A GRASPxELS for the vehicle routing problem with basic three-dimensional loading constraints

Philippe Lacomme, Hélène Toussaint, Christophe Duhamel

A GRASP×ELS for the vehicle routing problem with basic three-dimensional loading constraints

Philippe Lacomme[1], Hélène Toussaint[1],
Christophe Duhamel[2]

[1] Laboratoire d'Informatique (LIMOS, UMR CNRS 6158), Campus des Cézeaux,
63177 Aubière Cedex, France.

2 Institut Charles Delaunay (LOSI) and STMR (UMR CNRS 6279), Université de Technologie de Troyes,
BP 2060, 10010 Troyes Cedex, France

Abstract

This paper addresses an extension of the Capacitated Vehicle Routing Problem where the client demand consists of three-dimensional weighted items (3L-CVRP). The objective is to design a set of trips for a homogeneous fleet of vehicles based at a depot node which minimizes the total transportation cost. Items in each vehicle trip must satisfy the three-dimensional orthogonal packing constraints. This problem is strongly connected to real-life transportation systems where the packing of items to be delivered by each vehicle can have a significant impact on the routes. We propose a new way to solve the packing sub-problem. It consists of a two-step procedure in which the z-constraints are first relaxed to get a $(\boldsymbol{x};\boldsymbol{y})$ positioning of the items. Then, a compatible z-coordinate is computed to get a packing solution. Items can be rotated but additional constraints such as item fragility, support and LIFO are not considered. This method is included in a GRASP×ELS hybrid algorithm dedicated to the computation of VRP routes. The route optimization alternates between two search spaces: the space of VRP routes and the space of giant trips. The projection from one to the other is done by dedicated procedures (namely the Split and the concatenation algorithms). Moreover, a local search is defined on each search space. Furthermore, hash tables are used to store the result of the packing checks and thus save a substantial amount of CPU time. The effectiveness of our approach is illustrated by computational experiments on 3L-CVRP instances from the literature. A new set of realistic instances based on the 96 French districts are also proposed. They range from 19 nodes for the small instances to 255 nodes for the large instances and they can be stated as realistic since they are based on true travel distances in kilometers between French cities. The impact of the hash tables is illustrated as well.

Keywords: Vehicle routing problem, 3L-CVRP, 3D orthogonal packing, GRASP, Evolutionary local search

1 Introduction

The three-dimensional loading CVRP (3L-CVRP) is an extension of the CVRP where the dimension of the items (length, width and height) is also addressed. More formally, each vehicle of the homogenous fleet is defined by a weight capacity D and by a volume $V = L \times W \times H$ where L is the vehicle length, W is the vehicle width and H is the vehicle height (related to (x, y, z) coordinates). The demand of each client $i = 1 \ldots n$ consists of a set of m_i items of total weight d_i. Each item $k = 1 \ldots m_i$ is a three-dimensional cuboid of length l_{ik}, width w_{ik} and height h_{ik}. Each client must be visited by exactly one vehicle, which is assigned to a single trip. A trip t is a sequence $t = (t_0, t_1, \ldots, t_{n(t)}, t_{n(t)+1})$ of $n(t)$ clients where $t_0 = t_{n(t)+1}$ corresponds to the depot. Each trip must be both "weight-feasible" and "packing-feasible". A trip t is "weight-feasible" if the total weight of carried items does not exceed the vehicle capacity, *i.e.* $\sum_{i \in t} d_i \leq D$. It is "packing-feasible" if the client items can be loaded into the vehicle without overlapping and if it satisfies the classical orthogonal three-dimensional packing constraints. A solution to the 3L-CVRP is a set of "weight-feasible" and "packing-feasible" trips which involves all the clients.

The 3L-CVRP has been addressed by Gendreau *et al.* [10], by Fuellerer *et al.* [11] and more recently by Bortfeldt [12]. Only medium instances have been considered since three-dimensional

packing problems are hard to solve, in fact much harder than their two-dimensional counterparts. The seminal publication of [10] introduced a tabu search algorithm that iteratively uses a tabu search procedure for solving the inner loading sub-problem. Fuellerer *et al.* [11] introduce a highly efficient ant colony optimization algorithm which takes advantage of both fast packing heuristics for the loading sub-problem and of effective heuristics for the routing problem. Bortfeldt [12] proposes a tabu search for the routing problem and a tree search for the loading sub-problem. Its results are slightly worse than Gendreau and Fuellerer but the computational time is far lower on most instances, hence a different quality/time trade-off. However, its computer [12] is around 4 times faster than in [10] and [11]. These three publications also consider additional constraints as item fragility, LIFO unloading and support. Note that both instances from the literature and real-world instances were used by [10] to evaluate the performance of their method.

The packing problem within the 3L-CVRP falls into the Three-Dimensional Packing Problem (3PP) category since each trip has to be "packing-feasible". This corresponds to an existence checking problem and not an optimization problem. A 3PP instance consists of a set of items $I = \{1, \dots, n\}$ which have to be packed into a bin $B = (L; W; H)$ of length L, of width W and of height H. An item $i = (l_i, w_i, h_i)$ has a length l_i, a width w_i and a height h_i. A 3PP solution can be fully defined by the location of each item i, denoted (x_i, y_i, z_i), into the bin. This position corresponds to the coordinates of its bottom-left corner. Item rotation is only allowed in the (x, y) plane as other rotations may be prohibited in the corresponding real-life application (items often have a "top" side for instance). Moreover the packing must be orthogonal, *i.e.* the items edges must be parallel to the sides of the bin.

Additional constraints may also be considered:

- **fragility**: the items tagged as "fragile" cannot be located under another item;
- **support**: each item must have a minimum "supporting area", *i.e.* a given percent of its basis must correspond to the top of other items (or by the floor of the bin);
- **LIFO:** the items of any client in the trip can be unloaded by only using straight movements, *i.e.* the items of a client *i* must not be blocked by items of yet unvisited clients.

Such constraints correspond to realistic considerations in the industrial context of transportations and logistics. They are mandatory in many situations as CVRP solutions involving fully-loaded or nearly fully-loaded vehicles may not be 3L-CVRP feasible in practice, thus greatly reducing the interest of basic CVRP commercial solvers. The 3L-CVRP usually includes both three extra-constraints. We consider here the basic version where only three-dimensional constraints as well as rotations are considered. We propose an original way of addressing the 3PP since most previous publications rely on the same idea of insertion points (see Table 2). The inclusion of the three extra constraints (fragility, support and LIFO) is left for future work.

work	problem	method
(Gendreau et al., 2006)	packing	direct packing + tabu search
	routing	tabu search
(Fuellerer et al., 2010)	packing	direct packing
	routing	ant colony optimization
(Bortfeldt, 2012)	packing	direct packing + tree search
	routing	tabu search
our proposal	packing	2-step procedure based on a relaxation of the 3PP
	routing	GRASP×ELS

Table 1: Overview of the proposed methods for the 3L-CVRP

The paper is organized as follows: Section 2 presents the general framework used for solving the 3L-CVRP. The specific 3D loading problem is addressed in Section 3. Section 4 is dedicated to the numerical experiments, before concluding remarks.

2 GRASP×ELS framework for the 3L-CVRP

2.1 GRASP×ELS Principle

The GRASP×ELS [18] is a hybridization of the GRASP (Greedy Randomized Adaptive Search Procedure) metaheuristic and of the ELS (Evolutionary Local Search) metaheuristic combining the positive features of both methods. The GRASP [19] is a multi-start Local Search metaheuristic. At each GRASP iteration, an initial solution is constructed by using a greedy randomized heuristic. It is then improved by a local search and the best solution obtained at the end of each GRASP iteration is kept. The ELS [20] is an extension of the ILS (Iterated Local Search, [21]). At each iteration of the ELS, several copies of the current solution are built. Each copy is modified (mutation) before being improved by a Local Search. The best resulting solution is kept as the new current solution. The purpose of the ELS is to better investigate the neighborhood of the current local optimum before leaving it, while the GRASP aims at managing the diversity during the global solution space exploration. The framework we propose is a multi-start ELS in which the ELS is applied to the initial solutions generated by greedy randomized heuristics. Such an approach can also be viewed as a GRASP×ELS hybrid heuristic in which the ELS is used as Local Search. Besides combining GRASP with ELS, another important feature of our approach is the alternation between two solution spaces: the giant tour space and the 3L-CVRP solution space. By defining a dedicated exploration on each search space and by defining projections from one search space into the other one, one can more easily avoid being trapped in local optima. The high quality solutions obtained by Prins [20] for the VRP, alternating between two search spaces (giant tour and VRP solutions) or by Duhamel *et al.* [20] on the Heterogeneous VRP clearly illustrate the interest of such approaches.

Here, two solution representations are used: solutions encoded as giant tours (TSP tours on the *n* clients) and 3L-CVRP solutions encoded as the set of trips (see Figure 1). Converting a 3L-CVRP solution into a giant tour is done by the ***Concat*** procedure. It consists in removing the depot from each trip and then concatenating the resulting trips into a single one. The reverse operation, *i.e.* converting a giant tour into a 3L-CVRP, requires more work. It is usually done by a dedicated splitting procedure (***Split***) that relies on dynamic programming. Such an approach has been successfully applied to numerous routing problems including the Capacitated Arc Routing Problem, the Vehicle Routing Problem, the Location Routing Problem for instance, see [23] for a recent state of the art of Split in routing problems. As a giant trip is not a direct representation of a 3L-CVRP solution, we have chosen the inner ELS to work on 3L-CVRP solutions while GRASP focuses on giant tours.

