is applied. This is done by switching the positions of two items in the input order given to the heuristics, so as to obtain different 2L solutions. For 2|RO|L and 2|RN|L, where clustering of the items in the customer's sets is important, we first consider switchings of items belonging to the same customer, while in a second moment we also allow permutations of items belonging to different customers. For 2|UO|L and 2|UN|L instead, we allow from the beginning permutations of items belonging to different customers. The procedure is halted after a maximum number ϕ of iterations, or when a solution of value lower than or equal to L is found. At each iteration both the two simple heuristics described above are invoked with the new items input sorting.

If none of the previous algorithms has proven feasibility or infeasibility of the loading, then a truncated branch-and-bound is started. We generalize the approach in [5] for 2|RO|L to the cases of unrestricted and non-oriented loadings. At each node of the enumeration tree, we try to place each remaining item in a limited subset of location position, i.e., in the so-called *corner points* (see, e.g., [7]). Backtracking is performed when an item cannot enter any corner point. Lower bounds tailored for the 2L are used to fathom nodes. The rotation is considered by enlarging the enumeration tree: at each node we try to place each remaining item in each corner point in both item orientations. The procedure is halted when a maximum number of backtracking steps has been performed, or when a maximum CPU time has been elapsed, or when a solution of value lower than or equal to L has been found.

Note that the set of location positions used in the heuristics (see again [20] for a clear description of these positions) is an enlarged set with respect to the set of corner points. The use of corner points within heuristic algorithms was already attempted in [6] but led to deceiving results and was thus disregarded here. Note also that we tried to improve the simple local search scheme described above by introducing a long term memory, so as to obtain a tabu search approach. Despite several attempts this did not lead to interesting results, since the tabu search proved to be not as effective as the truncated branch-and-bound when allowed the same CPU time.

Furthermore, with the aim of limiting the total CPU time required for evaluating the several 2L sub-instances to be solved during the ACO search process, we keep a pool with all the routes for which the 2L was addressed. In the pool we store both the routes for which feasible loadings were found and those for which no feasible loading was provided.

Recall $c_r(k)$ was defined as the total arc cost of a route k. The pool is kept sorted by non-increasing values of $c_r(k)$. When the loading associated to a route r has to be checked, we quickly find in the pool a limited set, if any, of routes

having same cost and having already been checked. If in this set we find r, then we do not perform a call to the loading algorithms. Otherwise, after running the loading algorithms, we insert in the pool a new entry containing the loading information obtained for r.

The sketch of the pseudo code for the overall packing procedure is given in Algorithm 1. The procedure CheckLoading returns true if a feasible loading of the item set $S(k)$ associated to a route k is found, false otherwise.

Algorithm 1. Packing procedure

```
1:  procedure CHECKLOADING(S(k), φ)            ▷S(k) = set of items in route k
2:      Found := CHECKPOOL(S(k), Feas)         ▷Feas = true if k is feasible, false otherwise
3:      If Found = true return Feas
4:      end if
5:      L := LB_MV(S(k))          ▷ Lower bounds from Martello and Vigo [7]
6:      if L > 1 then
7:          Store S(k) in solution pool
8:          return false
9:      end if
10:     Create initial order O
11:     Ψ := BOTTOMLEFT_2L(S(k), O)            ▷Ψ = Length of the loading
12:     Ψ := min{Ψ, TOUCHINGPERIMETER_2L(S(k), O)}
13:     it := 1
14:     while Ψ > L and it ⩽ φ do
15:         Randomly select two items and switch their positions in O
16:         Ψ := min{Ψ, BOTTOMLEFT_2L(S(k), O)}
17:         Ψ := min{TOUCHINGPERIMETER_2L(S(k), O)}
18:         it := it + 1
19:     end while
20:     if Ψ > L then Ψ := min{Ψ, BRANCHANDBOUND(S(k), O)}
21:     end if
22:     Store S(k) in solution pool
23:     If Ψ ⩽ L then return true
24:     else return false
25:     end if
26: end procedure
```

5. Ant colony optimization

We briefly present the standard Savings-based ACO developed for the CVRP, and then describe how it has been modified so at to address the 2L-CVRP.

5.1. Standard savings-based ant colony optimization

The Savings-based ACO (see [3,4]) mainly consists of the iteration of three steps: (1) generation of solutions by a population of ants according to private and pheromone information, (2) application of a local search to the ants' solutions, and (3) update of the pheromone information.

The solution generation procedure is based on the well known *Savings Algorithm* due to Clarke and Wright [23]. This algorithm initially assigns each customer to a separate route. Then, for each pair of customers i and j, it evaluates the savings that can be obtained by combining the two customers on the same route, determined as $s_{ij} = c_{i0} + c_{0j} - c_{ij}$. The best combination is then chosen and the procedure is re-iterated.

The savings-based ACO is initialized with a population of P ants. Then each ant searches for a low-cost feasible solution through an iterative phase that generalizes the original Savings Algorithm. Customers, or partial routes, are combined by sequentially choosing feasible entries (with respect to the vehicle capacity) from a list of savings values. Such a list is denoted by Ω_{Π} and, at each decision step of an ant, is composed by the Π feasible combinations

(i, j) yielding the largest savings values. Choosing a restricted neighborhood size Π enables a more focused search than considering all savings values. The decision making about combining customers is based on a probabilistic rule that takes into account both the above mentioned savings values and the pheromone information. Let τ_{ij} denote the pheromone concentration on edge (i, j), representing how good the combination of these two customers was in the previous iterations. The probability of choosing to combine customers i and j in one route is given by

$$\mathscr{P}_{ij} = \frac{\xi_{ij}}{\sum_{(h,l)\in\Omega_\Pi} \xi_{hl}} \tag{1}$$

for each $(i, j) \in \Omega_\Pi$, where

$$\xi_{ij} = (\tau_{ij})^\alpha (s_{ij})^\beta \tag{2}$$

and α and β are non-negative parameters that bias the relative influence of the pheromone trails and the savings values, respectively. An ant selects an (i, j) combination by using the probability in (1) and then merges the two routes containing i and j. After that, the Ω_Π list of best feasible savings is updated and the procedure is re-executed. When no more feasible savings exist, the iterative procedure is halted and the heuristic solution found is returned.

A solution obtained through this procedure is then post-optimized through a local search procedure in order to ensure local optimality. The *move* and *swap* neighborhoods between routes are sequentially applied to improve the solution quality (see [24] and Kindervater and Savelsbergh [25] for a discussion of local search for vehicle routing problems).

