A Tabu Search heuristic for the vehicle routing problem with two-dimensional loading constraints

Michel Gendreau [†], Manuel Iori [‡], Gilbert Laporte [†,§], Silvano Martello [♯]

[†] *Centre de recherche sur les transports,*
Université de Montréal, C.P. 6128 succursale Centre-ville, Montréal, H3C 3J7 Canada
michelg@crt.umontreal.ca

[‡] *Dipartimento di Scienze e Metodi dell'Ingegneria,*
Università degli studi di Modena e Reggio Emilia, Viale Amendola 2, 42100 Reggio Emilia, Italy
manuel.iori@unimore.it

[§] *Canada Research Chair in Distribution Management,*
HEC Montréal, 3000 chemin de la Côte-Sainte-Catherine, Montréal, H3T 2A7 Canada
gilbert@crt.umontreal.ca

[♯] *Dipartimento di Elettronica, Informatica e Sistemistica,*
Università degli Studi di Bologna, Viale Risorgimento 2, 40136 Bologna, Italy
smartello@deis.unibo.it

Received September 2004; accepted October 2005.

Abstract

The article addresses the well-known *Capacitated Vehicle Routing Problem* (CVRP), in the special case where the demand of a customer consists of a certain number of two-dimensional weighted items. The problem calls for the minimization of the cost of transportation needed for the delivery of the goods demanded by the customers, and carried out by a fleet of vehicles based at a central depot. In order to accommodate all items on the vehicles, a feasibility check of the two-dimensional packing (2L) must be executed on each vehicle. The overall problem, denoted as 2L-CVRP, is NP-hard and particularly difficult to solve in practice. We propose a Tabu Search algorithm, in which the loading component of the problem is solved through heuristics, lower bounds and a truncated branch-and-bound procedure. The effectiveness of the algorithm is demonstrated through extensive computational experiments.

Keywords: Vehicle Routing, Two-dimensional Packing, Tabu Search.

1 Introduction

One of the most frequently studied combinatorial optimization problems is the *Capacitated Vehicle Routing Problem* (CVRP), which calls for the optimization of the delivery of goods, demanded by a set of clients, and operated by a fleet of vehicles of limited capacity based at a central depot. The application of this model to real world problems is limited by the presence of many additional constraints. In particular, in the CVRP client demands are expressed by an integer value, representing the total weight of the items to be delivered, while in real-world instances demands consist of lots of items characterized both by a weight and a shape.

In transportation it is often necessary to handle rectangular-shaped items that cannot be stacked one on top of the other because of their fragility or weight. This happens, for example, when the transported items are large kitchen appliances, such as refrigerators, or pieces of catering equipment, such as food trolleys. In such cases, the CVRP must contain additional constraints to reflect the two-dimensional loading feature of the problem.

The problem studied in this paper, which combines the feasible loading or unloading of the items into vehicles and the minimization of transportation costs, is particularly relevant for freight distribution companies. From an algorithmic point of view, the combination of these two areas of combinatorial optimization (vehicle routing and two-dimensional packing) is challenging. We consider two variants of the problem (see Section 2), one of which was first addressed by Iori, Salazar González and Vigo [22, 19, 20]. Following the notation of these authors, the problem will be denoted here as 2L-CVRP (*Two-Dimensional Loading Capacitated Vehicle Routing Problem*).

The vehicle routing problem has been deeply investigated since the seminal work of Dantzig and Ramser [9]. Branch-and-cut (-and-price) algorithms are nowadays the best exact approaches (see, e.g., Toth and Vigo [29] and Fukasawa, Longo, Lysgaard, Poggi de Aragão, Reis, Uchoa and Werneck [12]), but are able to systematically solve to optimality only relatively small instances. The problem of loading two-dimensional items into the vehicle is closely related to various packing problems. In particular:

- the *Two-Dimensional Bin Packing Problem* (2BPP): Pack a given set of rectangular items into the minimum number of identical rectangular bins. Exact approaches for the 2BPP are generally based on branch-and-bound techniques and are able to solve instances with up to 100 items (but some instances with 20 items remain unsolved). Recent exact algorithms and lower bounds have been proposed by Martello and Vigo [26], Fekete and Schepers [11], Boschetti and Mingozzi [1, 2] and Pisinger and Sigurd [27].

- the *Two-Dimensional Strip Packing Problem* (2SP): Pack a given set of rectangular items into a strip of given width and infinite height so as to minimize the overall height of the packing. An exact algorithm proposed by Martello, Monaci and Vigo [24] can solve instances with up to 200 items, although some instances with 30 items are still unsolved.

Both the CVRP and the 2BPP are strongly NP-hard and very difficult to solve in practice. The same holds for the 2L-CVRP. To our knowledge, the only available solution methodology for the 2L-CVRP is the exact approach developed by Iori, Salazar González and Vigo [22, 19, 20]: it consists of a branch-and-cut algorithm that handles the routing aspects of the problem, combined with a nested branch-and-bound procedure to check the feasibility of the loadings. The algorithm can solve instances involving up to 30 clients and 90 items. Since these limits are not reasonable for real-life problems, it is natural to consider heuristic and metaheuristic techniques. In this paper we propose a Tabu Search algorithm for the 2L-CVRP.

Several metaheuristic approaches are available for the CVRP (see Cordeau et al. [5] for a recent survey): Tabu Search, Genetic Algorithms, Simulated and Deterministic Annealing, Ant Colonies and Neural Networks. Very good results were obtained through Tabu Search (Cordeau and Laporte [7]). Metaheuristic techniques have also been widely applied to packing problems. The 2BPP was solved through Simulated Annealing by Dowsland [10] and through Tabu Search by Lodi, Martello and Vigo [23]. Tabu Search and Genetic Algorithms for the 2SP were developed by Iori, Martello and Monaci [21].

The success of Tabu Search both for routing and packing problems influenced our choice in favor of such technique for the 2L-CVRP. An additional consideration is that even the problem of finding a feasible solution to the 2L-CVRP (i.e., a partition of the clients into subsets that satisfy the weight capacity of the vehicles and for which a feasible two-dimensional loading exists) is already strongly NP-hard and very difficult to solve in practice. Techniques based on populations of solutions, such as Genetic Algorithms and Scatter Search, would be forced to deal with a high number of infeasible solutions, and hence do not seem very promising.

In Section 2 we introduce our notation and give a detailed description of the addressed problems. In Section 3 we present the proposed Tabu Search approach. The intensification phase, which constitutes a very important component of the algorithm, is described in detail in Section 4. Computational results on small-size and large-size instances are given in Section 5.

2 Problem Description

We are given a complete undirected graph $G = (V, E)$, where V is a set of $n + 1$ vertices corresponding to the depot (vertex 0) and to the clients (vertices $1, \ldots, n$), and E is the set of edges (i, j), each having an associated cost c_{ij}. There are v identical vehicles, each having a weight capacity D and a rectangular loading surface of width W and height H. Let $A = W \times H$ denote the loading area. The demand of client i ($i = 1, \ldots, n$) consists of m_i items of total weight d_i: item $I_{i\ell}$ ($\ell = 1, \ldots, m_i$) has width $w_{i\ell}$ and height $h_{i\ell}$. Let $a_i = \sum_{\ell=1}^{m_i} w_{i\ell} h_{i\ell}$ denote the total area of the lot demanded by client i.

We assume that the items have a fixed orientation, i.e., they must be packed with the w-edge (resp. the h-edge) parallel to the W-edge (resp. the H-edge) of the loading surface. This restriction is quite usual in logistics when pallet loading is considered, since

there can be a fixed orientation for the entering of the forks that prohibits rotations during the loading and unloading operations. We also assume, as is usual for the VRP, that each client must be served by a single vehicle. When a vehicle k is assigned a tour that includes a client set $S(k) \subseteq \{1, \ldots, n\}$, the two following constraints must be satisfied:

Weight constraint: the total weight $\sum_{i \in S(k)} d_i$ must not exceed the vehicle capacity D;

Loading constraint: there must be a feasible (non-overlapping) loading of the items in

$$I(k) = \bigcup_{i \in S(k)} \bigcup_{\ell \in \{1,\ldots,m_i\}} I_{i\ell} \tag{1}$$

into the $W \times H$ loading area.

The objective is to find a partition of the clients into at most v subsets and, for each subset $S(k)$, a route starting and ending at the depot (vertex 0) such that both conditions above hold, and the total cost of the edges in the routes is a minimum. An example is depicted in Figure 1.