A random heuristic generates an initial solution S (set of trips) at each iteration of GRASP. It is then transformed into a giant trip T before being perturbed in a way similar to the mutation operator in Genetic Algorithms. The resulting giant tour T' is split into 3L-CVRP trips which provides a solution S'. Then S' is improved using a Local Search operating on 3L-CVRP trips. The new solution S'' is associated to the giant trip T'' by trips concatenation and it becomes the incumbent solution (S, T). During ELS, nd "children" are generated out of S, each one being mutated then improved by the local search. The best child replaces S. The process is iterated until ne iterations have been done. The incumbent solution is updated before starting a new GRASP iteration.

The ***Local Search*** is done by a first improvement descent method using several classical VRP neighborhoods to improve the initial 3L-CVRP solution: 2-Opt within a trip, 2-Opt between two trips, Swap within a trip and Swap between two trips. The ***random heuristic*** is indeed a randomized version of both the Path-Scanning heuristic and the heuristic of Golden *et al*. Thus, each call is likely to produce a different solution. The ***mutation*** operator is defined on the giant tour $T = (T_1, \dots, T_{n(T)})$, where T_i is the i^{th} trip, $n(T)$ being the number of trips in T. It first generates a new giant trip by modifying the concatenation order. Then some clients are exchanged to get a new giant trip T'.

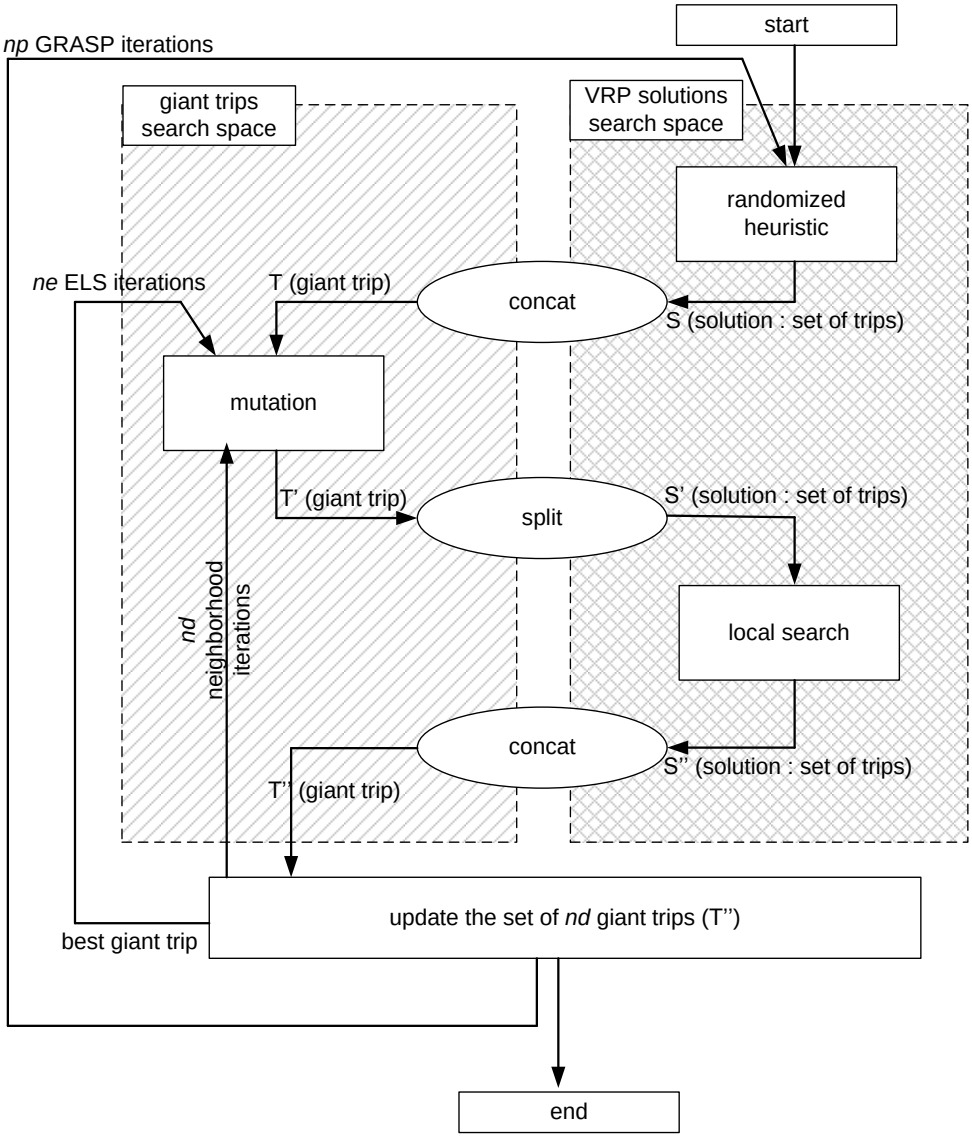


Figure 1: GRASP×ELS with alternation between the two search spaces

2.2 Split procedure

As previously mentioned, Split is a key-procedure which converts a giant tour into a 3L-CVRP solution (with respect to the sequence). It is based on the classical Split procedure [20][27][28], tuned to address the specific 3L-CVRP constraints.

Split first builds an auxiliary digraph $H_T = (X, Y, Z)$ where X is a set of $n+1$ nodes indexed from 0 to n. Node 0 is a dummy node, while the nodes $1 \dots n$ correspond to the client sequence of the giant tour $T = (v_1, \dots, v_n)$. An arc (i, j) belongs to Y if a trip visiting clients $v_{i+1}$to v_j (included) is both weight-feasible and 3D-feasible. The weight of the arc $(i, j) \in Y$ corresponds to the trip cost $z_{ij} = c_{0v_i} + \sum_{k=i \dots j-1} c_{v_k v_{k+1}} + c_{v_j 0}$. Optimally splitting T can be done by computing a min-cost path from node 0 to node n in H. An initial label is set at node 0. The labels are then propagated from node to node in H along the arcs. The best label at node n is kept as the optimal split.

```
1.   procedure Split
2.   input parameters
3.     T: giant tour
4.   output parameters
5.     S: 3L-CVRP solution
6.   global parameter
7.     D  : vehicle capacity (weight)
8.     V  : vehicle
9.     d_i : total weight of items for client i
10.     s_i : set of items for client i
11.    v_i : total volume of items for client i
12.    c_ij : cost from client i to j
13.    n  : number of clients
14.  begin
15.  L_0^1 = (N, 0, −1, −1), S := ∅
16.  pos_last := 0
17.  for i := 1 to n do L_i := ∅ endfor
18.  for i := 0 to n - 1 do
19.    j := i + 1
20.    trip := ∅; client := ∅
21.    repeat
22.      prev := client
23.      client := T_j
24.      trip := trip + client
25.      if (j = i + 1) then
26.        trip_load   := d_client
27.        trip_cost   := c_depot,client + c_client,depot
28.        trip_volume := v_client
29.        set_items   := ∅
30.        size        := 0
31.      else
32.        trip_load   := trip_load + d_client
33.        trip_cost   := trip_cost +c_prev,client +c_client,depot −c_prev,depot
34.        trip_volume := trip_volume + v_client
35.        size        := size + 1
36.      endif
37.      check := (trip_load ≤ D) and (trip_volume ≤ V.Volume)
38.      if (check = true) then
39.        set_items   := set_items + s_client
40.        if (j ≥ pos_last) and (size > 1) then
41.           res := 3D_Check_trip(set_items, V)
42.        else res := true
43.        endif
44.        if (res = true) then // 3D packing successfully solved
45.          for p := 1 to NB_i do
46.            let L_i^p = (N_i^p, z_i^p, k, j) be the current label
47.            propagate on j: L := (N_i^p − 1, z_i^p + trip_cost, i, p)
48.            if (Check_Domination_On_Node(L_j, j, NB_j) = false) then
49.              call Insert(L, j, NB_j)
50.            endif
51.          endfor
52.        endif
53.    j := j + 1
54.    until (check = false) or (j > n)
55.    pos_last := j
56.  endfor
57.  if (NB_n > 0) then
58.    S := call extract_trips () //gives trips generated according to the label
59.  endif
60.  end
```

Algorithm 1: Split algorithm for the 3L-CVRP

Let $L_i^p = \left(N_i^p, z_i^p, k, j\right)$ be the p^{th} label assigned to node *i*. It corresponds to a feasible split of the initial clients $t_1 ... t_i$ into trips. N_i^p is the number of vehicles still available, z_i^p is the cost of the trips previously built and $(k; j)$ is the reference to its father label, *e.g.* L_j^k, the k^{th} label at node *j*. The initial label at node 0 is defined as $L_0^1 = (N, 0, -1, -1)$. It corresponds to the empty solution where all the vehicles are available. Propagating the label L_i^p along the arc $(i, j) \in A$ produces the label $L_j^q = \left(N_j^q, z_j^q, i, p\right)$ the following way:

- $N_j^q = N_i^p - 1$
- $z_j^q = z_i^p + z_{ij}$

Since a lot of labels are generated and stored at each node, the computational time can quickly grow. Thus dominance rules must be defined in order to stay time-efficient. A label L_i^p is said to dominate a label L_i^q if one of the following conditions holds:

$$\begin{cases} \left(N_i^p > N_i^q\right) \text{ and } \left(z_i^p \leq z_i^q\right) \\ \left(z_i^p < z_i^q\right) \text{ and } \left(N_i^p \geq N_i^q\right) \end{cases}$$

The critical path leading to the best final label defines the trips of the 3L-CVRP solution. The procedure **`Split`** is detailed in Algorithm 1. For each node *i*, NB_i gives the number of associated labels. The procedure **`Check_Domination_On_Node`** checks if the new label *L* is dominated by another label at node *j*. The procedure **`Insert`** inserts this label into the set of labels from node *j* and also removes the dominated labels. The number of labels is updated accordingly.