The process is performed for each of the P ants in the population. This corresponds to a complete iteration of the ACO. After that, the pheromone is updated according to the rule proposed by Bullnheimer et al. [26], which is based on the concepts of ranking and elitism. All pheromone values τ_{ij} are initialized to $\tau_0 = 2$ in the first Savings-based ACO iteration. Then, let $0 \leqslant \rho \leqslant 1$ be the trail persistence and F the number of elitists (i.e., those ants leading to the best current solutions), the pheromone update can be written as

$$\tau_{ij} = \rho\tau_{ij} + \sum_{q=1}^{F-1} \Delta\tau_{ij}^q + \Delta\tau_{ij}^* \tag{3}$$

for $i, j = 0, \ldots, n$. First, the arcs belonging to the incumbent solution found by the ACO up to the current iteration are updated. The amount of pheromone laid by the best ant is $\Delta\tau_{ij}^* = F\varepsilon$ if arc (i, j) belongs to the incumbent solution, $\Delta\tau_{ij}^* = 0$ otherwise, with ε being a small constant. Second, the $F - 1$ best ants of the current iteration are allowed to lay pheromone on the edges they traversed. The quantity laid by these ants depends on their rank q, such that the qth best ant lays $\Delta\tau_{ij}^q = (F - q)\varepsilon$ if arc (i, j) belongs to the solution found by the ant, $\Delta\tau_{ij}^q = 0$ otherwise. Edges belonging to neither of these solutions just face a pheromone decay at the rate $(1 - \rho)$, which constitutes the trail evaporation.

The overall procedure described is re-iterated for T times (i.e., T complete iterations of the ACO population). A high level pseudo code is given in Algorithm 2. For a more detailed description of the Savings-based ACO we refer to Reimann et al. [3,4].

Algorithm 2 Standard savings-based ACO

```
1:  procedure AntAlgorithm(V, T, P, Π)          ▷ see Table 1
2:      for it := 0, it < T, it := it + 1 do     ▷ Iterations
3:          Determine ξ_ij from 2 and sort them in non-increasing manner
4:          for a := 0, a < P, a := a + 1 do     ▷ Population size
5:              Initialize ant a
6:              while Profitable and feasible combinations (i, j) exist do
7:                  Determine Ω_Π
8:                  Select two entries in Ω_Π and merge the corresponding partial routes
9:              end while
10:             Apply local search to ant a
11:         end for
12:         Determine elitist solutions and perform pheromone update according to (3)
13:     end for
14: end procedure
```

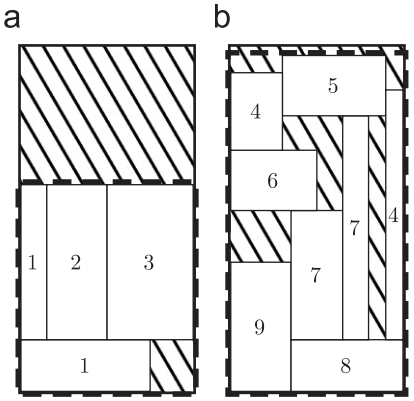

Fig. 3. Virtual rectangles used to evaluate the ξ'_{ij} values.

5.2. Adaptation of the savings-based ACO to the 2L-CVRP

The Savings-based ACO described above has been deeply modified so as to solve the 2L-CVRP. First, in each iteration the algorithms of Section 4 are used to determine feasible loadings and guide the ants decisions. In particular, a heuristic measure based on loading information has been introduced to modify the way in which ants iteratively generate solutions.

Second, the standard Savings-based ACO deals with a free number of vehicles, while in the 2L-CVRP the number of vehicles is limited by K, and this limit can be very strict. To deal with that we introduced an adapted determination of ξ_{ij}. Moreover, ants whose solution requires more than K vehicles are given a penalty term in the objective function (only used in local search) and are not allowed to update the pheromone trail. The determination of the potential customers in the neighborhood Ω_{Π}, originally given by (1) and (2), is modified so as to obtain:

$$\xi'_{ij} = [(\tau_{ij})^{\alpha}(s_{ij})^{\beta}]\frac{\sum_{g\in S} a_g}{\widetilde{a}_S} \tag{4}$$

for $i,\ j=0,\ldots,n$, where S is the subset of customers obtained by combining the two partial routes containing customers i and j. We recall that a_g is the area of all the items demanded by customer g, and define $\widetilde{a}_S$ as the area generated by a *virtual rectangle*, i.e., the smallest rectangle containing all the items currently loaded into the vehicle, as depicted in Fig. 3.

In the example in the figure, the algorithm has to decide whether (a) to combine the partial route (0, 3, 2, 0) and the partial route (0, 1, 0) or (b) to combine the partial route (0, 4, 5, 6, 0) with the partial route (0, 7, 8, 9, 0). Assuming for the sake of simplicity that the two combinations have same savings and pheromone values (i.e., $s_{21} = s_{67}$ and $\tau_{21} = \tau_{67}$), then the ξ'_{ij} values in (4) are mainly influenced by the different loadings. In this case, the combination in Fig. 3(a) would receive a higher probability, due to the fact that the virtual rectangle is better filled for the resulting route (0, 3, 2, 1, 0) than for (0, 4, 5, 6, 7, 8, 9, 0) (94% compared to 79%). At each iteration of the ACO, the loadings of the (i, j) combinations in Ω_{Π} are computed through the algorithms of Section 4 and are used to evaluate the feasibility of the combinations and the virtual rectangles. Then the ξ'_{ij} values are computed and sorted in non-increasing order. For runtime reasons the procedure is done only for the combinations in Ω_{Π} and not for all possible combinations.

The idea of accepting infeasible solutions but penalizing them proved to be very efficient in routing problems (see [9]) and loading and routing problems (see [6]). Hence, we decided to use a modified objective function $z'(s)$, given by

$$z'(s) = z(s) + \gamma q(s) + \delta u(s), \tag{5}$$

where s denotes a solution with $\tilde{v}$ routes. Given $c_r(k)$ as the total arc cost of route k, the terms in (5) are computed as

$$z(s) = \sum_{k=1}^{\tilde{v}} c_r(k),$$

$$q(s) = \sum_{k=1}^{\tilde{v}} \left[\sum_{i \in S(k)} d_i - D \right]^+,$$

$$u(s) = \begin{cases} \left\lceil \dfrac{K}{\sum_{k=1}^{\tilde{v}} \sum_{i \in S(k)} \dfrac{d_i a_i}{WLD}} \right\rceil & \text{if } \tilde{v} > K, \\ 0 & \text{otherwise,} \end{cases}$$

where $q(s)$ represents an excess of weight in the vehicles, and $u(s)$ is a penalty term for the capacity/area utilization, used when the number of vehicles exceeds K. $q(s)$ turned out to be useful for those instances for which capacity constraints are very tight, and is thus used only when $(\sum_{i=1}^{n} d_i)/(KD) \geqslant 0.9$. If the given number of vehicles is exceeded, $u(s)$ guarantees that the local search to a certain extent accepts solutions with worse $z(s)$ value but better capacity/area utilization, so as to be able to reduce $\tilde{v}$. The values γ and δ are positive parameters. While γ is kept constant during all the execution of the algorithm, δ is dynamically updated. The value δ is initialized to 1000, and then at each iteration, if the number of solutions for which $\tilde{v} > K$ exceeds half of the ants population, then δ is doubled, otherwise it is halved. The value δ is not allowed to descend below 10 and to exceed 10^9.

In the following the described adaptation of the standard savings based ACO procedure (ANTALGORITHM given in Algorithm 2) is denoted as ANTALGORITHM'.