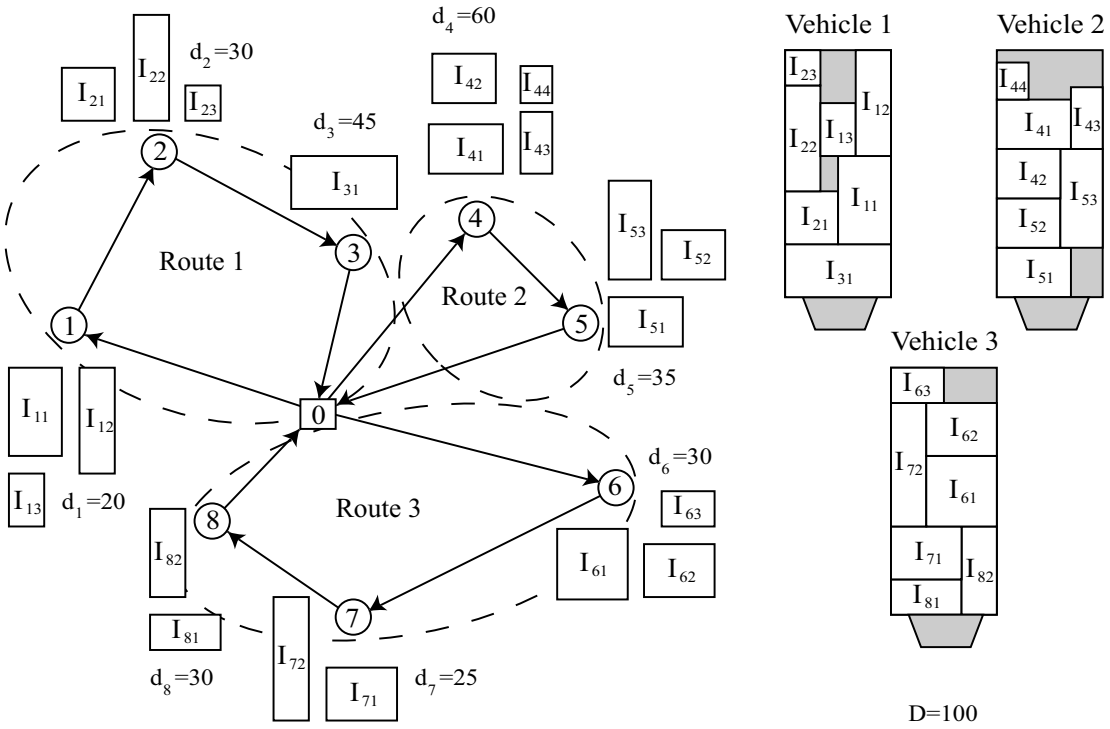

Figure 1: *Example of the 2L-CVRP.*

We will consider two variants of the problem: the *Unrestricted* 2L-CVRP and the *Sequential* 2L-CVRP. In the former variant no additional constraint is imposed, while in the latter we also impose to each vehicle, say k, the following constraint:

Sequence constraint: the vehicle loading must be such that when client $i \in S(k)$ is visited, the items of the corresponding lot, $\bigcup_{\ell \in \{1,\ldots,m_i\}} I_{i\ell}$, can be downloaded through a sequence of straight movements (one per item) parallel to the H-edge of the loading area.

In other words, for any item $I_{i\ell}$, no item demanded by a client visited later on in the tour must lay on the strip going from the w-edge of $I_{i\ell}$ to the rear of the vehicle. The dashed strip above item I_{il} in Figure 2 (a) shows the forbidden loading area.

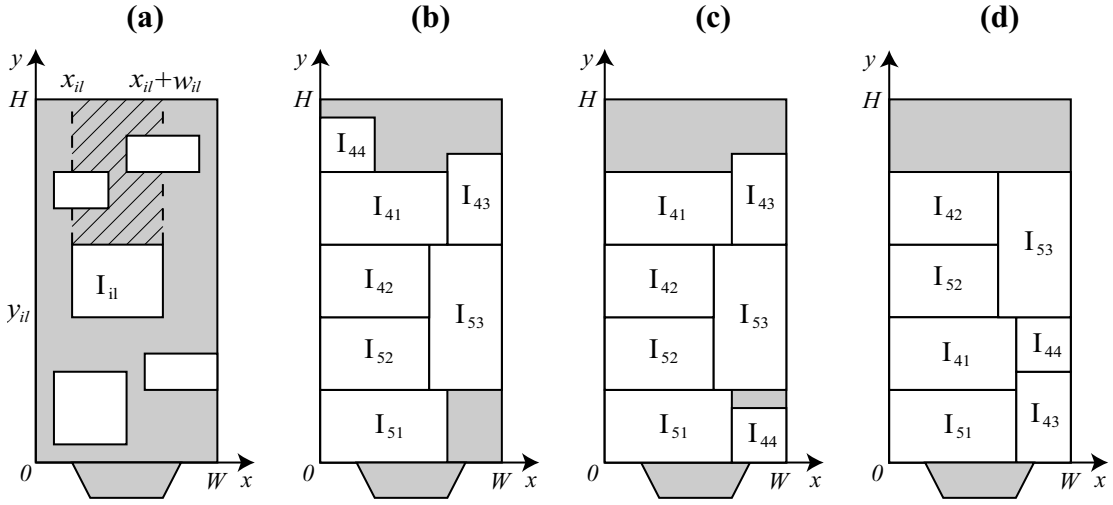

Figure 2: *The loading surface.*
(a) *forbidden area*; (b) *sequential loading*; (c), (d) *unrestricted loadings.*

This constraint can arise in practice when the items have great weight, size or fragility, so that moving them inside the vehicle is extremely difficult or impossible. The sequential variant is the one studied in [22, 19, 20]. In Figure 1, we show a simple example with 3 vehicles of weight capacity $D = 100$, 8 customers and 21 items. The three patterns given for the vehicles are feasible sequential loadings for the freight delivery. (An example of non-sequential loading is given in Figure 2.)

We assume in the following that all input data are positive integers. The loading area of a vehicle is represented in the positive quadrant of a Cartesian coordinate system, with its bottom left corner in $(0, 0)$ and the W-edge (resp. H-edge) parallel to the x-axis (resp. y-axis), and the vehicle rear coinciding with segment $[(0, H), (W, H)]$. The position of an item $I_{i\ell}$ in a loading pattern is given by the coordinates of its bottom left point, $(x_{i\ell},y_{i\ell})$. The coordinate system is shown in Figure 2: by assuming that the height of item I_{51} is $h_{51} = 40$, in Figure 2 (b) the position of item I_{51} is $(0, 0)$, that of item I_{52} is $(0, 40)$, and so on.

3 A Tabu Search Heuristic

Tabu Search is a very effective and simple technique for the solution of optimization problems. It was first proposed by Glover [15, 16] and has since then been applied to a large number of different problems (see, e.g., Glover [17] and Glover and Laguna [18] for surveys).

Tabu Search starts from an initial (usually feasible) solution and iteratively moves to a new one selected in a certain *neighborhood* of the current solution. At each iteration, the *move* yielding the best solution in the neighborhood is selected, even if this results in a worse solution. To avoid cycling, solutions possessing some attributes of previously visited solutions are declared tabu for a number of iterations. In order to obtain good practical results, this simple scheme is usually enriched with additional features. An *intensification* phase is often performed to accentuate the search in a promising area, while *diversification* is necessary to escape from a poor or already extensively searched area.

The Tabu Search approach introduced in the next sections can accept moves producing infeasible tours in the following sense. For the sequential 2L-CVRP, all tours considered in the search satisfy the sequence constraint. However, both in the sequential and the unrestricted 2L-CVRP, the tours can be *weight-infeasible* if the total weight exceeds D, or *load-infeasible* if the packing needs a loading surface of height exceeding H. The algorithm never produces packings requiring a loading surface wider than W or implying overlapping items. Infeasible moves are assigned a *penalty* proportional to the level of the violation.

Thus, when performing a move, the algorithm must consider the improvement of the current solution in terms of total edge cost, and check the feasibility of the candidate tour. For the routing aspects, we have adopted an approach (described later in Section 3.3) derived from some successful features of Taburoute, a Tabu Search heuristic developed by Gendreau, Hertz and Laporte [13] for the solution of vehicle routing problems. For what concerns feasibility, the weight constraint is immediate to check, but the loading constraint requires the solution of an NP-hard problem. We have thus developed a heuristic algorithm, described in the next section, executed for each set of clients assigned to a vehicle: it outputs a two-dimensional pattern of width W and height $Hstrip$, to be tested against the available height H.

3.1 A heuristic algorithm to check the loading constraints

Let $I(k)$ (see (1)) be the set of items currently assigned to vehicle k. In this subsection we describe two heuristics, $\mathrm{LH_{2SL}}$ and $\mathrm{LH_{2UL}}$, to check the loading constraints for vehicle k in the sequential and the unrestricted case, respectively. The algorithms iteratively apply an inner procedure based on the heuristic rule used in algorithm $\mathrm{TP_{RF}}$ (*Touching Perimeter*) proposed by Lodi, Martello and Vigo [23] for the two-dimensional bin packing problem, and applied by Iori, Martello and Monaci [21] to the two-dimensional strip packing problem. We have developed two inner procedures, $\mathrm{TP_{2SL}}$ and $\mathrm{TP_{2UL}}$, for the sequential and the unrestricted case, respectively. Both procedures assign one item at a time (according to a given item sorting) to a strip of width W, trying to maximize at each iteration the percentage of the item perimeter touching the strip or other items already packed (*touching*

perimeter). Each item is packed in the *normal position* (as defined by Christofides and Whitlock [3]), i.e., with its bottom edge touching either the bottom of the strip or the top edge of another item, and with its left edge touching either the left edge of the strip or the right edge of another item.