3 Proposal for a new 3D loading approach

As mentioned before, solving the 3PP is much harder than solving the 2PP, its 2D counterpart. The idea developed in [9] for the 2PP was based on a partial relaxation of the geometrical constraints. It cannot be used anymore as recovering from such a relaxation raises a lot of feasibility issues. Thus, we propose a two-step procedure. Let $I = \{1, \ldots, n\}$ be a set of items. The following two steps are performed to compute a solution to the 3PP:

- **Step 1**: (x_i, y_i) positions are computed for each item *i*. The 3D geometry of the items is relaxed: the height of the item is considered as a cost $c_i = h_i$. Thus the following "arrangement problem" has to be solved: "Let *I* be a set of rectangular items *i* defined by their length l_i, their width w_i and their cost c_i, and let a rectangular bin be defined by its length *L*, its width *W* and its capacity *C*. Find a position (x_i, y_i) for each item *i* of *I* in the bin such that (i) the packing is orthogonal, (ii) the sum of the overlapping items costs does not exceed C". This step is addressed in part 3.1.
- **Step 2**: given the (x_i, y_i) positions obtained in Step 1, the z_i coordinates are computed such that (x_i, y_i, z_i) positions lead to a 3PP solution for the set of items *I*. Thus a 3PP has to be solved in this step, except that the solution is already partially defined. This step is fully detailed in part 3.2.

A trip is feasible if (i) the total weight of the clients items does not exceed the vehicle capacity and if (ii) the items can be packed into the vehicle with respect to the 3PP constraints. Checking the first constraint is trivial. Checking the second constraint is trickier and we use the method described above. The global check is done by the **`3D_Check_trip`** procedure (see Algorithm 2). The procedure iteratively generates an ordered list *O* before checking it. It stops as soon as a packing has been found or when the maximal number of attempts has been reached. The procedure **`Solve_x_y_coordinate`** tries to solve the arrangement problem (step 1) which relies on the ordered set *O*. Upon success, **`Solve_z_coordinate`** is called (step 2). Otherwise, the **`Random_Neighboord_Generation`** generates a new list *O'* by randomly exchanging some items in *O*. Rotations are addressed by a random selection of item in *O* and by swapping their length and width.

```
1.   procedure 3D_Check_trip
2.   global variables
3.     nm, nm1, nm2 : maximal number of attempts
4.   input parameters
5.     O  : set of items
6.     V  : vehicle
7.   output parameters
8.     x_i  : x-position of item i
9.     y_i  : y-position of item i
10.    z_i  : z-position of item i
11.    ok : boolean (true upon success)
12.  begin
13.    k := 1, l := 1, j := 1
14.    ok := false
15.    while (k < nm) and (ok = false)        //main loop
16.      while (l < nm1) && (ok = false)      //step1 search for (x,y) coordinates
17.        O := Random_Neighboord_Generation(O)
18.        (ok, x, y) := Solve_x_y_coordinates (O, V)
19.        l := l+1
20.      endwhile
21.      if (ok = true) then                  //step2 search for z coordinate
22.        ok = false
23.        while (j < nm2) and (ok = false)
24.          O := Random_Neighboord_Generation(O)
25.          (ok, z) = Solve_z_coordinates(O, x, y, V)
26.          j := j+1
27.        endwhile
28.      endif
29.      k := k + 1
30.    endwhile
31.  end
```

Algorithm 2: trip checking for 3D

To the best of our knowledge, this kind of approach is original in 3D problems. Gilmore and Gomory proposed in 1965 a stack building approach [24]. It consists in packing items, stack after stack, by solving 2D packing problem for each stack. Our proposal is quite different since it does not solve as many 2D packing problems. Only one problem need to be solved in Step 1 (which can be seen as a relaxation of the 3PP and not as a 2PP) and its solution is transformed into a 3PP solution in Step 2.

3.1 Step 1: arrangement problem (relaxation of the 3PP)

The **arrangement problem** is defined as follows: let I be a set of rectangular items i defined by their length l_i, their width w_i and their cost c_i. Let a rectangular bin be defined by its length L, its width W and its capacity C. The problem consists in finding a (x_i, y_i) position for each item i of I in the bin such that (i) the packing is orthogonal, (ii) the sum of the overlapping items costs does not exceed C.

The arrangement problem has to be solved each time the packing feasibility is checked. Since the check has to be done each time a solution is modified, its time efficiency is crucial. Thus, for time efficiency, we propose a greedy (heuristic) approach where items are scanned in an ordered list O. The items in O are considered and tentatively placed into the bin while satisfying constraints (i) and (ii). This process is done by the **`Solve_x_y_coordinates`** procedure (see Algorithm 3).

The **`Solve_x_y_coordinates`** main loop uses a current position in the bin denoted by (`posx`, `posy`). It tries to pack as many items from O as possible at this position. Any successfully packed item from O is removed from O. The (`posx`, `posy`) position is first initialized at the origin (0, 0). It is then updated according to an increasing order of x-coordinates and y-coordinates. The way (`x`, `y`) coordinates are scanned allow us to state that an item i can be packed at the position (x, y) if:

$$\begin{cases} x + l_i < L \\ y + w_i < W \\ \mathrm{h}[x][k] + c_i \leq C \quad \forall\, k \in [y, y + w_i[\end{cases}$$

where h[x][k] is the sum of the items costs which are overlapping at the position (x, k).

The way the positions are scanned in the arrangement is crucial. One must look for empty spaces reduction above the items while limiting the items stow in order to be able to successfully solve the 3 dimensional packing in the following Step 2.

```
1.   procedure Solve_x_y_coordinates
2.   input parameters
3.     O  : ordered set of items
4.     V  : vehicle
5.   output parameters
6.     ok : boolean (true upon packing success)
7.     x_i  : x-position of item i
8.     y_i  : y-position of item i
9.   local parameters
10.    Lx  : item list ordered on x values
11.    h : 2-dimensional array representing the bin area
12.  begin
13.    Lx := {0}                          // ordered set of x value
14.    Ly := {0}                          // ordered set of y value
15.    ok := true
16.    iy := 1
17.    Initialize h to 0
18.    while ( (O ≠ ∅) and (ok = true) ) do
19.      posx := Lx[1]                    // first available position in Lx
20.      posy := Ly[iy]                   // next available position in Ly
21.      for i:=1 to Card(O) do
22.        item := O[i]
23.        if ( item can be packed at (posx, posy) ) then
24.          remove item from O
25.          (x_item, y_item) := (posx, posy)
26.          for k := posx to posx + item.length do
27.            for p := posy to posy + item.width do
28.              h[k][p] := h[k][p] + item.height
29.              add (posx + item.length) to Lx
30.              add (posy + item.width) to Ly
31.            endfor
32.          endfor
33.        endif
34.      endfor
35.      iy := iy + 1
36.      if (iy > Ly.size) then
37.        iy := 1
38.        remove Lx[1] from Lx
39.        if (Lx become empty) then ok := false endif
40.      endif
41.    endwhile
42.  end
```

Algorithm 3: packing items in step 1

The main drawback of this approach is its greediness. This means the local choices may lead to a packing failure even if a feasible packing exists. To prevent such wrong answers, one could consider a backtracki mechanism (like a tree search). However this would be computationally too expensive since **`Solve_x_y_coordinates`** is called a lot of times during the GRASP process. A partial workaround based on a look-ahead mechanism has been added. It consists in adding an extra condition when trying to pack one of the last three items from *O*: the candidate item *i* can be packed at the position (`posx`, `posy`) only if the remaining items from *O* can be packed afterwards. Setting a limit of

three remaining items has experimentally shown to be a good compromise between efficiency and time consumption.

A post processing step consists in spreading items over the bin. The way *x* and *y* coordinates are scanned leads to packing items into the bottom-left side of the bin. As a consequence, the opposite area (top-right part of the bin) is not addressed according to the best possible way. Thus packed items are scanned in the decreasing order of their right edge position. Each item is then shifted as much as possible to its right (without introducing new overlaps). The same process is applied on *y* coordinates. This step reduces the number of overlapping items and makes the problem at step 2 easier to be solved.