As observed in Reimann et al. [3] problem instances with more than 100 customers should be divided into several subproblems (see Algorithm 2). Each subproblem has approximately the same number of customers, λ. The number of clusters is set to n/λ rounded up or down to the next integer. Due to computational evidence we chose $\lambda = 30$ (see Section 6.1). The clustering is achieved by solving the complete instance according to the ACO described above with one iteration and 10 ants, so as to obtain a *master* solution. If no feasible master solution is found, then the process is re-iterated. In our tests three iterations were always enough to provide a feasible master solution. The used decomposition approach was inspired by Taillard et al. [27].

Algorithm 3 Main Algorithm

```
1:  procedure MAIN(V, λ, κ)
2:      if n > 100 then
3:          if n/λ − ⌊n/λ⌋ < 0.5 then t := ⌊n/λ⌋          ▷ Number of Clusters
4:          else t := ⌈n/λ⌉
5:          end if
6:          T := 1; P := 10; Π := max{|V|, 50}/4
7:          ANTALGORITHM'(V, T, P, Π)          ▷ Master Solution
8:          for k := 0, k < κ, k := k + 1 do          ▷Clustering Steps
9:              Compute the centers of gravity for the routes in the best solution found
10:             Sort the routes by their centers of gravity through the Sweep Algorithm
11:             Cluster the problem in t clusters (C_1, ..., C_t)
12:             for l := 0, l < t, l := l + 1 do
13:                 ñ = max{50, |C_l|}; T := 2ñ; P := ñ/2; Π := ñ/4
14:                 ANTALGORITHM'(C_l, T, P, Π)
15:             end for
16:             Combine partial solutions of clusters into a new candidate global solution
17:         end for
18:     else
19:         ñ := max{50, |V|}; T := 2ñ; P := ñ/2; Π := ñ/4
20:         ANTALGORITHM'(V, T, P, Π)
21:     end if
22: end procedure
```

The centers of gravity of the routes of the master solution are calculated and used within a Sweep Algorithm (see, e.g., [14]) to sort the routes. In Fig. 4 we give an example with 15 customers and four routes, where the triangles denote

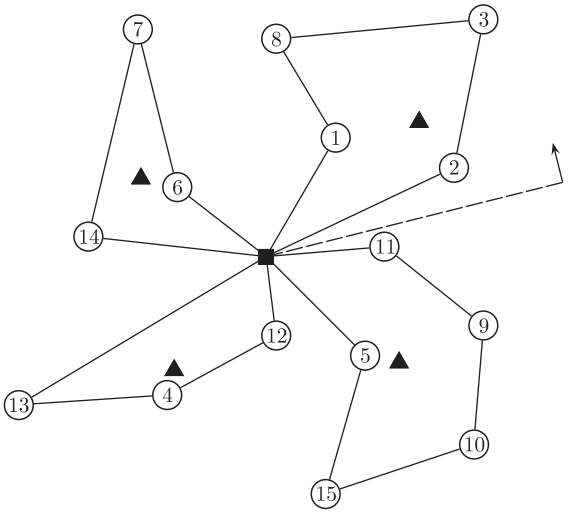

Fig. 4. Clustering of an instance through the Sweep Algorithm.

Table 1
Best parameter setting for the ACO ($\tilde{n} = \max\{50, |V|\}$)

Number of ACO iterations (T)	See Algorithm 3	Weight of pheromone (α)	5
Number of ants (P)	See Algorithm 3	Weight of savings (β)	5
Neighborhood size (Π)	See Algorithm 3	Weight of capacity penalty (γ)	100
Number of elitists (F)	6	Update constant (ε)	0.0005
Initial pheromone (τ_0)	2	Trail persistency (ρ)	0.95

the center of gravities of each route. In the following step the number of routes for each cluster is determined and a random starting route is chosen to start the Sweep Algorithm. In the example in Fig. 4 the route (0, 2, 3, 8, 1, 0) is chosen. The number of clusters is $t = 2$ and the first cluster consists of the routes (0, 2, 3, 8, 1, 0) and (0, 6, 7, 14, 0), while the second consists of the routes (0, 13, 4, 12, 0) and (0, 5, 15, 10, 9, 11, 0). After assigning the routes to the clusters, each cluster is independently optimized by the proposed ACO. The ACO parameter configuration remains the same one used in the unclustered instances, with a maximum number of T iterations allowed on each sub-instance.

In order to save computation time the master problem with all customers is only solved at the beginning of the main algorithm. Instead, the sweeping step and the ACO optimization step are re-iterated: the routes of the current best solution are again sorted by their centers of gravity, the solution is re-clustered and re-solved until a certain number of clustering steps $\kappa = 10$ has been reached. To guarantee a clustering in different regions with different customers, once a random starting route is chosen, then it is kept tabu for the next three clustering steps.

The overall algorithm has two different stopping conditions:

- T iterations have been performed in the unclustered case or the problem has been clustered $\kappa = 10$ times (see Table 1 for more details);
- a total runtime limit of 3 CPU hours has been reached.

6. Computational results

All solution procedures were coded using ANSI C++ and compiled using the Linux g++ compiler. All our runs were performed on a Pentium 4 with 3.2 GHz under the Linux operating system. The ACO was tested with a unique parameters configuration, described in Section 6.1.

The algorithm was tested on the classical 2L-CVRP instances, that can be downloaded from `http://www.or.deis.unibo.it/research.html` and were already used in [1,2,5,6]. In these instances the cost matrix

and the weights demanded by the customers were obtained from classical CVRP instances (see [28]). The number of items and their dimensions were created according to five classes. Class 1 is a pure CVRP instance, created so that the two-dimensional loading aspect is never tight. In class i, with $i = 2, \ldots, 5$, the number of items assigned to each customer is obtained with a uniform distribution between 1 and i. The dimensions of each item are uniformly generated in given intervals. For further details we refer to [6]. In total 36×5 instances were created, containing between 15 and 255 customers and between 15 and 786 items. Due to this large number of instances, in the following we report average results instead of results for each single instance. This also allows us to gain some insight in the complexity of the problem addressed. We refer the reader to the technical report [29] available at `http://www.univie.ac.at/bwl/prod/research/VRPandBPP` for a detailed description of the results for each instance. This web page contains the detailed computational results, the best solution value found for each instance and each 2L configuration, and a clear graphical representation of each of the solutions found.

The first 17×5 instances contain up to 40 customers and 127 items, and were addressed by means of an exact branch-and-cut in [5]. The branch-and-cut works on a problem definition in which the costs of the arcs are rounded down to the next integer, one-customer routes are not allowed, exactly K vehicles have to be used and a 2|RO|L is considered. For the majority of these instances (which will be referred to as *small-size* instances in the following) the optimal solutions are known. We present our results for this test bed in Section 6.2.

For larger size instances, with more than 40 customers, only heuristic solutions are available. The complete set of instances was addressed by means of tabu search in [6]. In this test, real costs on the arcs were used, one-customer routes were allowed, and less than K vehicles could be used. The loading could be either 2|RO|L or 2|UO|L. We compare our algorithm to these solutions in Section 6.3. Finally in Section 6.4 we evaluate the effect of the four different loading constraints on the solution values.