We next provide the description of procedure $\mathrm{TP}_{\mathrm{2SL}}$ (sequential case), shown in Figure 3. The first call to $\mathrm{TP}_{\mathrm{2SL}}$ is performed by $\mathrm{LH}_{\mathrm{2SL}}$ with an item sorting obtained by scanning the clients assigned to a route in reverse order of visit and, for each client, by listing the items in the corresponding lot according to non-increasing width, breaking ties by non-increasing height. Subsequent calls are performed by perturbing this order, as will be shown. Procedure $\mathrm{TP}_{\mathrm{2SL}}$ can terminate in one of three possible ways: with a feasible packing, with a load-infeasible packing on a surface $W \times Hstrip$ with $Hstrip > H$ (to be penalized within Tabu Search), or with no packing at all. Since the last outcome cannot occur at the first call, procedure $\mathrm{LH}_{\mathrm{2SL}}$ always returns a feasible or load-infeasible packing.

procedure $\mathbf{TP}_{\mathrm{2SL}}(I(k))$
1. pack the first item of $I(k)$ in position $(0,0)$ and set $\overline{H}$ at its height;
2. **for each** successive item, say I of size (w,h), of $I(k)$ **do**
   - 2.1 P:= {set of normal packing positions (x,y) : (a), (b), (c) below hold}
     - (a) $x + w \leq W$;
     - (b) the rectangle of sides $([(x,y),(x+w,y)], [(x,y),(x,y+h)])$ is empty;
     - (c) for each already packed item $\overline{I}$, say of size $(\overline{w},\overline{h})$ packed at $(\overline{x},\overline{y})$,
       - (c.1) if $\overline{I}$ belongs to the lot of a previous client then either $(\overline{y} \geq y)$ or $(\overline{x} + \overline{w} \leq x)$ or $(\overline{x} \geq x + w)$;
       - (c.2) if $\overline{I}$ belongs to the lot of a successive client then either $(\overline{y} \leq y + h)$ or $(\overline{x} + \overline{w} \leq x)$ or $(\overline{x} \geq x + w)$;
   - 2.2 **if** $P = \emptyset$ **then return** ∞ **else** $\overline{P} = \{(x,y) \in P : y + h \leq H\}$;
   - 2.3 **if** $\overline{P} \neq \emptyset$ **then** pack I in the position $(x,y) \in \overline{P}$ producing maximum touching perimeter **else** pack I in the position $(x,y) \in P$ for which $y + h$ is a minimum;
   - 2.4 $\overline{H} := \max\{\overline{H}, y + h\}$

   end for;
3. **return** $\overline{H}$.

Figure 3: *Inner heuristic for producing sequential loadings.*

The procedure works as follows. The first item is packed in position $(0,0)$. The next item, say $I_{i\ell}$, is packed, if possible, with its bottom-left corner at a coordinate (x,y) that maximizes the touching perimeter, provided that (i) the unloading sequence is feasible (see Figure 2 (a)), (ii) the induced strip height does not exceed the vehicle limit. If no such coordinate exists, the item is packed in that position, among those satisfying (i) (if any), for which the induced strip height is smaller. With item sortings other than the initial

one it is possible that an item is encountered for which no packing satisfying (i) exists, in which case a dummy infinite height is returned. Figure 2 (b) shows the packing produced by $\mathrm{TP}_{\mathrm{2SL}}$ for the second vehicle of Figure 1 with the initial sorting. Note that item I_{42}, which is packed after item I_{41}, can be placed below it, since this position is feasible.

Let $\overline{m}$ be the total number of items required by the clients of the considered route. For each item, the number of candidate normal packing positions (set P) is $O(\overline{m})$. The feasibility check of Step 2.1 (c) and the touching perimeter computation of step 2.3 require $O(\overline{m})$ time. The overall time complexity of $\mathrm{TP}_{\mathrm{2SL}}$ is thus $O(\overline{m}^3)$.

Procedure $\mathrm{TP}_{\mathrm{2SL}}$ is iteratively invoked by algorithm $\mathrm{LH}_{\mathrm{2SL}}$ on modified item sequences as shown in Figure 4, where λ is the maximum number of times the inner procedure is executed. On input, the items are ordered according to the initial sorting previously described. The output is either a feasible packing if the returned value is $Hstrip \leq H$, or a load-infeasible packing if $Hstrip > H$. Since Procedure $\mathrm{TP}_{\mathrm{2SL}}$ finds one of these two kinds of packing when invoked with the initial sorting, $\mathrm{LH}_{\mathrm{2SL}}$ will always return a packing.

Procedure $\mathbf{LH}_{\mathrm{2SL}}(I(k),\lambda)$
1. $Hstrip := \mathrm{TP}_{\mathrm{2SL}}(I(k))$, $it := 1$;
2. **while** $Hstrip > H$ **and** $it \leq \lambda$ **do**
   2.1 randomly select two items and switch their positions;
   2.2 $\xi := \mathrm{TP}_{\mathrm{2SL}}(I(k))$;
   2.3 $Hstrip := \min\{Hstrip, \xi\}$;
   2.4 $it := it + 1$;
   end while
3. **return** $Hstrip$.

Figure 4: *Heuristic algorithm for checking the sequential loading constraints.*

For the unrestricted case, the corresponding procedure, $\mathrm{LH}_{\mathrm{2UL}}$, is identical to $\mathrm{LH}_{\mathrm{2SL}}$ except for the call to a procedure $\mathrm{TP}_{\mathrm{2UL}}$ which is obtained from $\mathrm{TP}_{\mathrm{2SL}}$ by dropping condition (c) of Step 2.1. (Note that in this case P will never be empty at Step 2.2.) For the vehicle of Figure 2 (b), Figures 2 (c) and (d) show two packings produced by $\mathrm{TP}_{\mathrm{2UL}}$: the former with the initial sorting, i.e., the one used for the sequential case, the latter with items sorted by decreasing width, breaking ties by decreasing height. The initial sorting is generally preferable when one wants to reduce the number of items to be moved in the unloading phase, while the other sorting can produce a better filling of the loading areas. The former sorting was adopted for the computational experiments of Section 5.

3.2 Initial solution

We use two heuristic algorithms (each in two versions, for the sequential and the unrestricted cases) for determining an initial solution. The first algorithm, $\mathrm{IHG}_{\mathrm{2SL}}$ or $\mathrm{IHG}_{\mathrm{2UL}}$,

works for general 2L-CVRP instances and is always executed. For the special case of Euclidean instances, a second heuristic algorithm, IHE_{2SL} or IHE_{2UL}, is also executed.

Algorithm IHG_{2SL} (resp. IHG_{2UL}) was obtained by embedding the loading and sequencing constraints (resp. the loading constraints) in the classical Clarke and Wright [4] savings algorithm. Whenever an attempt is made to merge two routes, we check whether the weight constraint is satisfied, and, if so, we execute procedure LH_{2SL} (resp. LH_{2UL}) of Section 3.1 to check the load feasibility. The algorithm is first executed by only accepting merges for which a feasible packing is returned. If in this way we obtain a solution that uses v vehicles, the execution terminates. Otherwise, starting from the current solution, we execute a new round where load-infeasible packings are accepted.

The Euclidean algorithm, IHE_{2SL} (resp. IHE_{2UL}), was obtained by embedding the loading and sequencing constraints (resp. the loading constraints) in the algorithm developed by Cordeau, Gendreau and Laporte [6] for periodic and multi-depot VRPs. We start by randomly selecting a radius emanating from the depot, and by sorting the clients by increasing value of the angle between the radius and the segment connecting the client to the depot. The first client is assigned to the first vehicle. We then attempt to assign each subsequent client to the current vehicle by checking weight capacity and loading feasiblity through LH_{2SL} (resp. LH_{2UL}). For the first $v-1$ vehicles, only feasible packings are accepted, while a load-infeasible packing is only accepted for the last vehicle. The best feasible solution determined in the initialization phase (if any) is stored as the incumbent solution.

3.3 Tabu Search

As previously observed, our Tabu Search heuristic has to deal with weight-infeasible or load-infeasible solutions. We treat infeasibilities as penalties in an objective function to be minimized. Let s denote a solution, consisting of a set of $\widetilde{v}$ routes, with $1 \leq \widetilde{v} \leq v$, in which each client belongs to exactly one route. Let $c(k)$ be the total cost of the edges in route k. As in Taburoute, we express the objective function as a sum of three terms:

$$z'(s) = z(s) + \alpha q(s) + \beta h(s), \tag{2}$$

where $q(s)$ and $h(s)$ (see below) represent the entity of the violation, if any, of the weight and loading constraints, respectively, while α and β are self-adjusting parameters. Using the notation introduced in Section 2 and denoting by H_k the height of the two-dimensional loading of vehicle k, we have

$$z(s) = \sum_{k=1}^{\widetilde{v}} c(k), \tag{3}$$

$$q(s) = \sum_{k=1}^{\widetilde{v}} \left[\sum_{i \in S(k)} d_i - D \right]^+, \tag{4}$$

$$h(s) \;=\; \sum_{k=1}^{\widetilde{v}} [H_k - H]^+ , \tag{5}$$

where $[a]^+ = \max\{0, a\}$.