3.2 Step 2: solving the 3PP using the partial solution computed at step 1

This step aims at computing a solution to the 3PP by using the solution found at Step 1. It consists in computing the z position of the items. The *x* and *y* positions have already been computed in Step 1. The idea is to scan the *z* coordinates, starting from 0. For each *z* value, as much items as possible are packed respecting their (x, y) position. This process ends when all items are packed or when the top of the bin is reached. The **`Solve_z_coordinates`** procedure is fully described in Algorithm 4.

```
1.   procedure Solve_z_coordinates
2.   input parameters
3.     O  : ordered set of card(O) items
4.     x  : set of positions in x (x_i = x-position of item i)
5.     y  : set of positions in y (y_i = y-position of item i)
6.     V  : vehicle
7.   output parameters
8.     ok : boolean (true upon 3BPP success)
9.     z  : set of positions in z (z_i = z-position of item i)
10.  local parameters
11.    h : array [1…L][1…W]      //h[x][y] = height already reached at (x,y)
12.  begin
13.    z := 0
14.    ok := true
15.    while ( ok = true ) do
16.      for ( k := 1 to card(O) ) do
17.        item := O[k]
18.        if ( item can be packed in position (item.x, item.y, z) ) then
19.          update h
20.          item.z := z
21.          remove item from O
22.        endif
23.        if (z + item.height > V.height) then
24.          ok := false
25.        endif
26.      endfor
27.    endwhile
28.  end
```

Algorithm 4 : computing *z* coordinates (step 2)

3.3 3D packing resolution example

Let us consider the instance E023-05s.DAT from [10]: 5 clients have to be visited for a total of 12 items, detailed in Table 2.

Item	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}	b_{11}	b_{12}
Client	c_{20}	c_{20}	c_{20}	c_1	c_{13}	c_{13}	c_{13}	c_7	c_7	c_{22}	c_{22}	c_{22}
Length	36	29	29	34	14	24	15	15	22	18	22	19
Width	10	10	8	10	9	7	10	11	6	13	8	12
Height	10	8	9	13	11	7	8	17	12	11	11	17

Table 2: set of items to be packed

3.3.1 Solving the arrangement problem (`Solve_x_y_coordinates`)

Let us consider the ordered list $O = (b_3, b_{12}, b_2, b_5, b_{10}, b_4, b_6, b_8, b_{11}, b_9, b_1, b_7)$. The next figures (from Figure 2 to Figure 7) illustrate the evolution of the arrangement process. The large rectangular area (60×25) corresponds to the bin. The small rectangles correspond to the items already packed and the associated values are the total cost for the associated area of the bin. The item cost corresponds to its height. The limit on the cost (the bin height) is set to 30. For each figure, the last packed item is filled with dotted lines.

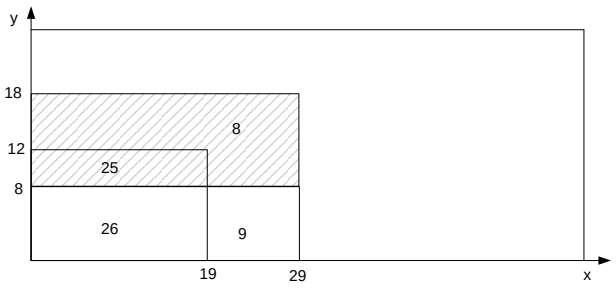


Figure 2: putting the first three items

The first two items b_3 and b_{12} are packed at position (0,0) and item b_2 is located at position (0,8) as stressed in Figure 2. The cost at the bottom-left side is the sum of b_3 cost and of b_{12} cost (9+17=26) since items b_3 and b_{12} are overlapping at this area. The costs in the other rectangles are computed the same way, considering the overlapped item.

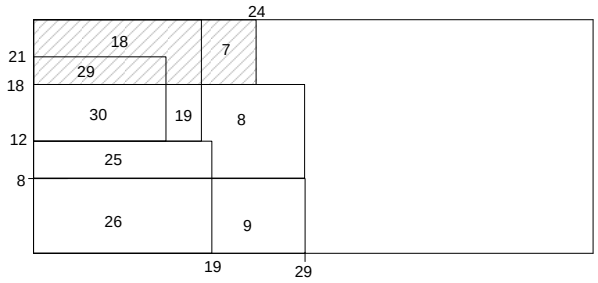


Figure 3: adding b_5, b_{10} and b_6

No other item can be packed in (0,8). Thus the next position investigated is (0,12). The items remaining in the list are checked: the first packable item is b_5. It is packed, as well as b_{10}. Then the position (0,18) is eligible for packing b_6, which leads to the arrangement in Figure 3.

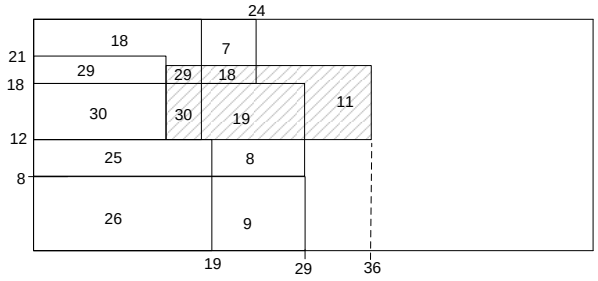
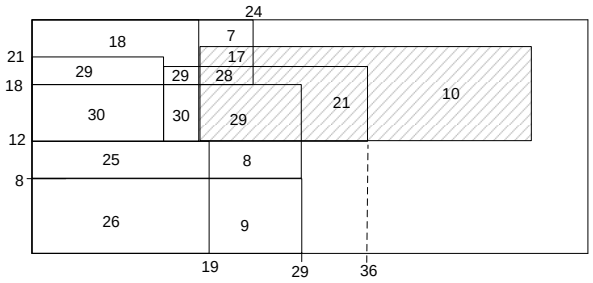


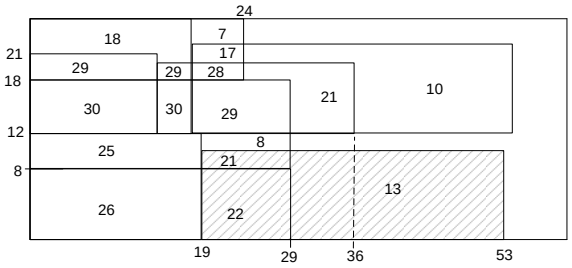
Figure 4: adding b_{11}

The method skips to abscissa 14, considering positions (14,0), (14,8) and (14,12). The item b_{11} can be placed at (14,12) leading to the packing solution of Figure 4.



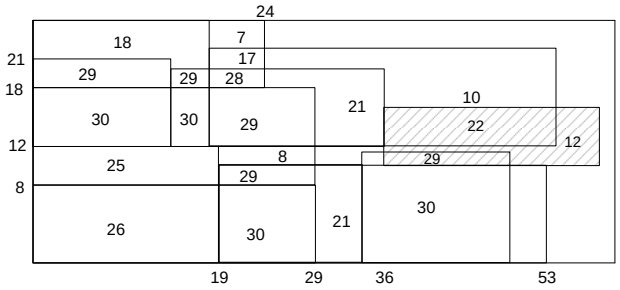
No item can be put at the next positions investigated. The first interesting position is (18,12) where item b_1 can be placed.

Figure 5: adding b_1



The next interesting position is (19,0) where b_4 is placed. The remaining items to pack are b_8, b_9, and b_7.

Figure 6: adding b_4



Only b_7 can be placed at position (19,0). The next position successfully investigated is (34,0) where b_8 is placed and finally b_9 is placed at (36,10).

Figure 7: Final arrangement

The **`Solve_x_y_coordinates`** procedure has produced a compact arrangement and the computed position for each item is reported in Table 3.

Item	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}	b_{11}	b_{12}
x-coordinate	18	0	0	19	0	0	19	34	36	0	14	0
y-coordinate	12	8	0	0	12	18	0	0	10	12	12	0

Table 3: Items position after resolution of the arrangement problem

3.3.2 Items shift

Shifting the items is done iteratively along the *x*-axis and then along the *y*-axis until no further shift can be done. This process leads to the new items coordinates in Table 4.

Item	b_1	b_2	b_3	b_4	b_5	b_6	b	b_8	b_9	b_{10}	b_{11}	b_{12}
x-coordinate	24	1	1	26	2	0	30	45	38	6	16	7
y-coordinate	15	8	0	5	16	18	5	4	19	12	17	0

Table 4: Items position after items shift

3.3.3 Items packing in *z* (`Solve_z_coordinates` procedure)

The sequence from Figure 8 to Figure 13 illustrates the way the packing is built by `Solve_z_coordinates`. The ordered set of items is $O = (b_2, b_7, b_3, b_5, b_{11}, b_4, b_{10}, b_8, b_1, b_{12}, b_6, b_9)$. First coordinate $z = 0$ is investigated and as many items as possible are packed at this current z according to their (x,y) position and according to the O. Thus b_2, b_3, b_7, b_8, b_6 and b_9 are packed at $z = 0$ (see Figure 8). The current z is updated to the smallest available height, *i.e.* $z = 8$. Items b_5 and b_{11} can be packed leading to the partial vehicle load shown in Figure 9.