6.1. Parameter setting

All the tests were performed with the same configuration of the ACO. The algorithm was run 10 times on each instance, each run with a different random seed. Based on computational evidence, the ACO parameters were set to the values in Table 1, where $\tilde{n} = \max\{50, n\}$. This parameter choice is consistent with the results obtained in previous implementations of Savings-based ACO, as in Reimann et al. [4] and Doerner et al. [11].

Concerning the loading routines, at each decision step of an ant, (i.e., each time a merging of two partial routes is attempted) a feasible loading is searched by means of the procedure described in Algorithm 1. According to the outcome of preliminary testing, the procedure is given a maximum number ϕ of local search iterations equal to $5|r|$, with $|r|$ being the number of customers in the route to be checked. The branch-and-bound is allowed 1000 backtrackings or a time limit of 10 CPU seconds. Also this parameter choice was motivated by computational evidence.

The loading routines are not only used to provide feasible loadings for a route, but also for evaluating the dimensions of the virtual rectangles (see Section 5.2). The choice of using this rectangle as a measure of the efficiency of the loading is quite new. Another option that was tested is to use the classical *envelope* (or *contour*) as defined in Martello et al. [8] and Scheithauer [30]. This other option was disregarded since it led to worse computational results here.

The clustering was applied by dividing instances with more than 100 customers into sub-instances with λ customers. We tested λ values from 20 to 60 and found out that there are no significant performance differences, i.e., the algorithm is very robust with respect to this parameter. The results in the next subsections are based on $\lambda = 30$. Clustering also smaller instances with $50 < n < 100$ customers was also tested, but did not provide better results. The choice of setting $\tau_0 = 2$ as initial pheromone concentration on each edge was motivated by the extensive tests in [3,4].

6.2. Results for small-size instances

In Table 2 we present the results obtained by running the ACO on the 85 small-size instances, and compare them with the branch-and-cut in [5] and the tabu search in [6]. Here only the 2|RO|L case is considered because for the other cases no benchmark results are available. In this table all algorithms used the integer distances. Column I gives the number of the original CVRP instance, as used in [6]. Column n reports the number of customers in the CVRP instance. Each line gives the average value over the five 2L-CVRP instances created according to the five classes used to generate the items dimensions.

Table 2
Aggregate results for 2L-CVRP small-size instances (averages over five instances per line)

I	n	Branch-and-cut [5] (Pentium IV, 1.7 GHz)				Tabu search [6] (Pentium IV, 1.7 GHz)						ACO (Pentium IV, 3.2 GHz)							
		z	sec_h	sec_{tot}	opt	z	sec_h	sec_{tot}	$\%gap$	opt	$impr$	$z_{\min}$	z_{avg}	$z_{\max}$	sec_h	sec_{tot}	$\%gap$	opt	$impr$
1	15	281.0	25.4	29.9	5	281.4	0.8	7.6	0.14	4	0	285.0	286.4	288.0	5.2	8.9	1.82	3	0
2	15	336.6	13.9	14.7	5	337.2	0.4	3.9	0.17	4	0	337.2	337.4	339.0	0.2	0.7	0.22	4	0
3	20	375.4	18.6	25.4	5	377.6	5.1	18.0	0.93	4	0	375.4	376.6	377.2	2.9	7.0	0.32	5	0
4	20	430.0	11.6	17.0	5	433.8	1.5	14.0	0.87	3	0	431.0	431.0	431.0	1.9	4.1	0.23	4	0
5	21	377.2	597.7	597.8	5	387.0	3.8	28.8	2.59	2	0	377.2	378.0	379.2	11.0	19.5	0.20	5	0
6	21	490.4	62.4	67.4	5	494.6	2.8	23.1	0.86	3	0	490.4	491.7	493.6	2.4	5.6	0.27	5	0
7	22	687.2	674.2	676.3	5	699.8	12.7	39.2	1.76	1	0	689.0	690.0	693.4	3.5	11.9	0.40	3	0
8	22	705.8	256.4	261.6	5	717.4	10.8	45.0	1.62	2	0	708.6	713.1	728.0	10.1	16.6	1.03	4	0
9	25	614.2	304.8	469.9	5	616.6	5.4	38.2	0.39	3	0	614.4	616.0	618.4	5.4	8.3	0.29	4	0
10	29	675.8	22559.1	51161.0	3	684.0	40.7	150.8	1.58	0	1	664.0	666.9	672.8	37.6	48.6	−1.40	1	1
11	29	725.6	42684.4	69120.4	1	718.4	110.7	175.7	−0.88	1	2	688.0	691.7	695.4	117.1	134.0	−4.51	1	4
12	30	609.4	39002.8	86400.4	0	612.0	48.3	87.3	0.41	0	2	600.6	602.3	604.8	9.9	12.9	−1.16	0	5
13	32	2713.6	26502.5	58268.0	2	2588.4	95.9	296.2	−3.27	1	2	2533.6	2540.7	2552.8	48.3	59.0	−6.36	2	3
14	32	1233.4	50872.6	74228.8	1	1157.8	84.7	343.3	−5.09	0	4	1143.8	1146.0	1149.4	109.2	131.9	−6.93	0	4
15	32	1252.6	26353.5	69124.2	1	1191.6	220.7	308.0	−4.50	1	3	1179.0	1182.4	1186.2	205.4	235.2	−6.02	1	4
16	35	683.8	8582.7	10098.1	5	686.4	35.3	161.8	0.38	4	0	684.2	684.9	686.2	7.2	11.1	0.15	4	0
17	40	854.6	359.9	86400.3	0	844.4	46.4	234.7	−1.19	0	5	843.8	846.2	848.2	3.3	9.4	−0.99	0	5
AVG		767.4	12875.4	29821.2	58	754.6	42.7	116.2	−0.21	33	19	743.8	746.0	749.6	34.2	42.6	−1.32	46	26

Integer costs, no one-customer routes allowed, fixed number of vehicles, 2|RO|L.

For the branch-and-cut we present the average solution value (z), the average time in seconds in which this solution was found (sec_h), the average time required by the algorithm to run to completion (sec_{tot}) and the total number of proven optima found (opt). For the tabu search, we also give the percentage gap ($\%gap$) with respect to the branch-and-cut solution value and the number of times in which the metaheuristic solution was better than the one (sub-optimal) found by the branch-and-cut ($impr$). For the ACO, being a random metaheuristic, we performed 10 runs. In the table we give the smallest, average and highest solution value found over the 10 runs $z_{\min}$, z_{avg} and $z_{\max}$. We compute sec_h and sec_{tot} as averages over the five classes and on the 10 runs.

The average percentage gap to the branch-and-cut is computed with respect to z_{avg}, while the total numbers of optima and of improvements are evaluated with respect to $z_{\min}$. The values provided for branch-and-cut and tabu search are directly taken from [6].