A move consists of selecting a client i, currently inserted in a route k, removing it from the route and inserting it into a new route k'. The cost of the new solution is obtained through the GENI (Generalized Insertion Procedure) heuristic developed by Gendreau, Hertz and Laporte [14] for the Travelling Salesman Problem. GENI considers a subset of the possible 4-opt modifications of the two tours (the one from which the client is deleted and the one where it is inserted), selecting the best one for each tour. Let $N_p(i)$ be the set of the p vertices closest to i (*neighbors* of client i). In order to reduce the number of possible moves, only empty routes and routes containing at least one neighbor of i are considered for possible insertion of i. A good compromise between computing time and solution quality was experimentally obtained with $p = 10$.

At each iteration, all possible moves are evaluated, i.e., for each client we consider its possible insertion into all routes satisfying the neighbor condition above: for each move, the resulting cost $z'(s)$ (see (2)–(5)) is obtained by determining $z(s)$ through GENI, directly computing $q(s)$, and computing $h(s)$ through procedure $\mathrm{LH_{2SL}}$ (or $\mathrm{LH_{2UL}}$). In certain situations, a *penalty term* π (see below, equation (6)) is possibly added to the cost in order to diversify the search. The move producing the solution with minimum overall cost is selected. After a move has been performed by removing client i from route k, reinserting i in k is declared tabu for the next ϑ iterations. An aspiration criterion is based on an $n \times v$ array M that stores in M_{ij} the value of the best feasible solution so far obtained, if any, with client i assigned to route j ($M_{ij} = \infty$ if no such solution has been identified). A tabu move that reinserts client i in route k can be accepted if it improves the current M_{ij} value. The value of ϑ was experimentally determined as $\vartheta = \log(nv)$.

Three main parameters govern the search process: α controls weight constraint violations (see (2)), while β and λ control packing constraint violations (see (2) and Section 3.1). In the initialization phase α and β are set to 1, and λ to 2. In the iterative phase the three parameters are recursively updated following the characteristics of the solutions produced by the moves. If the current solution is (resp. is not) weight-infeasible we set $\alpha = \alpha(1 + \delta)$ (resp. $\alpha = \alpha/(1 + \delta)$). In addition, if the current solution is (resp. is not) load-infeasible we set $\beta = \beta(1 + \delta)$ and $\lambda = \min\{\lambda + 1, 4\}$ (resp. $\beta = \beta/(1 + \delta)$ and $\lambda = \max\{\lambda - 1, 1\}$. A good value of δ was experimentally determined as $\delta = 0.01$.

We use a simple diversification technique to escape from poor local minima. Let s_a be the current solution, and s_b the solution produced by a move that removes customer i from route k. As in Taburoute, if $z'(s_b) \geq z'(s_a)$, we add to $z'(s_b)$ a penalty term π, proportional to the frequency of the considered move during the previous iterations. Let ρ_{ik} denote the number of times customer i was removed from route k, divided by the total number of iterations. The penalty term is then

$$\pi = \rho_{ik}(z'(s_b) - z'(s_a))\gamma, \tag{6}$$

with γ set to $\sqrt{nv}$, as suggested by Taillard [28] and Cordeau, Laporte and Mercier [8]. In

this way, when no downhill move is found, the algorithm is more likely to escape from poor local search areas. In order to deeply explore promising areas of the solution space, we extensively use intensification algorithms. Because of their importance they are described in the next section.

4 Intensification

Two kinds of intensification techniques are adopted, one executed periodically, every ω moves, and one (computationally heavier) executed whenever a weigth-feasible potential new incumbent is obtained. Both intensification processes make use of a procedure ($\text{Repack}_{2\text{SL}}(k)$ for the sequential case, or $\text{Repack}_{2\text{UL}}(k)$ for the unrestricted case) that looks for a feasible packing for the items of each route k of a load-infeasible solution s, i.e., one for which $H_k > H$.

Recall that $I(k)$ is the set of items associated with the clients of route k. Procedure $\text{Repack}_{2\text{SL}}(I(k), \lambda)$, shown in Figure 5, starts by computing lower bounds (Martello and Vigo [26]) for the two-dimensional bin packing problem instance induced by the items associated with route k and bin size $W \times H$. If more than one bin is needed, no feasible loading exists and the procedure terminates. Otherwise we execute the heuristic search $\text{LH}_{2\text{SL}}$ of Section 3.1. For this computation, it turned out to be convenient to allow a much higher number of calls, $\lambda = 10$, to the inner heuristic $\text{TP}_{2\text{SL}}$. If no feasible packing is still found, we execute the branch-and-bound algorithm of Iori, Salazar González and Vigo [22, 19, 20] with a limit of 1 000 backtrackings.

Procedure $\text{Repack}_{2\text{SL}}(I(k), \lambda)$
1. $L :=$ lower bound for the 2BPP instance defined by $I(k)$ with bin size $W \times H$;
2. **if** $L > 1$ **then return** ∞;
3. $\xi := \text{LH}_{2\text{SL}}(I(k), \lambda)$;
4. **if** $\xi \leq H$ **then return** ξ;
5. execute a truncated branch-and-bound search;
6. **if** a feasible loading is found **then return** H
   else return ∞.

Figure 5: *Intensification for packing constraints.*

Procedure $\text{Repack}_{2\text{UL}}$ is identical to $\text{Repack}_{2\text{SL}}$ except for the call to $\text{LH}_{2\text{UL}}$ instead of $\text{LH}_{2\text{SL}}$, and for the execution of a truncated branch-and-bound search that does not consider the sequence constraints.

4.1 Periodic Intensification

Every ω moves, a local search for improving the incumbent solution is performed through a modified version of the heuristic algorithm of Iori, Salazar González and Vigo [22, 19, 20].

This algorithm is embedded within a branch-and-cut approach and operates on a modified cost matrix $\overline{c}$ obtained by setting, for each edge (i, j), $\overline{c}_{ij} = c_{ij}(1 - x_{ij}/2)\mu$, where x_{ij} is the (possibly fractional) value of the 0-1 decision variable associated with the edge and μ is a randomization factor uniformly distributed in [0,1]. The new solution is then obtained by iteratively building routes through a parametric version of the savings algorithm. Each route (client set) S is constructed by first selecting the client i for which the quantity $\sigma_c(c_{0i}+ c_{i0}) + \sigma_d d_i + \sigma_a a_i$ is a maximum (σ_c, σ_d and σ_a prefixed parameters), and then iteratively inserting in S the client j for which the residual weight capacity $d_{res} = D - \sum_{i \in S} d_i - d_j$ and loading area $a_{res} = A - \sum_{i \in S} a_i - a_j$ are non-negative, and $\sigma_c($ insertion cost $) + \sigma_d d_{res} + \sigma_a a_{res}$ is a minimum.

In our case, a convenient way to define the modified cost matrix turned out to be

$$\overline{c}_{ij} = \begin{cases} rc_{ij} \text{ (with } r \text{ uniformly random in [0,2])} & \text{if } (i,j) \text{ belongs to a route,} \\ c_{ij} & \text{otherwise.} \end{cases} \tag{7}$$

and only clients for which the residual loading area a_{res} is no less than a threshold value τ are considered for possible insertion, in order to take into account the difficulty of packing the corresponding items. Indeed any loading usually implies a space waste, so it is improbable that a feasible loading can be found if a_{res} is close to zero. The value τ is randomly generated in $[0, A \cdot R/4]$, where

$$R = \max\{\max_{i\ell}\{h_{i\ell}/H\}, \max_{i\ell}\{w_{i\ell}/W\}\} \tag{8}$$

(maximum relative size of an item) measures the "difficulty" of the packing problem associated with the instance. In other words, we decrease the residual packing area by a percentage that takes into account the probability that portions of the loading surface remain unused.

We use a set of five triplets for the values of $(\sigma_c,\ \sigma_d,\ \sigma_a)$: $T = \{(1,0,0),\ (\frac{1}{2},\ \frac{1}{2},0),\ (\frac{1}{2},0,\ \frac{1}{2}),\ (0,\frac{1}{2},\ \frac{1}{2}),\ (\frac{1}{3},\frac{1}{3},\ \frac{1}{3})\}$. The savings algorithm is executed for each triplet, and all solutions that use no more than v routes are improved through 2-opt modifications. In addition, all solutions of value less than that of the incumbent are stored in a *pool* Π. We then consider the solutions in Π in increasing value. For each solution, we try to obtain a feasible packing for each route through procedure $\text{Repack}_{2\text{SL}}(I(k), \lambda)$, stopping as soon as an overall feasible solution (if any) is obtained. Given $\overline{c}$ and a triplet σ, let $s =$ SAVINGS($\overline{c}$, σ) be the solution built by the savings algorithm, and TWO-OPT(c,s) the improved solution produced by the first improvement (if any) found through the two-opt local search algorithm (TWO-OPT$(c, s) \equiv s$ if no improvement was found). All solution values $z(s)$ are obviously evaluated using the original cost matrix c. The overall procedure, $\text{Improve}_{2\text{SL}}$, is shown in Figure 6, where z denotes the value of the incumbent solution.