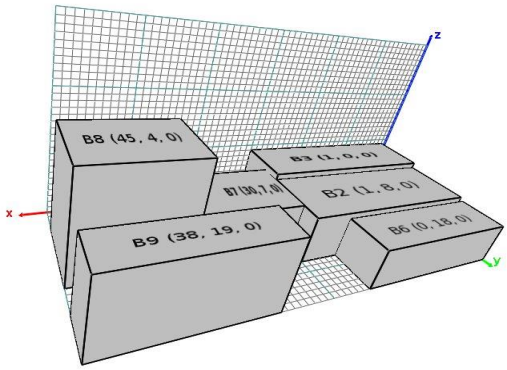


Figure 8: Packing items at *z* = 0

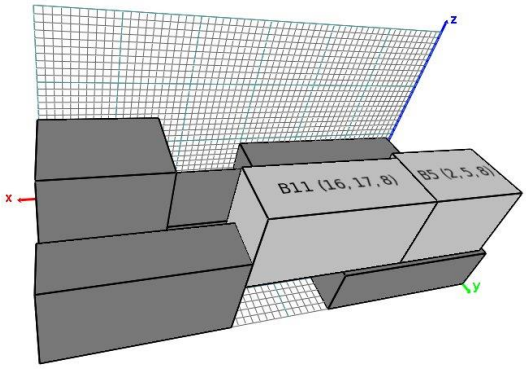


Figure 9: Packing items at *z* = 8

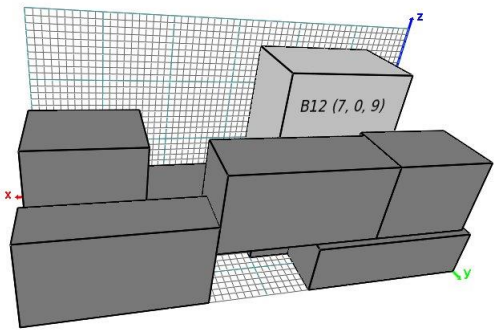


Figure 10: Packing items at *z* = 9

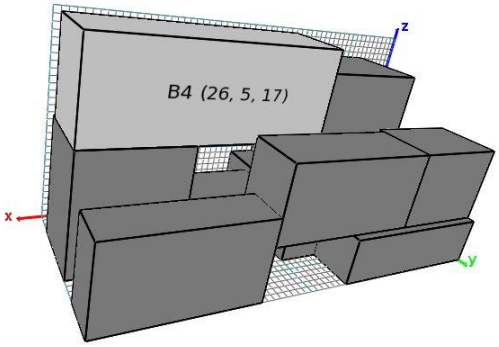


Figure 11: Packing items at *z* = 17

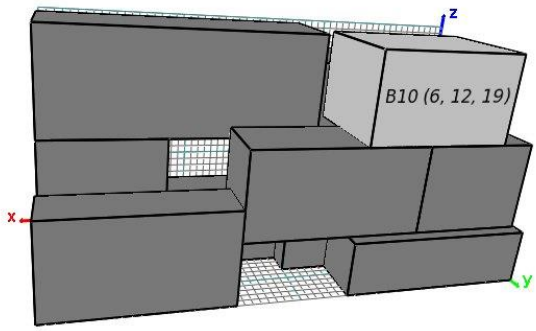


Figure 12: Packing item b_{10} at *z* = 19

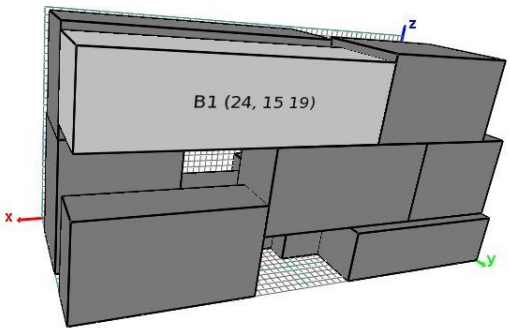


Figure 13: Packing item b_1 at *z* = 19

This process iterates with *z* = 9 where only b_{12} can be packed (Figure 10), $z = 17$ where only b_4 is packed (see Figure 11) and finally $z = 19$ where b_{10} and b_1 are packed. This leads to the final items packing in Figure 13.

Packing the items based on increasing values on the *z*-coordinate usually produces dense layers with as much items as possible packed at the same time. The final 3D-loading solution is shown in Table 5.

Item	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}	b_{11}	b_{12}
x-coordinate	24	1	1	26	2	0	30	45	38	6	16	7
y-coordinate	15	8	0	5	16	18	5	4	19	12	17	0
z-coordinate	19	0	0	17	8	0	0	0	0	19	8	9

Table 5: 3D-packing solution

3.4 Improvements on the computational and on the storage requirements

A lot of trips are evaluated during the optimization process. Moreover, the same trip can be evaluated several times. Thus, a way to save time consists in avoiding unprofitable calls to **`3D_Check_trip`** (several runs with identical parameters) by saving the result (true or false) of each trip feasibility check. A dedicated data structure is used and it is updated along the GRASP×ELS process.

A combination of data structures can be introduced: three matrices are dedicated to trips which deliver from 2 to 4 customers. Trips with a single client are trivially feasible, unless the instance is unfeasible. Note that items for one customer can be packable or not depending if items rotations are allowed or not. These matrices provide a *O*(1) check if the trip has already been checked, either being packing-feasible or not. Otherwise the feasibility check is performed and the result is stored into the corresponding matrix. The major drawback is the huge memory footprint, especially for the last 4-dimensional matrix. Another data structure is used for trips involving more than 4 clients. It is a red-black tree (self-balancing binary search tree), see the seminal contributions [25] [26]. In associative data structures, each element is associated to a key which is used to find it back. Here the key corresponds to the set of clients of the trip without any relative order consideration. In order for the storage to be efficient, the relation between the keys and the trips should be a 1-1 correspondence. We propose the following key computation: given a trip $t = (t_0, t_1, \dots, t_{n(t)}, t_{n(t)+1})$, its key is generated by first computing the number of clients $n(t)$ in the trip. Then the client identification numbers are concatenated in the increasing order, leading to a value $v(t)$. For example, if $t = (0, 15, 8, 6, 12, 3, 0)$, then $n(t) = 5$ and $v(t) = 3681215$. Such an order is total since it is always possible to compare two different trips t and t':

$$t < t' \Leftrightarrow \begin{cases} n(t) < n(t') \\ \text{or} \\ \big((n(t) = n(t')) \text{ and } (v(t) < v(t')) \big) \end{cases}$$

The search in a red-black tree is done in *O*(log(*n*)) where *n* is the size of the tree.

The **`Compute_Load`** procedure (see Algorithm 5) is in charge of evaluating a trip. This happens if the trip has never been evaluated or if it has been submitted less than p unproductive attempts. For convenience, `Store(t)` denotes the storing evaluation of the trip t. It is independent of the structure used to store the trip. `Store(t)` has the following meaning:

$$Store(t) = \begin{cases} 0, & \text{if } t \text{ has never been evaluated} \\ 1, & \text{if } t \text{ is 3D feasible} \\ -\alpha, & \text{if } t \text{ has been evaluated } \alpha \text{ times and still not 3D feasible} \end{cases}$$

```
1.   procedure Compute_Load
2.   input parameters
3.     t  : trip
4.     p  : number of 3D trip evaluation attempts
5.     V  : vehicle
6.   output parameters
7.     ok : boolean (true upon success)
8.   local parameters
9.     O : set of items in trip t
10.  begin
11.    ok := false
12.    switch case:
13.      case Store(t) = 1
14.        ok := true
15.      endcase
16.      case Store(t) = -p
17.        ok := false
18.      endcase
19.      otherwise
20.        (x, y, z, ok) = 3D_Check_trip (O, V)
21.        if (ok = true) then
22.          Store(t) := 1
23.        else
24.          Store(t) := Store(t) - 1
25.        endif
26.      endcase
27.    endswitch
28.  end
29.
```

Algorithm 5: Vehicle Load Resolution

4 Computational experiments

All procedures have been implemented in C++ and compiled using g++. Numerical experiments have been carried out on a 2.1 GHz Opteron computer running Linux operating system. The CPU power has been evaluated at around 4140 Mflops/s. The numerical experiments are based on two instance sets:

- a set of instances introduced in [10] and widely used since;
- a new set of instances based on the 96 French counties. To the best of our knowledge, this is the first step towards the definition of realistic and available instances for the 3L-CVRP. They are available at http://www.isima.fr/~toussain, http://www.isima.fr/~lacomme/3lcvrp/3lcvrp.html, and http://www.isima.fr/~duhamel.