Within a time limit of 24 CPU hours on a Pentium IV with 1.7 GHz, the branch-and-cut could solve to optimality all instances with up to 25 nodes, but only a limited number of the bigger ones. The tabu search could obtain solutions equivalent to the optimal ones 33 times, and 19 times managed to improve sub-optimal solutions. This was done in almost two CPU minutes on the same Pentium IV with 1.7 GHz, leading to an average gap of −0.21%. The ACO clearly outperforms the tabu search, since in less than a CPU minute (even if on a faster computer) produces an average percentage gap of −1.32%, obtaining 46 optima and 26 improvements. Even when considering the worst solution value found on the 10 runs for each instance ($z_{\max}$), the ACO is on average better than the tabu search, leading to an average value of 749.6, against 754.6.

6.3. Results for the complete set of instances

In Table 3 we present the results of the ACO for the complete set of 180 instances, and compare them with the tabu search results in [6]. Note that in this case the original non integer distances from the classical CVRP instances are used. The columns have the same meanings as in Table 2, with the exception of column $\%gap$, which now gives the average gap between the average solution value found by the ACO and the solution value found by the tabu search, computed as $100(z_{\text{avg}} - z)/z$. The values are again computed as averages over five instances per line.

Table 3
Aggregate results for the complete set of instances with 2|RO|L (averages over five instances per line)

I	n	Tabu search [6] (Pentium IV, 1.7 GHz)			ACO (Pentium IV, 3.2 GHz)					
		z	sec_h	sec_{tot}	$z_{\min}$	z_{avg}	$z_{\max}$	sec_h	sec_{tot}	$\%gap$
1	15	295.01	2.6	9.2	291.33	291.83	294.15	5.5	7.2	−1.03
2	15	343.18	0.4	3.5	343.18	343.22	343.38	0.3	0.6	0.01
3	20	380.19	3.8	18.9	376.80	378.17	379.50	2.3	3.1	−0.51
4	20	440.91	1.4	17.0	439.06	439.07	439.06	2.3	3.0	−0.41
5	21	382.30	4.1	27.6	380.97	381.63	382.97	9.7	12.8	−0.17
6	21	501.40	5.1	19.5	498.76	499.77	502.05	3.3	4.3	−0.32
7	22	691.23	15.5	53.0	675.53	678.22	683.28	9.2	11.7	−1.79
8	22	691.89	32.8	83.7	683.03	684.37	685.55	9.6	16.1	−1.02
9	25	620.77	10.6	40.0	613.16	614.88	617.33	4.0	4.9	−0.93
10	29	679.68	43.5	179.6	664.92	668.48	674.23	47.8	50.1	−1.64
11	29	719.76	99.0	199.4	683.89	690.22	697.87	55.1	57.8	−3.51
12	30	627.59	58.8	99.5	614.61	615.90	619.87	6.5	7.4	−1.77
13	32	2550.89	49.0	312.8	2474.68	2480.04	2491.86	58.9	61.8	−2.60
14	32	1048.72	146.0	439.5	1001.40	1007.92	1014.47	154.9	163.1	−3.69
15	32	1160.25	165.4	313.4	1137.88	1145.96	1151.66	133.9	138.8	−1.17
16	35	703.60	28.0	157.2	700.74	701.09	702.34	6.6	8.3	−0.35
17	40	865.72	88.9	226.2	863.60	864.92	866.89	4.1	7.2	−0.09
18	44	1037.65	566.5	1167.8	999.22	1003.84	1008.71	285.2	292.6	−3.03
19	50	746.91	365.2	1521.5	724.19	728.89	735.66	118.6	122.1	−2.15
20	71	513.84	808.9	3370.3	481.72	484.23	487.84	1057.4	1073.8	−4.95
21	75	1025.79	1702.2	3561.2	977.44	987.54	999.99	1027.3	1037.3	−3.40
22	75	1052.39	1573.8	3461.8	1009.85	1018.76	1029.07	726.8	738.9	−2.89
23	75	1121.18	675.8	3600.0	1039.50	1051.16	1066.33	1197.3	1206.5	−5.71
24	75	1208.52	2642.5	3324.6	1126.69	1134.90	1143.55	312.6	323.4	−5.62
25	100	1350.56	2336.5	3600.1	1296.80	1309.98	1323.15	2424.5	2454.9	−2.74
26	100	1341.30	1554.6	3600.3	1289.52	1306.24	1320.40	2370.4	3558.1	−2.52
27	100	1439.37	1308.2	3600.0	1331.61	1341.25	1352.45	1536.8	1570.3	−6.30
28	120	2502.48	2576.9	3600.1	2367.33	2417.89	2460.27	8349.4	8714.5	−2.16
29	134	2296.03	1162.5	3600.2	2078.55	2131.54	2194.25	8180.4	8837.5	−6.02
30	150	1873.27	2021.4	3600.2	1717.05	1734.46	1768.06	8267.0	8720.3	−6.90
31	199	2366.54	2102.2	3600.5	2182.04	2219.34	2258.18	8512.6	8747.4	−6.27
32	199	2354.60	2305.2	3600.6	2158.79	2191.97	2241.74	8687.9	8745.8	−6.21
33	199	2360.74	2221.2	3600.6	2214.52	2245.46	2287.48	8631.9	8742.7	−4.45
34	240	1408.64	2184.4	3601.0	1143.76	1160.98	1177.64	8645.4	8771.6	−17.51
35	252	1786.93	2223.1	3600.2	1439.48	1465.85	1497.24	8822.0	8942.2	−16.00
36	255	1693.10	2626.3	3600.9	1586.03	1603.86	1618.74	8978.6	9011.7	−5.46
AVG		1171.75	936.5	1817.0	1100.21	1111.77	1125.48	2462.4	2560.3	−3.65

Out of 36 cases, just in one small case the tabu search finds better results than ACO. In all other cases ACO has a negative $\%gap$, leading to a maximum improvement of 17.51% and to an average improvement of 3.65%. Considering the single instances, only in 18 cases out of 180 the tabu search could find a solution of better quality than the ACO. Even considering $z_{\max}$, ACO leads to an average improvement of 2.74%.

This improvement in the solution quality is obtained by elapsing a larger CPU time: ACO needs on average approximately 42 CPU minutes against the 30 required by the tabu search. However, if we allowed the same CPU time limit of one hour used for the tabu search, the ACO again could provide better results. In such a case the average improvement is 3.04% (against 3.65%), and the tabu search outperforms ACO on 30 instances out of 180 (against 18). Allowing a CPU time limit of 30 min to the ACO (for compensating the differences in the speeds of the computers used), our algorithm would obtain a consistent average improvement of 2.56% and an improvement of 1.54% based on the worst ACO solutions with respect to the tabu search.