Procedure $\text{Improve}_{2\text{UL}}$ is identical to $\text{Improve}_{2\text{SL}}$ except for the call to $\text{Repack}_{2\text{UL}}(I(k), \lambda)$ instead of $\text{Repack}_{2\text{SL}}(I(k), \lambda)$. Basing on the outcome of computational testings, we set ω to 1 for $n \leq 40$ and to 5 for larger values.

Procedure Improve$_{\text{2SL}}$
1. create the modified cost matrix $\overline{c}$ and set $\Pi := \emptyset$;
2. **for each** triplet $\sigma = (\sigma_c, \sigma_d, \sigma_a) \in T$ **do**
    $s :=$ SAVINGS($\overline{c}$, σ);
    if s includes no more than v routes **then**
      if $z(s) < z$ **then** $\Pi := \Pi \cup \{s\}$;
      $z_{\text{old}} := \infty$;
      while $z(s) < z_{\text{old}}$ **do**
        $z_{\text{old}} := z(s)$;
        $s :=$ TWO-OPT(c, s);
        if $z(s) < z$ **then** $\Pi := \Pi \cup \{s\}$
      end while
    end if
  end for.
3. sort the solutions $s \in \Pi$ by non-decreasing $z(s)$ value;
4. **for each** $s \in \Pi$ in order **do**
    for each route k of s **do**
      $\xi :=$ Repack$_{\text{2SL}}(I(k), \lambda)$;
      if $\xi > H$ **then break**
    end for;
    if $\xi \leq H$ **then return** s
  end for;
  return $\emptyset$.

Figure 6: *Algorithm for improving the incumbent solution.*

4.2 Special intensification

Whenever a solution s having a cost $z(s)$ lower than that of the incumbent is produced, different actions are taken, depending on the feasibility of the weight and of the loading associated with s:

(i) if s is weight-infeasible, no attempt is made to make it feasible;

(ii) if s is totally feasible, the incumbent is updated, and some more time is spent in an attempt to further improve s on an enlarged neighborhood. This is achieved by doubling, for each client i, the size of $N_p(i)$, and applying GENI for each resulting possible move. In our implementation this results in considering all routes that are empty or contain at least one of the 20 vertices closest to i;

(iii) if s is weight-feasible but load-infeasible, we first iteratively execute procedure Repack for each route k of s for which $H_k > H$. If a feasible solution is obtained in this way, the additional intensification of case (ii) above is further performed.

5 Computational Results

The Tabu Search heuristic developed in the previous sections was coded in C and tested on a test set obtained by modifying instances from the literature. All experiments were executed on a Pentium IV 1700 MHz. The graph and the weights demanded by the customers were obtained from CVRP instances (test instances for the CVRP are described in [29], and can be downloaded from `http://www.or.deis.unibo.it/research.html`). The number of items and their dimensions were created according to the five classes introduced in [22] and [19, 20], namely:

Class 1: each customer i, for $i = 1, \ldots, n$ is assigned one item with unit width and height.

Classes 2 – 5: a uniform distribution on a certain interval (see Table 1, column 2) is used to generate the number m_i of items for each customer i. Each item is then randomly assigned, with equal probability, one out of three possible shapes: *vertical* (the relative heights are greater than the relative widths), *homogeneous* (the relative heights and widths are generated in the same intervals), or *horizontal* (the relative heights are smaller than the relative widths). Finally, the dimensions of the items are uniformly generated in a given interval (see again Table 1, columns 3–8).

The values $H = 40$ and $W = 20$ were chosen for the dimensions of the loading area. Class 1 corresponds to pure CVRP instances, for which no loading constraint is imposed. The other classes proved to be a challenging mix of items of different sizes.

Table 1: Classes 2 –5

		Vertical		*Homogeneous*		*Horizontal*	
Class	m_i	$h_{i\ell}$	$w_{i\ell}$	$h_{i\ell}$	$w_{i\ell}$	$h_{i\ell}$	$w_{i\ell}$
2	$[1, 2]$	$\left[\frac{4H}{10}, \frac{9H}{10}\right]$	$\left[\frac{W}{10}, \frac{2W}{10}\right]$	$\left[\frac{2H}{10}, \frac{5H}{10}\right]$	$\left[\frac{2W}{10}, \frac{5W}{10}\right]$	$\left[\frac{H}{10}, \frac{2H}{10}\right]$	$\left[\frac{4W}{10}, \frac{9W}{10}\right]$
3	$[1, 3]$	$\left[\frac{3H}{10}, \frac{8H}{10}\right]$	$\left[\frac{W}{10}, \frac{2W}{10}\right]$	$\left[\frac{2H}{10}, \frac{4H}{10}\right]$	$\left[\frac{2W}{10}, \frac{4W}{10}\right]$	$\left[\frac{H}{10}, \frac{2H}{10}\right]$	$\left[\frac{3W}{10}, \frac{8W}{10}\right]$
4	$[1, 4]$	$\left[\frac{2H}{10}, \frac{7H}{10}\right]$	$\left[\frac{W}{10}, \frac{2W}{10}\right]$	$\left[\frac{H}{10}, \frac{4H}{10}\right]$	$\left[\frac{W}{10}, \frac{4W}{10}\right]$	$\left[\frac{H}{10}, \frac{2H}{10}\right]$	$\left[\frac{2W}{10}, \frac{7W}{10}\right]$
5	$[1, 5]$	$\left[\frac{H}{10}, \frac{6H}{10}\right]$	$\left[\frac{W}{10}, \frac{2W}{10}\right]$	$\left[\frac{H}{10}, \frac{3H}{10}\right]$	$\left[\frac{W}{10}, \frac{3W}{10}\right]$	$\left[\frac{H}{10}, \frac{2H}{10}\right]$	$\left[\frac{W}{10}, \frac{6W}{10}\right]$

Some details on the instances generated in this way are reported in Table 2. For each CVRP instance, we created five 2L-CVRP instances (one per class) by generating the number of vehicles v through modified versions of the heuristic of Section 3.1. Recall that heuristic $\mathrm{LH}_{\mathrm{2SL}}$ returns the best solution obtained by iteratively invoking, on modified item sequences, procedure $\mathrm{TP}_{\mathrm{2SL}}$, which produces a packing of the item set into a strip of width W and minimum height. In the modified version, procedure $\mathrm{TP}_{\mathrm{2SL}}$ produces a packing of

Table 2: Details on the instance generation.

		Class 1			*Class 2*			*Class 3*			*Class 4*			*Class 5*		
Instance	n	M	v	LB	M	v	LB	M	v	LB	M	v	LB	M	v	LB
1) E016-03m	15	15	3	3	24	3	3	31	3	3	37	4	3	45	4	3
2) E016-05m	15	15	5	5	25	5	5	31	5	5	40	5	5	48	5	5
3) E021-04m	20	20	4	4	29	5	4	46	5	4	44	5	4	49	5	4
4) E021-06m	20	20	6	6	32	6	6	43	6	6	50	6	6	62	6	6
5) E022-04g	21	21	4	4	31	4	4	37	4	4	41	4	4	57	5	4
6) E022-06m	21	21	6	6	33	6	6	40	6	6	57	6	6	56	6	6
7) E023-03g	22	22	3	3	32	5	4	41	5	4	51	5	4	55	6	3
8) E023-05s	22	22	5	5	29	5	5	42	5	5	48	5	5	52	6	5
9) E026-08m	25	25	8	8	40	8	8	61	8	8	63	8	8	91	8	8
10) E030-03g	29	29	3	3	43	6	5	49	6	4	72	7	6	86	7	5
11) E030-04s	29	29	4	4	43	6	5	62	7	6	74	7	6	91	7	5
12) E031-09h	30	30	9	9	50	9	9	56	9	9	82	9	9	101	9	9
13) E033-03n	32	32	3	3	44	7	5	56	7	5	78	7	6	102	8	5
14) E033-04g	32	32	4	4	47	7	5	57	7	5	65	7	5	87	8	4
15) E033-05s	32	32	5	5	48	6	5	59	6	6	84	8	7	114	8	6
16) E036-11h	35	35	11	11	56	11	11	74	11	11	93	11	11	114	11	11
17) E041-14h	40	40	14	14	60	14	14	73	14	14	96	14	14	127	14	14
18) E045-04f	44	44	4	4	66	9	7	87	10	8	112	10	8	122	10	6
19) E051-05e	50	50	5	5	82	11	9	103	11	10	134	12	10	157	12	8
20) E072-04f	71	71	4	4	104	14	12	151	15	13	178	16	13	226	16	13
21) E076-07s	75	75	7	7	114	14	12	164	17	14	168	17	14	202	17	14
22) E076-08s	75	75	8	8	112	15	13	154	16	14	198	17	14	236	17	14
23) E076-10e	75	75	10	10	112	14	13	155	16	14	179	16	14	225	16	14
24) E076-14s	75	75	14	14	124	17	14	152	17	14	195	17	14	215	17	14
25) E101-08e	100	100	8	8	157	21	18	212	21	18	254	22	19	311	22	19
26) E101-10c	100	100	10	10	147	19	16	198	20	17	247	20	18	310	20	18
27) E101-14s	100	100	14	14	152	19	17	211	22	19	245	22	19	320	22	19
28) E121-07c	120	120	7	7	183	23	20	242	25	21	299	25	21	384	25	21
29) E135-07f	134	134	7	7	197	24	21	262	26	22	342	28	24	422	28	24
30) E151-12b	150	150	12	12	225	29	25	298	30	27	366	30	27	433	30	27
31) E200-16b	199	199	16	16	307	38	33	402	40	35	513	42	37	602	42	37
32) E200-17b	199	199	17	17	299	38	33	404	39	34	497	39	34	589	39	34
33) E200-17c	199	199	17	17	301	37	32	407	41	35	499	41	36	577	41	36
34) E241-22k	240	240	22	22	370	46	40	490	49	42	604	50	44	720	50	44
35) E253-27k	252	252	27	27	367	45	39	507	50	43	634	50	45	762	50	45
36) E256-14k	255	255	14	14	387	47	41	511	51	44	606	51	44	786	51	44

the item set into the minimum number v of finite bins (vehicles) of width W and height H, such that the weight and loading constraints are satisfied.