Table 6 reports the parameters setting for the two sets of instances.

parameter	definition	value
np	number of GRASP iterations	60
ne	number of ELS iterations	15 + min(6, nbVehicules)
nd	number of neighborhoods	10
p	maximal number of 3D trip evaluation	5

Table 6: parameters setting for the classical instances

4.1 Implementation and classical benchmarks used

We report results on the set of instances used in [10], [11] and [12]. It is referred as classical since it is used over all publications on the 3L-CVRP. The number of clients varies from 15 to 100 and the total number of boxes varies from 32 to 198. The number of vehicles varies from 5 for the small instances

to 28 for the largest ones. These instances can be downloaded at http://www.or.deis.unibo.it/research_pages/ORinstances/.

instance	*clients*	*items*	*vehicles*
01	15	32	4
02	15	26	5
03	20	37	4
04	20	36	6
05	21	45	6
06	21	40	6
07	22	46	6
08	22	43	6
09	25	50	8
10	29	62	8
11	29	58	8
12	30	63	9
13	32	61	8
14	32	72	9
15	32	68	9
16	35	63	11
17	40	79	14
18	44	94	11
19	50	99	12
20	71	147	18
21	75	155	17
22	75	146	18
23	75	150	17
24	75	143	16
25	100	193	22
26	100	199	26
27	100	198	23

Table 7: instances characteristics

The details of the GRASP×ELS solutions are available at the same webpages the new instances are proposed. The GRASP×ELS is compared with the Ant Colony Scheme of [11], and with the Tabu Search of [10] and [12].

The GRASP×ELS is a random search algorithm. Thus, to provide a fair comparative study with [10], [11] and [12], each instance has been solved ten times, the same way it has been done in the other experiments. We report the average cost as well as the average CPU time to get the best solution over the 10 replications. Note that the best solution found over the 10 runs is also kept with the corresponding CPU time to reach it. Using the information provided in [10] [11] and [12], the computational time of each method has been scaled by a speed factor available at http://www.intel.com/support/fr/processors/sb/cs-017346.htm (see Table 8). The speed factor 3.87 assigned to [12] means its processor is 3.87 times faster than in [11]. This is based on MFlops characteristics and it gives an indication of the CPU performances. It must be interpreted with due caution since information comes from CPU manufacturers and does not include the whole computer characteristics such as memory speed and the memory bus performances.

	Gendreau *et al.* [10]	Fuellerer *et al.* [11]	Bortfeldt [12]	GRASP×ELS
Computer	PIV 3 GHz	PIV 3.2 GHz	Intel 3.17 GHz	Opteron 2.1 GHz
OS	?	Linux	?	Linux
Language	C	C++	C	C++
MFlops	6 132 MFlops (1)	6 540 MFlops (1)	25 328 MFlops (1)	4 251 MFlops (2)
Speed factor	0.94	1	3.87	0.66
Time limit	1h	1h	1h	1h30
Nb of runs	1	10	10	10

(1) Intel MFlops in http://www.intel.com/support/fr/processors/sb/cs-017346.htm
(2) http://browse.geekbench.ca/geekbench2/view/463443

Table 9: comparative performance of processors

All previously published methods were benchmarked over 1 hour of computational time, *i.e.* 1 hour of computation is assigned for one run of the methods. Since the reference results [11] have been obtained on a computer which is estimated to be 1.5 times faster than ours, the GRASP × ELS time limit is set to 1h30.

4.2 Average results for 3L-CVRP instances

A summary of the results for the basic version of the 3L-CVRP is presented in Table 10. Thus, item fragility, support and LIFO constraints are not considered. Yet, items can be rotated. For each method, the number of time the method is the only one to get the best result (line 2), the number of time it is not the only one the produce the best result (line 3) and the number of time it does not produce the best result (line 4) are reported. The average value set over the runs and the average value on the best solutions found over the runs are also reported in the last two lines. The results show that the GRASP×ELS method finds the best solution for 16 out of 27 instances. It outperforms the Tabu search from Gendreau *et al.* [10], the Ant Colony Scheme from Fuellerer *et al.* [11] and the Tabu search from Bortfeldt [12] on the basic version of the 3L-CVRP. The average value 847.04 is also the best. More details on those results are provided in Tables A1 and A2. For those tables and for each instance, $\bar{s}$ corresponds to the target solution value, $\bar{t}$ is the CPU in seconds and s_{best} is the best result we have obtained over the 10 replications (t_{best} is the time spent in the corresponding replication). The best average results are highlighted in bold. Our best results that strictly improve the best know results are also set in bold.

	Gendreau et al. [10]	**Fuellerer *et al.* [11]**	**Bortfeldt [12]**	**GRASP×ELS**	**GRASP×ELS**
rotation?	yes	yes	yes	yes	no
nb best	0	2	1	15	14
nb equal	9	8	4	8	5
nb_worse	18	17	22	4	8
avg value	876.31	856.7	864.53	847.04	848.88
best value	?	?	?	841.96	845.48

Table 10: average GRASP×ELS performance, with and without rotations

Two versions of our method have been reported in the last two columns. The last column corresponds to of the results of GRASP×ELS when item rotation is not allowed. Thus, items must be packed the way they are defined in the instance (*x* and *y* dimension cannot be swapped to get a better 3D packing). Quite surprisingly, forbidding item rotation does not deteriorate much the quality of the solutions found and the results are very similar to the GRASP×ELS version with rotations. Checking the results (see the Tables A1 and A2 in the Appendix) shows that solutions without items rotation are identical to the solutions with item rotation for half the instances. This is an indication that many instances in the benchmark are biased in such a way that discarding items rotation might still lead to high-quality solutions. It is quite unfortunate as it reduces the impact of one decision set (the variables on the items rotation). This is also the reason for proposing a new set of instances for which items rotation is mandatory to get the best solutions and even to get feasible solutions on some instances.

4.3 Hash table performances on results

The hash table stores the results from the 3D packing heuristic in order to avoid unnecessary subsequent calls for the same route. The time saved can be quite large and the larger the hash table, the more the time saved. Thus, the method becomes more time efficient as iterations go since the hash table is progressively filled. The hash table is even kept from one replication to the other one.

The impact of the hash table can be easily evidenced on many instances including instance 07 in Table 11. For this instance, the iteration limit is set to 63000. The total time to perform those iterations is about 3079.4 s in the first replication. It quickly drops to 1475.4 s in the second iteration. The time difference corresponds to the packing results from the first replication kept in memory, since both replications have exactly the same settings. The total time decreases over the 10 replications, dropping from 3000 s to 700 s.

replication	s_{best}	total time (s)
1	732.51	3079.4
2	732.51	1475.4
3	725.70	1228.7
4	732.51	1082.4
5	727.27	959.7
6	725.70	840.6
7	732.51	854.6
8	727.54	792.8
9	725.43	729.9
10	727.03	728.6

Table 11: GRASP×ELS performances over the 10 iterations

4.4 Example of a 3L-CVRP solution

Let us consider the instance 08 in which 22 clients have to be visited, for a total of 43 items to load and 8 vehicles available. The GRASP×ELS provides a solution of value 730.66 (without item rotation, see Table A2). This improves the result computed by the Ant Colony Scheme [11]. This solution uses 5 trips, as shown in **Erreur ! Source du renvoi introuvable.**.

Trip 1: depot, 14, 17, 22, 20, 19, depot

Trip 2: depot, 11, 13, 9, 5, 4, 7, depot

Trip 3: depot, 16, 15, 3, 2, 1, 6, 12, depot

Trip 4: depot, 21, 8, 10, depot

Trip 5: depot, 18, depot

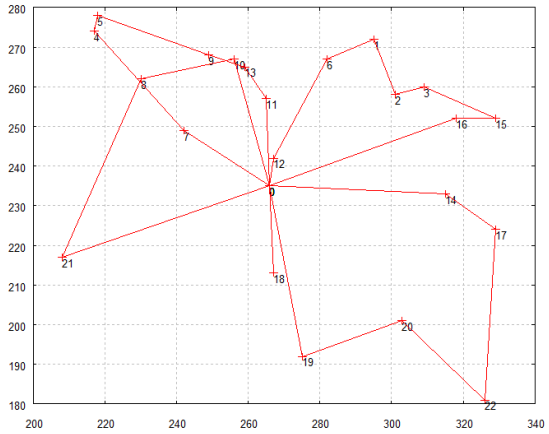


Figure 14: Solution for the instance 08

For each trip, Table 11 reports the total items weight, the total item volume and the trip cost.

trip	weight	volume	cost
1	1925	32861	212.611
2	2725	34932	142.299
3	994	26906	160.881
4	4425	21017	170.821
5	120	11628	44.045
		Total cost	**730.657**

Table 12: trips details

Trip 4 visits clients {21, 8, 10}. Table 13 reports the set of boxes and their dimensions for each client.

client 21	client 8	client 10
box 1 (24,15, 8)	box 1 (36,11,13)	box 1 (18,11, 8)
box 2 (13,14,14)	box 2 (27,11,17)	
	box 3 (34, 7,16)	

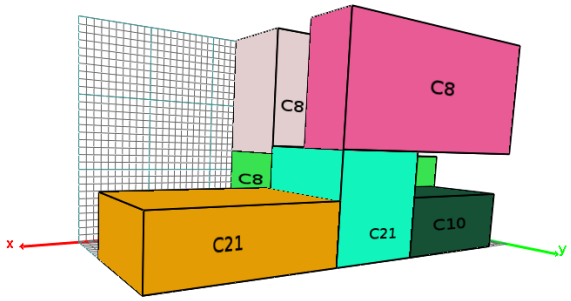
Table 13: list of boxes to pack for each client

A feasible corresponding 3PP solution with the bottom-left position of each item is as follows:

Client 21, box 1 at (31,0,0)
Client 21, box 2 at (18,11,0)

Client 8, box 1 at (0,0,17)
Client 8, box 2 at (0,0,0)
Client 8, box 3 at (0,11,14)

Client 10, box 1 at (0,11,0)



A 3D visualization tool can be downloaded at http://www.isima.fr/~toussain/.