In Table 4 we present the results of ACO and tabu search averaged by class (i.e., by the way in which the items numbers and dimensions were created). The columns have the same meanings of the ones in Table 3, but the values

Table 4
Aggregate results for the complete set of instances with 2|RO|L (averages over 36 instances per line)

Class	Tabu search [6] (Pentium IV, 1.7 GHz)			ACO (Pentium IV, 3.2 GHz)					
	z	sec_h	sec_{tot}	z_{min}	z_{avg}	z_{max}	sec_h	sec_{tot}	$\%gap$
1	792.31	772.9	1757.4	776.04	784.14	793.72	210.8	236.6	−0.89
2	1290.19	867.5	1670.4	1191.58	1202.93	1217.64	3157.8	3252.9	−5.43
3	1273.77	902.8	1789.4	1202.35	1213.36	1229.30	2937.8	3018.2	−3.62
4	1339.18	1021.5	1836.3	1237.60	1255.82	1274.20	3011.6	3207.8	−4.62
5	1163.28	1117.6	2031.5	1093.49	1102.62	1112.54	2993.9	3085.9	−3.68
AVG	1171.75	936.5	1817.0	1100.21	1111.77	1125.48	2462.4	2560.3	−3.65

Table 5
Aggregate results on the complete set of instances (averages over 180 instances per line) with two different loading configurations evaluated with different runtime limits

Loading	sec_{max}	Tabu search [6] (Pentium IV, 1.7 GHz)			ACO (Pentium IV, 3.2 GHz)					
		z	sec_h	sec_{tot}	z_{min}	z_{avg}	z_{max}	sec_h	sec_{tot}	$\%gap$
2|RO|L	1800				1120.22	1135.80	1152.42	605.2	645.5	−2.56
2|UO|L	1800				1073.11	1082.00	1092.33	530.9	539.8	−3.20
2|RO|L	3600	1171.75	936.5	1817.0	1111.56	1125.81	1142.57	1040.2	1067.7	−3.04
2|UO|L	3600	1131.33	757.9	1822.2	1070.46	1078.05	1087.95	892.5	902.0	−3.38
2|RO|L	7200				1104.08	1116.62	1132.24	1787.3	1815.4	−3.44
2|UO|L	7200				1067.21	1074.58	1083.64	1601.0	1611.9	−3.54
2|RO|L	10800				1100.21	1111.77	1125.48	2462.4	2560.3	−3.65
2|UO|L	10800				1066.25	1073.20	1081.82	2165.3	2285.2	−3.60

in the table are now averages over 36 instances per line. This allows us to provide additional considerations on the effect of the loading on the overall transportation problem. The average improvement with respect to the tabu search that we observe in Table 4 is well distributed among the five different classes. The smallest improvement is found in Class 1, in which the $\%gap$ is −0.89%, and the best improvement is found in Class 2, in which the $\%gap$ is −5.43%. It is important to note that Class 2 is the most difficult one from the loading point of view, while Class 1 is the easiest (simple CVRP). Hence, we can conclude that ACO performs particularly well on difficult instances.

Even better results were obtained by comparing ACO and tabu search on the 2|UO|L case. ACO outperformed the tabu search on 34 cases out of 36, with an average improvement of 3.6% and a maximum improvement of 18.24%. Out of the 180 single instances, the tabu search could outperform ACO only in 22 cases. The worst solution found by ACO on the 10 runs (z_{max}) was on average 3.02% better than the one found by the tabu search. We refer again to [29] for all the detailed results for this comparison.

The comparison we studied with the Tabu Search in [6] (which, we remind, was allowed only one hour on a Pentium IV 1.7 GHz) did not take into detailed account so far the difference in the time limit allowed. A fair comparison is the one in which we evaluate the ACO performance under a time limit of 1800 CPU seconds (so as to compensate the difference in the computers speeds). Such a comparison is shown in Table 5,where sec_{max} denotes the maximum CPU time allowed to each algorithm. Also in this case we see how the ACO clearly outperforms the Tabu Search, obtaining an average improvement of 2.56% for 2|RO|L and 3.20% for 2|UO|L. Also the worst solutions found by the ACO are on average better than the ones found by the Tabu: the percentage improvement is 1.54% for 2|RO|L and 2.53% for 2|UO|L.

Apart from that, Table 5 additionally shows how the time limit of 3 CPU hours allows the ACO to produce better results than the ones obtained with reduced execution times. This limit turned out to be the best compromise between execution time and solution quality, since allowing larger CPU time led to only limited improvements.

Table 6
Aggregate results for the complete set of instances (averages over five instances per line) with the four different loading configurations