The CVRP instances 1–16 were already used in [22] and [19, 20], producing 80 different 2L-CVRP instances according to the five above classes, while the 100 instances obtained from 17–36 are considered here for the first time. In the lines of Table 2 (one for each group of five instances), the first column gives the name of the original CVRP instance and the second one the number of customers (n). The five following groups of three columns give, for each class, the total number of items ($M = \sum_{i=1}^{n} m_i$), the number of vehicles (v) and a lower bound (LB) on the minimum number of vehicles needed to ensure feasibility. This lower bound, reported here in order to show that the produced instances are indeed "reasonable", was computed as the maximum among two valid lower bounds: the number of vehicles in the original CVRP instance and the value obtained by solving the two-dimensional bin packing problem induced by the item sizes and the vehicle loading area. This second part of the bound was computed through the code by Martello, Pisinger and Vigo [25], available at `http://www.diku.dk/~pisinger`. (This code works for the three-dimensional bin packing problem and can obviously be used for the two-dimensional case as well.) The table shows that the v and LB values are very close: out of a total of 180 instances, these values are identical in 73 cases, and differ by at most two units in 57 cases. The complete set of 180 test instances that were obtained may be downloaded from `http://www.or.deis.unibo.it/research.html`.

In Tables 3 and 4, the performance of our approach is compared with the branch-and-cut algorithm by Iori, Salazar González and Vigo [22, 19, 20] on small-size instances involving less than 45 customers. Since the latter approach does not consider single-customer routes and always uses the v available vehicles, the Tabu Search algorithm was executed by making infeasible (through penalties) any solution not satisfying these properties. In addition, as in [22, 19, 20], the edge costs were computed by rounding down to the next integer the Euclidean distances between vertex pairs. The branch-and-cut algorithm had a time limit of 24 hours per instance, as in the experiments presented in [22] and [19, 20]. A convenient policy for controlling the Tabu Search was experimentally determined as follows. The algorithm is halted after $2n^2v$ iterations, or after one hour of CPU time.

For each instance, Tables 3 and 4 give the best solution obtained both by branch-and-cut (z_{bc}) and Tabu Search (z_{TS}), the elapsed CPU time in seconds when the best solution was found (sec_h), and the total CPU time in seconds (sec). Because the branch-and-cut is not always run to completion, the value of z_{bc} may be higher than that of z_{TS}. In addition, asterisks indicate proven optimal solutions, and $\%gap$ evaluates the quality of the Tabu Search solution as $\%gap = 100\ (z_{TS}\ \text{-}\ z_{bc})/z_{bc}$. The solutions found by the Tabu Search algorithm are very close to optimality, and are provably optimal in 33 out of 58 known optima. For half of the instances with 29 customers or more (Table 4), the Tabu Search solution is better than that found by the branch-and-cut in a much higher CPU time (negative $\%gap$ values). On the other hand, for about one third of the instances with 25 customers or less (Table 3) the Tabu Search CPU time is higher than that of branch-and-cut and, in six of these cases, no optimal solution is found. The worst percentage error is 7.69, the best percentage improvement is 21.23.

Table 3: Performance of the Tabu Search heuristic with respect to branch-and-cut. Sequential instances with $n \leq 25$, integer edge costs.

Instance		*Branch-and-cut*				*Tabu Search*			
no	*Class*	z_{bc}		sec_h	*sec*	z_{TS}	sec_h	*sec*	$\%gap$
1	1	273	*	3.8	3.9	273	0.0	2.4	0.00
	2	285	*	51.4	68.8	285	0.2	4.6	0.00
	3	280	*	17.1	21.4	280	1.3	7.8	0.00
	4	288	*	1.8	2.4	290	0.3	7.5	0.69
	5	279	*	53.1	53.1	279	2.3	15.7	0.00
2	1	329	*	0.2	0.5	329	0.1	1.4	0.00
	2	342	*	11.1	11.9	342	1.1	2.0	0.00
	3	347	*	6.4	8.1	350	0.2	3.2	0.86
	4	336	*	22.4	23.4	336	0.1	6.7	0.00
	5	329	*	29.2	29.4	329	0.3	6.1	0.00
3	1	351	*	14.9	15.6	351	0.2	8.4	0.00
	2	396	*	32.6	65.8	407	8.9	9.8	2.78
	3	387	*	6.0	6.1	387	1.0	20.0	0.00
	4	374	*	39.3	39.4	374	14.3	21.9	0.00
	5	369	*	0.2	0.2	369	1.0	29.9	0.00
4	1	423	*	0.4	0.4	423	0.2	5.7	0.00
	2	434	*	5.3	5.5	434	0.3	7.4	0.00
	3	432	*	5.2	8.2	438	5.0	12.8	1.39
	4	438	*	23.9	26.4	451	0.8	16.9	2.97
	5	423	*	23.0	44.7	423	1.2	27.1	0.00
5	1	367	*	0.1	0.1	367	2.9	12.8	0.00
	2	380	*	8.0	8.4	396	4.0	19.7	4.21
	3	373	*	1.8	1.9	377	0.6	20.5	1.07
	4	377	*	50.2	50.3	406	5.9	33.0	7.69
	5	389	*	2928.2	2928.3	389	5.7	57.8	0.00
6	1	488	*	5.9	10.7	488	0.1	10.3	0.00
	2	491	*	145.8	145.9	498	1.8	14.0	1.43
	3	496	*	135.4	150.4	496	11.2	16.9	0.00
	4	489	*	14.1	16.6	503	0.1	40.2	2.86
	5	488	*	10.8	13.3	488	0.7	34.2	0.00
7	1	558	*	0.0	0.0	558	0.3	22.0	0.00
	2	724	*	32.0	32.5	752	0.7	18.9	3.87
	3	698	*	3.2	4.4	704	19.9	29.8	0.86
	4	714	*	2596.8	2597.3	742	26.5	50.4	3.92
	5	742	*	738.9	747.2	743	16.1	75.1	0.13
8	1	657	*	0.0	0.0	657	7.0	31.3	0.00
	2	720	*	75.9	91.8	720	3.1	20.1	0.00
	3	730	*	70.0	73.0	752	20.5	33.4	3.01
	4	701	*	7.4	14.1	722	11.2	50.0	3.00
	5	721	*	1128.9	1128.9	736	12.0	90.2	2.08
9	1	609	*	6.2	31.91	609	1.9	11.9	0.00
	2	612	*	453.5	460.52	612	2.9	15.4	0.00
	3	615	*	164.8	194.31	626	7.9	38.4	1.79
	4	626	*	852.1	1593.34	627	5.9	43.5	0.16
	5	609	*	47.4	69.17	609	8.5	81.9	0.00

Table 4: Performance of the Tabu Search heuristic with respect to branch-and-cut. Sequential instances with $25 < n \leq 40$, integer edge costs.