4.5 Typical convergence rate of the GRASP

	Bortfeldt [12]			GRASP×ELS	
instance	$\bar{s}$	$\bar{t}$	scaled $\bar{t}$	$\bar{t}$	scaled $\bar{t}$
01	297.65	10.8	41.8	3.5	**2.3**
02	334.96	0.2	0.8	0.1	**0.0**
03	381.36	80.2	310.4	2.8	**1.9**
04	430.89	1.3	5.0	0.4	**0.3**
05	397.16	182.1	704.7	7.0	**4.6**
06	498.07	3.0	11.6	0.3	**0.2**
07	741.80	95.5	369.6	10.1	**6.6**
08	735.14	97.5	377.3	36.6	**24.2**
09	631.82	3.0	11.6	2.2	**1.4**
10	739.94	160.7	621.9	22.5	**14.8**
11	723.44	105.4	407.9	15.9	**10.5**
12	623.10	5.2	20.1	0.7	**0.4**
13	2348.48	314.9	1218.7	77.4	**51.1**
14	1234.54	313.1	1211.7	81.0	**53.5**
15	1202.34	314.7	1217.9	53.3	**35.2**
16	704.47	1.0	3.9	0.3	**0.2**
17	928.93	0.7	2.7	0.1	**0.1**
18	1108.37	294.5	1139.7	594.4	**392.3**
19	678.59	413.5	1600.2	1186.6	**783.2**
20	520.55	417.9	1617.3	1002.6	**661.7**
21	964.66	436.8	1690.4	2100.6	**1386.4**
22	1041.92	416.3	1611.1	1760.5	**1161.9**
23	995.22	417.5	**1615.7**	2456.9	1621.6
24	1053.41	295.8	1144.7	1346.8	**888.9**
25	1238.83	432.7	1674.5	2375.6	**1567.9**
26	1444.58	415.5	1608.0	1570.7	**1036.7**
27	1342.23	422.8	1636.2	2362.5	1559.3
Avg. Time		**209.4**	**810.4**	**632.3**	**417.3**

Table 14: Comparative study of convergence rate

We report the convergence speed of the GRASP to provide a fair comparative study with previously methods, especially [12] which is the last published method. Table 13 gives the average computational time over 10 GRASP×ELS iterations to reach results at least as good as in [12]. For each instance, $\bar{s}$ corresponds to the target solution value, $\bar{t}$ is the CPU in seconds and "scaled $\bar{t}$" is the CPU after applying the scaling factor. The average scaled time is lower than in [12] for all instances except for the instance 23. Besides, the target can be reached quite fast for the 17 first instances. For those instances, our proposal favorably compares with the Tabu search both in terms of solution quality and speed. Then, for the remaining 10 instances (instance 18 to 27), our method requires significantly more time to reach the target value. This should suggest the method in [12] offers a better scalability. However, the results computed by our method still compete with Bortfeldt's hybrid algorithm. Note that some results might differ from Table A1 since only 10 GRASP iterations are done. Thus the CPU time is much lower, especially on the large instances and even if no early stops have been implemented when the target value is reached. On instance 24, the average value (Table A1) is larger than the target value, due to 2 GRASP iterations (over the 60 done) providing a poor solution.

4.6 New benchmarks

Using the GIS system developed by Bajart and Charles [29], a new set of instance is proposed. Cities larger than 500 inhabitants are selected for each of the 96 French counties. They correspond to the clients and the size of the instances ranges from 19 to 255. The shortest path is computed between any pair of cities. This is done by using the Google web service. The distances thus correspond to the roadmap distance in kilometers between the cities and provide quite realistic and accurate data. To the best of our knowledge, those are the first available instances based on real counties. They can be divided into 4 categories:
- small: the 13 instances with less than 100 nodes;
- medium: 40 instances with 100 to 150 nodes;
- large: 33 instances with 150 to 200 nodes;
- very-large: 11 instances with more than 200 nodes.

The results for the 96 instances are available in Appendix 2. Table 14 reports synthetic results when rotations are allowed. Results without rotations are also available at http://www.isima.fr/~toussain/. For each category, $\bar{s}$ is the average value of the solutions found over the 10 replications of GRASP×ELS and $\bar{t}$ is the average CPU time. The last two lines correspond to the average value (s_{best}) of the best solutions found over the 10 replications and the associated time (t_{best}). The meaning is the same for all the tables in the Appendix. One can note the deviation grows as the size of the instances grows.

	small	medium	large	very large
$\bar{s}$	1069.24	2522.99	3936.89	5370.56
$\bar{t}$	3462.26	4949.80	5220.17	5493.75
s_{best}	1038.20	2457.20	3520.25	4424.02
t_{best}	3519.08	5148.23	5357.04	5532.92

Table 15: GRASP×ELS performance for the new instances

For the classical instances, allowing items rotation slightly improves the results, as shown in Table 10 and in Table 10 in more details. For the proposed new set of instances, 8 instances cannot be solved when rotations are forbidden since the items of some clients cannot be packed with the heuristic we introduced. Table 15 lists those instances along with the client for which item rotation is mandatory. Moreover, for one instance, GRASP×ELS found a solution with 16 vehicles when item rotation is forbidden while only 15 vehicles are available.

instances	client
DLT_3LCVRP_2b	10
DLT_3LCVRP_12	104
DLT_3LCVRP_09	205
DLT_3LCVRP_21	119
DLT_3LCVRP_30	25
DLT_3LCVRP_40	116
DLT_3LCVRP_49	83
DLT_3LCVRP_50	44

Table 16: Client packing failure with the heuristic when rotations are forbidden

5 Concluding remarks

This article considers an extension of the well-known CVRP in which three dimensional packing constraints must be addressed in each trip visiting clients. We consider the basic version, where only 3PP are addressed. This problem involves two combinatorial optimization sub-problems: vehicle routing and three-dimensional packing. The proposed packing heuristic relies on two steps: items coordinates are computed on the (x, y) plane, and then a feasible z-coordinate is calculated. Hash tables are also used to speed-up the search. The method we propose competes with the best published methods for the basic 3L-CVRP on the classic instances. Besides, a new set of realistic instances has been built with up to 255 clients, without any bias on the items rotation. Since our method is mostly dedicated to the 3L-CVRP with only rotation, we are currently investigating those additional constraints (item fragility, support and LIFO), trying to extend the original 3D-packing scheme we have introduced.

References

[1] Baldacci R., Battarra M. and Vigo D. Routing a Heterogeneous Fleet of Vehicles. In: The Vehicle Routing Problem: Latest Advances and New Challenges, Golden B. and Wasil E. (Eds). 2008, 3-27.

[2] Toth P. and Vigo D. An overview of vehicle routing problems. in: The Vehicle Routing Problem, Toth P and Vigo D. (Eds). SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, 2002, 1-26.

[3] Prins C. Two memetic algorithms for heterogeneous fleet vehicle routing problems. Engineering Applications of Artificial Intelligence, 2009, 22: 916-928.

[4] Cordeau J.F., Gendreau M., Hertz A., Laporte G. and Sormany J.S. New heuristics for the vehicle routing problem. In: Logistic systems: design and optimization, A. Langevin and D. Riopel (eds.), Wiley, New York, 2005, 279-298.

[5] Gendreau M., Iori M., Laporte G. and Martello S. A Tabu Search Heuristic for the Vehicle Routing Problem with Two-Dimensional Loading Constraints, Networks, 2008, 51(1): 4-18.

[6] Zachariadis E.E, Tarantilis C.D. and Kiranoudis C. A guided Tabu Search for the Vehicle Routing Problem with two dimensional loading constraints, European Journal of Operational Research, 2009, 3(16): 729-743.

[7] Fuellerer G., Doerner K.F., Hartl R.F. and Iori M. Ant colony optimization for the two-dimensional loading vehicle routing problem, Computers and Operations Research, 2009, 36(3): 655-673.

[8] Iori M., Salazar González J.J. and Vigo D. An exact approach for capacitated vehicle routing problems with two-dimensional loading constraints, Transportation Science, 2007, 41(2), 253–264.

[9] Duhamel C., Lacomme P., Quilliot A. and Toussaint H. A multi-start evolutionary local search for the two-dimensional loading capacitated vehicle routing problem, Computers and Operations Research, 2011, 38(3): 617-640.

[10] Gendreau M., Iori M., Laporte G. and Martello S. A Tabu Search Algorithm for a Routing and Container Loading Problem. Transportation Science, 2006, 40 (3): 342-350.

[11] Fuellerer G., Doerner K.F., Hartl R.F. and Iori M. Metaheuristics for vehicle routing problems with three-dimensional loading constraints, European Journal of Operational Research, 2010, 201(3): 751-759.