I	n	2|RO|L			2|UO|L				2|RN|L				2|UN|L			
		z_{min}	z_{avg}	z_{max}	z_{min}	z_{avg}	z_{max}	$\%gap$	z_{min}	z_{avg}	z_{max}	$\%gap$	z_{min}	z_{avg}	z_{max}	$\%gap$
1	15	291.33	291.83	294.15	284.36	284.82	285.50	−2.40	281.11	281.40	281.83	−3.57	280.67	280.76	280.78	−3.79
2	15	343.18	343.22	343.38	339.81	339.81	339.81	−0.99	339.81	339.81	339.81	−0.99	339.81	339.81	339.81	−0.99
3	20	376.80	378.17	379.50	372.73	372.73	372.73	−1.44	373.00	373.00	373.00	−1.37	370.02	371.16	371.77	−1.86
4	20	439.06	439.07	439.06	437.10	437.11	437.10	−0.45	434.18	435.26	437.10	−0.87	434.18	434.19	434.18	−1.11
5	21	380.97	381.63	382.97	378.28	378.28	378.28	−0.88	377.93	378.99	380.92	−0.69	377.93	377.93	377.93	−0.97
6	21	498.76	499.77	502.05	496.99	497.02	497.06	−0.55	497.27	497.40	497.97	−0.48	496.81	496.85	497.06	−0.59
7	22	675.53	678.22	683.28	671.24	671.36	671.75	−1.01	670.69	671.13	671.86	−1.05	664.51	664.51	664.51	−2.02
8	22	683.03	684.37	685.55	666.62	666.62	666.62	−2.59	665.79	668.50	670.01	−2.32	656.71	656.98	657.25	−4.00
9	25	613.16	614.88	617.33	611.15	611.15	611.15	−0.61	611.15	611.72	614.34	−0.51	611.15	611.28	611.82	−0.59
10	29	664.92	668.48	674.23	653.18	655.55	662.36	−1.93	650.90	656.20	661.88	−1.84	643.96	645.72	647.54	−3.41
11	29	683.89	690.22	697.87	678.46	678.51	678.62	−1.70	670.34	676.06	680.51	−2.05	659.60	662.97	665.97	−3.95
12	30	614.61	615.90	619.87	611.01	612.74	615.82	−0.51	613.11	615.16	618.65	−0.12	610.89	612.73	615.63	−0.51
13	32	2474.68	2480.04	2491.86	2417.85	2424.14	2425.01	−2.25	2412.21	2420.00	2429.90	−2.42	2375.82	2376.06	2377.40	−4.19
14	32	1001.40	1007.92	1014.47	960.54	966.55	976.68	−4.10	961.56	965.63	971.33	−4.20	947.38	952.16	955.59	−5.53
15	32	1137.88	1145.96	1151.66	1101.36	1105.76	1116.94	−3.51	1099.83	1104.71	1110.39	−3.60	1073.52	1081.99	1086.57	−5.58
16	35	700.74	701.09	702.34	699.56	699.92	700.47	−0.17	699.56	699.86	701.08	−0.18	699.56	699.55	699.56	−0.22
17	40	863.60	864.92	866.89	863.60	864.62	865.18	−0.03	861.79	862.69	863.28	−0.26	862.25	862.63	863.15	−0.26
18	44	999.22	1003.84	1008.71	979.36	984.70	988.18	−1.91	959.87	963.71	969.76	−4.00	954.46	955.83	958.56	−4.78
19	50	724.19	728.89	735.66	706.56	707.89	709.53	−2.88	695.11	704.15	710.50	−3.39	686.49	689.59	692.35	−5.39
20	71	481.72	484.23	487.84	470.93	472.45	475.67	−2.43	462.63	465.73	469.17	−3.82	455.34	456.35	458.18	−5.76
21	75	977.44	987.54	999.99	950.08	952.79	956.70	−3.52	946.56	952.53	958.81	−3.55	929.43	933.71	939.53	−5.45
22	75	1009.85	1018.76	1029.07	984.18	988.21	994.24	−3.00	971.85	980.24	987.96	−3.78	955.40	958.74	963.27	−5.89
23	75	1039.50	1054.69	1073.69	1006.85	1012.19	1022.29	−4.03	1002.88	1009.97	1018.22	−4.24	988.47	995.32	1001.30	−5.63
24	75	1126.69	1134.90	1143.55	1103.81	1109.38	1117.43	−2.25	1098.73	1109.90	1120.35	−2.20	1084.93	1090.91	1099.83	−3.88
25	100	1306.40	1320.94	1336.02	1258.78	1266.07	1272.83	−4.15	1247.24	1258.28	1266.90	−4.74	1225.23	1232.04	1238.56	−6.73
26	100	1289.52	1309.67	1326.29	1240.89	1245.32	1248.72	−4.91	1237.38	1246.94	1254.94	−4.79	1215.38	1220.37	1225.79	−6.82
27	100	1331.61	1341.25	1352.45	1293.87	1303.41	1310.41	−2.82	1295.40	1302.21	1310.44	−2.91	1267.85	1273.93	1282.60	−5.02
28	120	2403.48	2468.98	2544.92	2316.54	2363.21	2414.59	−4.28	2305.36	2362.07	2413.75	−4.33	2248.82	2290.36	2332.19	−7.23
29	134	2127.45	2187.90	2258.62	2031.05	2061.22	2109.15	−5.79	2022.45	2054.10	2111.23	−6.12	1980.32	2001.96	2037.46	−8.50
30	150	1753.81	1796.54	1857.16	1649.84	1665.76	1685.62	−7.28	1658.36	1672.35	1693.52	−6.91	1607.11	1620.49	1640.06	−9.80
31	199	2239.43	2282.91	2331.43	2122.06	2141.48	2170.56	−6.20	2127.83	2150.15	2185.08	−5.82	2058.64	2078.40	2101.48	−8.96
32	199	2210.13	2260.37	2313.86	2076.44	2101.41	2140.83	−7.03	2092.87	2113.43	2143.05	−6.50	2020.23	2040.34	2060.50	−9.73
33	199	2267.95	2315.51	2380.18	2136.65	2158.10	2186.03	−6.80	2134.66	2163.05	2196.77	−6.58	2062.42	2084.64	2108.63	−9.97
34	240	1180.02	1198.04	1216.41	1102.98	1112.06	1124.63	−7.18	1107.65	1116.34	1127.89	−6.82	1068.17	1075.71	1086.26	−10.21
35	252	1489.24	1516.70	1542.43	1338.19	1354.65	1380.29	−10.68	1355.11	1370.23	1386.29	−9.66	1291.06	1304.47	1319.26	−13.99
36	255	1615.08	1632.44	1651.21	1527.54	1543.62	1557.29	−5.44	1533.01	1549.77	1567.76	−5.06	1477.20	1489.36	1501.97	−8.76
AVG		1111.56	1125.83	1142.67	1070.57	1078.07	1087.95	−3.27	1068.75	1077.85	1088.79	-3.38	1046.71	1053.33	1060.95	−5.06

Table 7
Aggregate results on the complete set of instances (averages over 180 instances per line) with the four different loading configurations

Loading	$z_{\min}$	z_{avg}	$z_{\max}$	sec_h	sec_{tot}
2|RO|L	1100.21	1111.77	1125.48	2462.4	2560.3
2|UO|L	1066.25	1073.20	1081.82	2165.3	2285.1
2|RN|L	1063.80	1071.84	1080.98	2247.9	2352.4
2|UN|L	1044.51	1050.55	1057.28	1818.1	1950.2

6.4. Evaluating the effect of loading constraints

A question of practical relevance is the effect of a loading constraint with respect to the complexity of the overall loading and routing problem, and with respect to the performance of the algorithm. In Table 6 we present the results obtained by running the ACO on the 4 different loading configurations presented in Section 2. For the sake of conciseness we report only four values: the minimum solution value $z_{\min}$, the average solution value z_{avg}, the maximum solution value $z_{\max}$ of the 10 runs (computed as in the previous table as average of the 10 runs and of five instances per line), and the percentage gap (*%gap*). In this case *%gap* is evaluated as the percentage gap between the z_{avg} value of a given loading configuration and the z_{avg} value of the most constrained loading (2|RO|L) obtained by ACO.

As one would expect, the average solution value decreases when the rear loading constraint is relaxed (-3.27%) and when the fixed orientation constraint is relaxed (-3.38%). When both constraints are removed, further improvement is possible, reaching an average of -5.06%. What is interesting is that this improvement is well distributed among the different groups of instances. In none of the 36 cases, relaxing a constraint leads to a worse solution, which could happen when applying heuristics.

The relaxation of the rear loading and of the fixed orientation constraints seem to have similar effect on the reduction of the transportation cost, and none of them leads to a clearly lower solution cost than the other. Out of the 180 instances, the z_{avg} value with 2|UO|L was better than the one with 2|RN|L in 62 cases, worse in 63 cases and equal in 55 cases.

Decreasing of the solution cost when the loading becomes looser can be noted also for the best and for the worst solution value obtained on the 10 runs, as reported in Table 7.The table reports on each line average values on the complete set of 180 instances. The values of $z_{\min}$ and $z_{\max}$ decrease consistently when removing constraints. This fact highlights the importance in distribution of allowing rotation of the items, which, in this particular set of instances, would allow a distribution company to save almost 3% in the transportation costs. Also allowing rearrangements of the items within the vehicle and along the route can lead to an important saving of more than 3%. This can influence the decision of a company, but has to be compared with the risk of damaging items during the rearrangements.

Table 7 is also of interest for evaluating the behavior of the ACO. When loading constraints are relaxed, the CPU time needed by the algorithm to find feasible loadings is decreased, and this leads to a consistent improvement in the CPU time required by the overall algorithm. In the most relaxed configuration (2|UR|L), the ACO reaches the termination condition in an average CPU time (sec_{tot}) of 32 min, which is 10 minutes less than what is needed for the most constrained loading (2|SO|L). Also the CPU time required to reach the best heuristic solution (sec_h) is consistently reduced form an average value of 41 CPU minutes for 2|SO|L, to 30 min for 2|UR|L.