Instance		*Branch-and-cut*				*Tabu Search*			
no	*Class*	z_{bc}		sec_h	sec	z_{TS}	sec_h	sec	$\%gap$
10	1	524	*	1226.2	83249.5	544	19.9	97.3	3.82
	2	774		48373.3	86401.1	703	3.8	72.2	−9.17
	3	637	*	2304.5	3150.0	676	47.9	118.7	6.12
	4	738	*	12671.3	12696.3	773	47.3	156.9	4.74
	5	706		48220.2	70308.0	724	84.5	308.9	2.55
11	1	500	*	0.1	0.1	500	0.8	107.8	0.00
	2	789		99.8	86400.5	734	4.7	72.2	−6.97
	3	763		63747.8	86400.3	785	72.2	101.7	2.88
	4	881		85181.5	86400.8	877	196.4	209.5	−0.45
	5	695		64392.9	86400.1	696	279.2	387.0	0.14
12	1	599		50093.5	86400.3	598	19.4	33.8	−0.17
	2	625		75.6	86400.7	628	9.6	42.9	0.48
	3	597		36171.0	86400.6	597	12.9	50.7	0.00
	4	624		86321.6	86400.2	640	96.0	120.6	2.56
	5	602		22352.4	86400.4	597	103.5	188.2	−0.83
13	1	1991	*	13.2	14.5	1991	47.4	218.7	0.00
	2	3523		10784.1	86400.3	2775	43.7	123.1	−21.23
	3	2574	*	3087.9	32124.5	2696	121.5	170.3	4.73
	4	2673		34999.2	86400.4	2743	29.3	277.0	2.62
	5	2807		83628.4	86400.1	2737	237.5	691.9	−2.49
14	1	827		25734.8	86400.4	823	21.8	145.0	−0.48
	2	1459		80521.1	86400.8	1266	52.3	144.4	−13.23
	3	1211		42204.7	86400.1	1204	137.9	207.1	−0.58
	4	1166	*	25391.4	25542.6	1187	108.5	324.9	1.80
	5	1504		80511.1	86400.1	1309	103.2	895.0	−12.97
15	1	907	*	17.8	17.8	907	51.9	196.4	0.00
	2	1203		386.9	86401.4	1135	94.7	133.9	−5.65
	3	1405		15880.7	86401.2	1183	37.1	205.9	−15.80
	4	1358		55614.5	86400.3	1372	268.2	332.2	1.03
	5	1390		59867.8	86400.1	1361	651.4	671.7	−2.09
16	1	682	*	7086.5	7086.7	682	56.1	96.6	0.00
	2	682	*	1767.0	3374.8	682	20.5	91.7	0.00
	3	682	*	345.1	3853.3	682	15.3	138.1	0.00
	4	691	*	33375.9	33391.8	704	21.7	206.1	1.88
	5	682	*	339.0	2784.0	682	62.8	276.6	0.00
17	1	859		493.1	86400.3	842	84.4	158.6	−1.98
	2	866		283.8	86400.4	851	56.6	132.5	−1.73
	3	850		656.0	86400.0	842	38.8	200.6	−0.94
	4	853		102.9	86400.4	845	7.4	279.7	−0.94
	5	845		263.6	86400.5	842	44.7	402.2	−0.36

Table 5: Synthesis of Tables 3 and 4: Sequential instances, integer edge costs.

	Branch-and-cut				*Tabu Search*			
Class	z_{bc}	sec_h	*sec*	%*opt*	z_{TS}	sec_h	*sec*	%*gap*
1	643.76	4982.2	20566.6	82.35	643.65	18.5	68.3	0.07
2	841.47	8418.1	35827.7	58.82	777.65	18.2	54.4	−2.66
3	769.29	9694.6	27741.0	70.59	769.12	32.4	80.9	0.31
4	783.94	19839.2	29882.1	70.59	799.53	49.4	128.1	2.03
5	798.82	21443.2	35088.7	58.82	783.12	95.0	249.4	−0.81
average	767.46	12875.4	29821.2	68.24	754.61	42.7	116.2	−0.21

In order to provide an easier way to evaluate the results presented in Tables 3 and 4, these are summarized in Table 5. The columns give the same information, but the entries provide the average values for each class, plus the overall average values in the last line. In addition, the asterisks are here replaced by column %*opt*, giving the average percentage of proven optimal solutions. The values in the last column, %*gap*, were computed as the average of the percentage gaps in Tables 3 and 4, hence it can happen (see, e.g., Class 1) that the average solution value is lower for Tabu Search but the percentage gap is positive, due to differences in the absolute solution values. The table shows that the quality of the solutions found by the two algorithms is practically equal for Classes 1 and 3, better for branch-and-cut for Class 4, and better for Tabu Search for Classes 2 and 5. On average the Tabu Search solutions improve those obtained through branch-and-cut by 0.21%. The CPU times required by the Tabu Search algorithm are very reasonable. On average it takes 116 seconds in total, and 43 seconds to find the best solution, while for the branch-and-cut algorithm these values increase to 29 821 and 12 875 seconds, respectively.

In Table 6 we present the results for the complete set of instances, both for sequential and unrestricted loading. The limit given to the Tabu Search heuristic is again the first occurrence between $2n^2v$ iterations and one hour of CPU time. The edge costs are computed as the rational Euclidean distances between any pair of vertices, without rounding them to integer values. One-customer routes are allowed, and fewer than v vehicles may be used. The first group of three columns refer to Class 1, i.e., to pure CVRP instances, while the next two groups give the average values for Classes 2–5, respectively for the unrestricted and the sequential case. In each group, the first entry gives the average solution value (z_1 and $\overline{z}_{2-5}$), the second one the average elapsed CPU time when the final solution was found (sec_h) and the third one the average total CPU time (*sec*). (Detailed results for each instance can be found in [19, 20]). The last line of the table reports the average values over all instances.

The Tabu Search algorithm could find a feasible solution for all 360 instances. By comparing columns z_1 and $\overline{z}_{2-5}$ (unrestricted) one can observe that the inclusion of the loading constraint considerably worsens the solution values of the CVRP (on average by

Table 6: Performance of the Tabu Search heuristic. Real edge costs.

	CVRP			*Unrestricted*			*Sequential*		
Inst.	z_1	sec_h	sec	$\overline{z}_{2-5}$	sec_h	sec	$\overline{z}_{2-5}$	sec_h	sec
1	278.73	2.0	2.2	291.60	4.2	9.7	299.09	2.6	9.2
2	334.96	0.0	1.4	341.02	0.1	3.7	345.23	0.4	3.5
3	359.77	3.5	8.4	377.35	1.6	19.6	385.30	3.8	18.9
4	430.88	0.1	5.7	437.45	0.5	14.7	443.42	1.4	17.0
5	375.28	1.4	12.8	380.20	5.0	25.2	384.06	4.1	27.6
6	495.85	0.3	8.7	501.02	7.2	18.8	502.78	5.1	19.5
7	568.56	0.5	22.6	700.34	6.3	48.4	721.90	15.5	53.0
8	568.56	0.5	36.2	694.99	11.2	61.3	722.73	32.8	83.7
9	607.65	0.4	13.5	619.69	3.6	41.2	624.06	10.6	40.0
10	538.79	6.1	81.7	700.39	36.0	192.0	714.90	43.5	179.6
11	505.01	2.5	98.9	739.04	55.7	207.6	773.45	99.0	199.4
12	610.57	28.5	32.5	620.62	49.0	82.7	631.85	58.8	99.5
13	2006.34	29.9	161.6	2598.20	57.5	332.0	2687.03	49.0	312.8
14	837.67	22.2	152.1	1047.72	375.8	565.6	1101.49	146.0	439.5
15	837.67	1.7	182.5	1201.38	156.7	375.9	1240.89	165.4	313.4
16	698.61	2.7	99.0	702.03	20.5	160.1	704.85	28.0	157.2
17	862.62	59.0	162.8	866.37	64.9	218.2	866.50	88.9	226.2
18	723.54	81.9	587.3	1085.84	589.3	1318.0	1116.17	566.5	1167.7
19	524.61	128.8	641.9	772.25	633.7	1524.5	802.48	365.2	1521.5
20	241.97	253.6	1005.2	564.67	954.5	3237.3	581.81	808.9	3370.3
21	688.18	325.0	2978.7	1066.21	460.1	3600.0	1110.19	1702.2	3561.2
22	740.66	2070.7	3600.1	1087.46	1191.2	3544.5	1130.33	1573.8	3461.8
23	860.47	2210.1	3600.0	1104.72	2032.4	3538.6	1186.36	675.8	3600.0
24	1048.91	866.9	3600.0	1187.62	1454.1	3256.1	1248.43	2642.5	3324.6
25	830.26	2371.0	3598.8	1436.09	1205.8	3600.0	1480.63	2336.5	3600.1
26	819.56	3597.6	3600.3	1404.49	1173.9	3600.0	1471.74	1554.6	3600.3
27	1099.95	355.9	3600.1	1450.18	521.3	3600.0	1524.22	1308.2	3600.0
28	1078.27	985.2	3600.0	2738.31	2051.2	3600.1	2858.53	2576.9	3600.1
29	1179.01	3080.0	3600.0	2474.33	1406.5	3600.0	2575.28	1162.5	3600.2
30	1061.55	1834.4	3600.8	1948.72	1185.4	3600.3	2076.20	2021.4	3600.2
31	1464.04	288.8	3600.9	2506.99	2375.8	3600.2	2592.17	2102.2	3600.5
32	1352.61	1780.8	3600.1	2486.43	1664.8	3600.4	2605.10	2305.2	3600.5
33	1361.51	2531.7	3601.1	2504.00	1843.2	3600.2	2610.55	2221.2	3600.6
34	858.94	1941.9	3604.4	1466.06	1359.1	3601.3	1546.06	2184.4	3601.0
35	992.86	766.7	3600.3	1765.30	2061.7	3600.4	1985.44	2223.1	3600.2
36	678.87	1530.9	3603.9	1909.88	2265.8	3600.1	1946.66	2626.3	3600.9
Aver.	792.31	772.9	1757.4	1216.08	750.7	1837.4	1266.61	977.3	1831.9

53.48%). Column $\overline{z}_{2-5}$ (sequential) shows that a relatively smaller further increase (on average by 4.15%) is produced by the constraint on sequential loading. The total CPU times are very close for the two versions, partially also due to the fact that for the largest instances with $n > 75$ the time limit of one hour was almost systematically reached. The time needed to find the best solution is lower for the unrestricted version.