7. Conclusions

We have considered an extension of the classical vehicle routing problem in which two-dimensional loading constraints are introduced. The problem is of interest because of its theoretical complexity and of the many real-world applications. We have developed an ACO-based algorithm and have applied it to all available benchmark data from the literature.

Our contribution is threefold. First, we obtain very good results for these benchmark instances. On small-size instances the ACO provides low gaps from the proven optimal solutions, reaching 46 proven optima out of 58. On the complete set of instances the ACO clearly outperforms previous heuristics from the literature.

The second main contribution of the paper concerns the managerial implications. For the first time in the literature, we compare the cost of four different loading configurations. We explicitly compare rear loading to unconstrained loading and cases where the items can be rotated to those with fixed orientation. We observe that on average over 180 instances about 5% of total transportation cost can be saved when both restrictions are relaxed. If only one restriction (i.e., no rotation or rear loading) is relaxed, still more than 3% can be saved. It is important to observe that the savings are higher as the problem instances increase in dimension.

Third, in ACO a combination of different heuristic information usually did not turn out to be successful in the past. Our approach provides an example where an ACO algorithm successfully combines two completely different heuristic measures (with respect to loading and routing) within one pheromone matrix.

Acknowledgements

Financial support from the Oesterreichische Nationalbank (OENB) under grant #11984, from the Fonds zur Förderung der wissenschaftlichen Forschung (FWF) under grant #L286-N04, and from the Ministero dell'Istruzione, dell'Università e della Ricerca (MIUR) is gratefully acknowledged.

References

[1] Iori M. Metaheuristic algorithms for combinatorial optimization problems. PhD thesis, University of Bologna, Italy, 2004.
[2] Iori M. Metaheuristic algorithms for combinatorial optimization problems. 4OR 2005;3:163–6.
[3] Reimann M, Doerner K, Hartl RF. D-ants: savings based ants divide and conquer the vehicle routing problem. Computers & Operations Research 2004;31(4):563–91.
[4] Reimann M, Stummer M, Doerner KF. A savings based ant system for the vehicle routing problem. In: Proceedings of the genetic and evolutionary computation conference 2002. Los Altos, CA: Morgan Kaufmann; 2002. p. 1317–25.
[5] Iori M, Salazar González JJ, Vigo D. An exact approach for the vehicle routing problem with two-dimensional loading constraints. Transportation Science 2007;41(2):253–64.
[6] Gendreau M, Iori M, Laporte G, Martello S. A tabu search heuristic for the vehicle routing problem with two-dimensional loading constraints. Networks 2007, to appear.
[7] Martello S, Vigo D. Exact solution of the two-dimensional finite bin packing problem. Management Science 1998;44:388–99.
[8] Martello S, Pisinger D, Vigo D. The three-dimensional bin packing problem. Operations Research 2000;48:256–67.
[9] Gendreau M, Hertz A, Laporte G. A tabu search heuristic for the vehicle routing problem. Management Science 1994;40:1276–90.
[10] Gendreau M, Iori M, Laporte G, Martello S. A tabu search algorithm for a routing and container loading problem. Transportation Science 2006;40:342–50.
[11] Doerner K, Fuellerer G, Gronalt M, Hartl R, Iori M. Metaheuristics for vehicle routing problems with loading constraints. Networks 2007;49(4):294–307.
[12] Moura A, Oliveira JF. An integrated approach to vehicle routing and container loading problems. In EURO XX—20th European conference on operational research, Rhodes, Greece, 2004.
[13] Moura A, Oliveira JF. Uma heurística composta para a determinação de rotas para veículos em problemas com janelas temporais e entregas e recolhas. Investigação Operacional 2004;24(1):45–62.
[14] Toth P, Vigo D. The vehicle routing problem. Philadelphia, PA: SIAM Monographs on Discrete Mathematics and Applications; 2002.
[15] Cordeau J-F, Gendreau M, Hertz A, Laporte G, Sormany J-S. New heuristics for the vehicle routing problem. In: Langevin A, Riopel D, editors. Logistics systems: design and optimization. Boston: Kluwer; 2005. p. 279–98.
[16] Cordeau J-F, Laporte G. Tabu search heuristics for the vehicle routing problem. In: Rego C, Alidaee B, editors. Metaheuristic optimization via memory and evolution: tabu search and scatter search. Boston: Kluwer; 2004. p. 145–63.
[17] Wäscher G, Haussner H, Schumann H. An improved typology of cutting and packing problems. European Journal of Operational Research 2007;183:1109–30.
[18] Martello S, Monaci M, Vigo D. An exact approach to the strip packing problem. INFORMS Journal on Computing 2003;15(3):310–9.
[19] Alvarez-Valdes R, Parreño F, Tamarit JM. Reactive GRASP for the strip-packing problem. Computers & Operations Research 2008;35(4):1065–83.
[20] Burke EK, Kendall G, Whitwell G. A new placement heuristic for the orthogonal stock-cutting problem. Operations Research 2004;52: 655–71.
[21] Dell'Amico M, Martello S. A lower bound for the non-oriented two-dimensional bin packing problem. Discrete Applied Mathematics 2002;118:13–24.
[22] Lodi A, Martello S, Vigo D. Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. INFORMS Journal on Computing 1999;11(4):345–57.
[23] Clarke G, Wright JV. Scheduling of vehicles from a central depot to a number of delivery points. Operations Research 1964;12:568–81.
[24] Gendreau M, Laporte G, Potvin J-Y. Vehicle routing: modern heuristics. In: Aarts E, Lenstra JK, editors. Local search in combinatorial optimization. Chichester: Wiley; 1997.

[25] Kindervater GAP, Savelsbergh MWP. Vehicle routing: handling edges exchanges windows. In: Aarts E, Lenstra JK, editors. Local search in combinatorial optimization. Chichester: Wiley; 1997. p. 337–60.
[26] Bullnheimer B, Hartl RF, Strauss C. A new rank based version of the ant system: a computational study. Central European Journal of Operations Research 1999;7:25–38.
[27] Taillard E, Badeau P, Gendreau M, Guertin F, Potvin JY. A tabu search heuristic for the vehicle routing problem with soft time windows. Transportation Science 1997;31:170–86.
[28] Christofides N, Mingozzi A, Toth P. The vehicle routing problem. In: Christofides N, Mingozzi A, Toth P, Sandi C, editors. Combinatorial optimization. Chichester: Wiley; 1979. p. 315–38.
[29] Doerner K, Fuellerer M, Hartl R, Iori M. Ant colony optimization for the two-dimensional loading vehicle routing problem: detailed results. Technical Report, POM, University of Vienna, available online at ⟨http://www.univie.ac.at/bwl/prod/research/VRPandBPP⟩, 2007.
[30] Scheithauer G. Equivalence and dominance for problems of optimal packing of rectangles. Ricerca Operativa 1997;83:3–34.