In order to gain some insight into the quality of the loadings provided by Tabu Search, Table 7 reports, for each class, the following average values computed over the corresponding 36 instances, separately for the unrestricted and the sequential case.

v = number of available vehicles;

$\overline{v}$ = number of routes in the solution;

$\#clients$ = average number of clients per route;

$\#items_{\min}, \#items_{\max}$ = average minimum and maximum number of items per route;

$\#items_{\text{av}}$ = average number of items per route;

$\%area$ = average percentage of used loading surface.

No significant differences are observed between the two cases but, not surprisingly, in the unrestricted case the loading appears to be slightly easier.

Table 7: Loading information for classes 1 – 5.

		Unrestricted					
Class	v	$\overline{v}$	$\#clients$	$\#items_{\min}$	$\#items_{\text{av}}$	$\#items_{\max}$	$\%area$
1	8.89	8.78	8.91	–	–	–	–
2	16.47	16.28	4.80	4.72	7.25	9.36	76.39
3	17.50	17.22	4.59	6.39	9.17	11.47	77.12
4	17.86	17.56	4.45	6.75	10.93	14.47	77.74
5	18.00	14.78	5.21	10.00	15.56	20.00	75.51
av	15.74	14.92	5.59	6.97	10.73	13.83	76.69
		Sequential					
Class	v	$\overline{v}$	$\#clients$	$\#items_{\min}$	$\#items_{\text{av}}$	$\#items_{\max}$	$\%area$
1	8.89	8.78	8.91	–	–	–	–
2	16.47	16.44	4.67	4.69	7.08	9.17	74.63
3	17.50	17.39	4.54	6.28	9.07	11.33	76.26
4	17.86	17.67	4.41	7.31	10.82	14.50	77.02
5	18.00	15.11	5.16	9.78	15.29	20.22	74.00
av	15.74	15.08	5.54	7.01	10.57	13.81	75.47

5.1 Robustness

The robustness of the algorithm was tested by running it with different values of the parameters. These tests were performed on the instances of Tables 3 and 4.

The algorithm is quite sensitive to the number of iterations allowed. As previously mentioned, the maximum number of iterations was experimentally determined as $2n^2v$, halting the execution, in any case, after one hour of CPU time. Other rules were tested, leading to a worse average ratio of solution values over computing times. In particular, replacing $2n^2v$ with a fixed limit of value 200 000 improved the average gap from -0.19% to -0.63%, and the number of negative gaps from 19 to 22, but the average computing time increased from 116 to 1547 seconds. Trying 100 000 iterations and 30 minutes halved the CPU time but decreased by one half the average improvement . Attempts with 300 000 iterations did not lead to any significant variation.

The algorithm proved to be particularly robust with respect to the policy adopted for the penalization factor γ, which was finally set to $\sqrt{nv}$, as in [28] and [8]. Other functions attempted, such as $\sqrt{nM}$, nv and nM, produced marginal variations. The approach proved to be also quite robust with respect to the length of the Tabu list, ϑ. The value $\vartheta = \log(nv)$ was finally chosen, but other functions of n and v produced similar results. Functions of the total number of items M proved instead to be a bad choice. As for the size p of the neighborhood, we tested the values 5, 9, 10, 11 and 20. The best choice proved to be $p = 10$. The results were close for $p = 9$ and $p = 11$, but definitely worse for $p = 5$ and $p = 20$.

Different rules were tried for the updating of parameters α and β, but none managed to find the results obtained through the rule described in Section 3.3. The behavior of the algorithm also consistently changed when changing the maximum limit given to λ, the maximum number of calls to $\mathrm{TP}_{\mathrm{2SL}}$ and resp. $\mathrm{TP}_{\mathrm{2UL}}$. The values 1, 2, 3, 4, 5 and 10 were attempted, and the value 4 was chosen. With the value 5, the CPU time increased by 3%, improving the average gap only by 0.02%. An almost symmetric effect was produced by using the value 3.

The periodic intensification (see Section 4.1) is performed every ω iterations. Several values were tested for ω: 10, 5, 4, 3, 2 and 1. It turned out that frequent executions are useful for small-size instances. The best results were obtained by setting ω to 1 for $n \leq 40$, and to 5 for larger values.

6 Conclusions

We have considered an extension of the classical vehicle routing problem in which two-dimensional packing constraints are introduced. This problem features of two classical combinatorial optimization problems. We have developed, implemented and tested a Tabu Search algorithm which was applied to a large test set obtained by modifying instances from the literature. On instances solved optimally by branch-and-cut, the proposed algorithm also finds an optimal solution in 33 cases out 58. On instances for which the branch-and

cut was interrupted before completion, our heuristic finds feasible solutions that are on the average 0.21% better. The Tabu Search algorithm was also applied to a larger set of 360 instances for which the optimum was not known. It succeeded in identifying a feasible solution in all cases. Our results also show that the introduction of a loading constraint considerably increases solution costs.

Acknowledgements

We thank the Ministero dell'Istruzione, dell'Università e della Ricerca (MIUR), the Consiglio Nazionale delle Ricerche (CNR) and the Canadian Natural Sciences and Engineering Research Council under grants OPG0038816 and OPG0039682. Their support is gratefully acknowledged. The computational experiments were executed at the Laboratory of Operations Research of the University of Bologna (LabOR). Thanks are due to two anonymous referees for their valuable comments.

References

[1] M.A. Boschetti and A. Mingozzi. The two-dimensional finite bin packing problem. Part I: New lower bounds for the oriented case. *4OR*, 1:27–42, 2003.

[2] M.A. Boschetti and A. Mingozzi. The two-dimensional finite bin packing problem. Part II: New lower and upper bounds. *4OR*, 1:135–147, 2003.

[3] N. Christofides and C. Whitlock. An algorithm for two-dimensional cutting problems. *Operations Research*, 25:30–44, 1977.

[4] G. Clarke and J.W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12:568–581, 1964.

[5] J.-F. Cordeau, M. Gendreau, A. Hertz, G. Laporte, and J.-S. Sormany. New heuristics for the vehicle routing problem. In A. Langevin and D. Riopel, editors, *Logistics Systems: Design and Optimization*. Kluwer, Boston, 2005 (to appear).

[6] J.-F. Cordeau, M. Gendreau, and G. Laporte. A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks*, 30:105–119, 1997.

[7] J.-F. Cordeau and G. Laporte. Tabu search heuristics for the vehicle routing problem. In C. Rego and B. Alidaee, editors, *Metaheuristic Optimization via Memory and Evolution: Tabu Search and Scatter Search*, pages 145–163. Kluwer, Boston, 2004.

[8] J.-F. Cordeau, G. Laporte, and A. Mercier. A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational Research Society*, 52:928–936, 2001.

[9] G.B. Dantzig and J.H. Ramser. The truck dispatching problem. *Management Science*, 6:80, 1959.

[10] K.A. Dowsland. Some experiments with simulated annealing techniques for packing problems. *European Journal of Operational Research*, 68:389–399, 1993.

[11] S.P. Fekete and J. Schepers. On more-dimensional packing III: Exact algorithms. *Operations Research*, 2003 (to appear).

[12] R. Fukasawa, H. Longo, J. Lysgaard, M. Poggi de Aragão, M. Reis, E. Uchoa, and R.F. Werneck. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. Technical report, 2004 (submitted to *Mathematical Programming*).

[13] M. Gendreau, A. Hertz, and Laporte G. A tabu search heuristic for the vehicle routing problem. *Management Science*, 40:1276–1290, 1994.

[14] M. Gendreau, A. Hertz, and G. Laporte. New insertion and postoptimization procedures for the traveling salesman problem. *Operations Research*, 40:1086–1094, 1992.

[15] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13:533–549, 1986.

[16] F. Glover. Tabu seach – Part I. *ORSA Journal on Computing*, 1:190–206, 1989.

[17] F. Glover. Tabu seach and adaptive memory programming – advances, applications and challenges. In R.S. Barr, R.V. Helagson, and J.L. Kennington, editors, *Interfaces in Computer Science and Operations Research*, pages 1–75. Kluwer, Boston, 1996.

[18] F. Glover and M. Laguna. *Tabu Search*. Kluwer, Boston, 1997.

[19] M. Iori. *Metaheuristic algorithms for combinatorial optimization problems*. PhD thesis, DEIS, University of Bologna, Italy, 2004.

[20] M. Iori. Metaheuristic algorithms for combinatorial optimization problems. *4OR*, 3:163–166, 2005.

[21] M. Iori, S. Martello, and M. Monaci. Metaheuristic algorithms for the strip packing problem. In P. Pardalos and V. Korotkich, editors, *Optimization and Industry: New Frontiers*, pages 159–179. Kluwer, Boston, 2003.

[22] M. Iori, J.J. Salazar González, and D. Vigo. An exact approach for the symmetric capacitated vehicle routing problem with two dimensional loading constraints. Technical Report OR/03/04, DEIS, Università di Bologna, 2003.

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

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

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

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

[27] D. Pisinger and M. Sigurd. Using decomposition techniques and constraint programming for solving the two-dimensional bin packing problem. Technical Report 03/1, Department of Computer Science, University of Copenhagen, 2003.

[28] É. D. Taillard. Parallel iterative search methods for vehicle routing problems. *Networks*, 23:661–673, 1993.

[29] P. Toth and D. Vigo. *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, 2002.