[12] Bortfeldt A., A hybrid algorithm for the capacitated vehicle routing problem with three-dimensional loading constraints, Computers and Operations Research, 2012, 39: 2248-2257.

[13] Hifi M., Kacem I.,Nègre S. and Wu L. A Linear Programming Approach for the Three-Dimensional Bin-Packing Problem. Electronic Notes in Discrete Mathematics, 2010, 36: 993-1000.

[14] Wu Y., Li W., Goh M. and de Souza R. Three-dimensional bin packing problem with variable bin height. European Journal of Operational Research, 2010, 202(2): 347-355.

[15] Bortfeldt A. and Mack D. A heuristic for the three-dimensional strip packing problem. European Journal of Operational Research, 2007, 183(3): 1267-1279.

[16] Allen S.D., Burke E.K. and Kendall G. A hybrid placement strategy for the three-dimensional strip packing problem, European Journal of Operational Research, 2011, 209(3): 219-227.

[17] De Almeida A. and Figueiredo M.B. A particular approach for the Three-dimensional Packing Problem with additional constraints, Computers and Operations Research, 2010, 37(11): 1968-1976.

[18] Prins C. A GRASP×Evolutionary Local Search Hybrid for the Vehicle Routing Problem, in: Pereira F.B. and Tavares J. (Eds). Bio-inspired Algorithms for the Vehicle Routing Problem, Studies in Computational Intelligence, publisher Springer, Berlin, 2009, 161: 35–53.

[19] Feo T.A. and Resende M.G.C. Greedy randomized adaptive search procedures, Journal of Global Optimization, 1995, 6: 109–33.

[20] Prins C. A simple and effective evolutionary algorithm for the vehicle routing problem, Computers and Operations Research, 2004, 31: 1985–2002.

[21] Lourenço H., Martin O. and Stützle T. Iterated Local Search. In: Glover F. and Kochenberger G. (Eds). Handbook of Metaheuristics. Springer New York, 2003, 57: 320-353.

[22] Duhamel C., Lacomme P. and Prodhon C. A hybrid evolutionary local search with depth first search split procedure for the heterogeneous vehicle routing problems. Engineering Applications of Artificial Intelligence, 25(2): 345-358 (2012).

[23] Duhamel C., Lacomme P. and Prodhon C. Efficient frameworks for greedy split and new depth first search split procedures for routing problems, Computers and Operations Research, 2011; 38(4):723-739.

[24] Gilmore P.C. and Gomory R.E. Multistage cutting stock problems of two and more dimensions, Operations Research, 1965, 13: 94–120.

[25] Guibas L.J. and Sedgewick R. A Dichromatic Framework for Balanced Trees, 19th Annual Symposium on Foundations of Computer Science, Ann Arbor, Michigan, 16-18 October, 1978: 8-21.

[26] Bayer R. and McCreight E.M. Organization and Maintenance of Large Ordered Indices. Acta Informatica 1972, 1: 173-189.

[27] Lacomme P., Prins C. and Ramdane-Chérif W. Competitive Memetic Algorithms for Arc Routing Problems, Annals of Operations Research, 2004, 131: 159-185.

[28] Beasley J.E. Route-first cluster-second methods for vehicle routing, Omega, 1983, 11: 403–408.

[29] Bajart V. and Charles C. Systèmes d'Information Géographique. 3rd year project report, ISIMA. http://www.isima.fr/~lacomme/students.html, 2009.

Appendix 1

instance	(Gendreau et al., 2006)		(Fuellerer et al., 2010)		(Bortfeldt, 2012)		GRASP×ELS			
	$\bar{s}$	$\bar{t}$	$\bar{s}$	$\bar{t}$	$\bar{s}$	$\bar{t}$	$\bar{s}$	$\bar{t}$	s_{best}	t_{best}
01	**297.65**	3.40	**297.65**	1.00	**297.65**	10.8	**297.65**	3.53	297.65	0.0
02	**334.96**	0.60	**334.96**	0.10	**334.96**	0.2	335.67	0.06	334.96	0.0
03	**362.27**	448.10	**362.27**	16.20	381.36	80.2	**362.27**	13.99	362.27	0.2
04	**430.89**	11.10	**430.89**	0.50	**430.89**	1.3	**430.89**	0.40	430.89	0.0
05	395.64	0.50	406.50	9.60	397.16	182.1	**379.43**	8.16	**379.43**	0.1
06	**495.85**	14.70	**495.85**	1.20	498.07	3.0	**495.85**	0.30	495.85	0.0
07	742.23	1.80	732.52	18.10	741.80	95.5	**725.43**	237.39	**725.43**	4.9
08	**735.14**	104.90	**735.14**	13.30	**735.14**	97.5	**735.14**	36.63	735.14	1.1
09	**630.13**	977.80	**630.13**	3.70	631.82	3.0	**630.13**	2.18	630.13	0.1
10	717.90	410.70	711.45	92.60	739.94	160.7	**687.57**	589.11	**687.57**	32.1
11	**718.24**	208.10	718.25	81.90	723.44	105.4	**718.24**	1453.35	718.24	1.8
12	614.60	1302.70	612.63	7.50	623.10	5.2	**610.05**	19.66	**610.00**	2.0
13	2316.56	2317.30	2391.77	174.50	2348.48	314.9	**2306.04**	1242.44	**2306.04**	86.9
14	1276.60	2121.30	1222.17	425.90	1234.54	313.1	**1186.96**	2423.64	**1184.44**	3600.2
15	1196.55	2916.14	1182.86	645.00	1202.34	314.7	**1161.20**	2144.72	**1161.11**	689.3
16	**698.61**	863.00	**698.61**	2.80	704.47	1.0	**698.61**	2.87	698.61	0.0
17	906.42	753.20	862.18	3.10	928.93	0.7	**861.80**	8.58	**861.79**	1.2
18	1124.33	2198.90	1112.18	1484.60	1108.37	294.5	**1084.26**	1893.69	**1078.41**	2030.8
19	680.29	1390.30	671.60	414.40	678.59	413.5	**670.44**	3322.67	**658.34**	3429.6
20	529.00	7007.50	515.39	1436.70	520.55	417.9	**510.95**	2892.97	**503.30**	1469.7
21	1004.40	6262.50	951.87	2105.70	964.66	436.8	**943.05**	4173.74	**921.25**	4697.4
22	1068.96	2078.70	1030.12	1218.40	1041.92	416.3	**1029.87**	3561.80	**1009.45**	3348.3
23	1012.51	4314.10	**971.05**	1231.70	995.22	417.5	987.06	3120.66	976.46	1889.1
24	1063.61	1052.50	1057.39	184.70	**1053.41**	295.8	1056.33	2610.20	**1047.75**	682.8
25	1371.32	500.90	**1207.97**	3986.10	1238.83	432.7	1232.73	4489.01	1219.77	4658.4
26	1557.12	1075.00	1453.39	2843.60	1444.58	415.5	**1415.15**	3484.63	**1393.76**	3066.6
27	1378.52	3983.20	1333.16	2208.30	1342.23	422.8	**1317.38**	3372.87	**1304.82**	2422.3
Avg. Cost	**876.31**		**856.67**		**864.53**		**847.04**		**841.96**	
Avg. Time		**1567.40**		**689.30**		**209.40**		**1522.56**		**1189.44**
Avg. scaled Time		**1473.35**		**689.30**		**810.38**		**1004.89**		**785.03**

Table A1: Solution values (rotations allowed)

instance	GRASP×ELS				GRASP×ELS			
	$\bar{s}$	$\bar{t}$	s_{best}	t_{best}	$\bar{s}$	$\bar{t}$	s_{best}	t_{best}
	rotations allowed				rotations forbidden			
01	297.65	3.53	297.65	0.0	297.65	1.01	297.65	0.0
02	335.67	0.06	334.96	0.0	335.67	0.06	334.96	0.0
03	362.27	13.99	362.27	0.2	362.27	5.31	362.27	0.2
04	430.88	0.40	430.88	0.0	430.88	0.19	430.88	0.0
05	379.43	8.16	379.43	0.1	379.43	1.56	379.43	0.0
06	495.85	0.30	495.85	0.0	495.85	0.34	495.85	0.0
07	**725.43**	237.39	**725.43**	4.9	732.51	104.85	732.51	0.1
08	735.14	36.63	735.14	1.1	**730.66**	115.78	**730.66**	0.1
09	**630.13**	2.18	**630.13**	0.1	633.72	2.52	633.72	0.1
10	**687.57**	589.11	**687.57**	32.1	704.97	1009.19	704.64	34.2
11	718.24	1453.35	718.24	1.8	**718.24**	225.10	718.24	13.6
12	610.05	19.66	610.00	2.0	**610.00**	11.62	610.00	2.7
13	2306.04	1242.44	2306.04	86.9	**2299.05**	111.17	**2299.05**	3.9
14	**1186.96**	2423.64	**1184.44**	3600.2	1194.57	2966.97	1192.29	612.0
15	**1161.20**	2144.72	**1161.11**	689.3	1163.34	1975.90	1163.23	1050.3
16	**698.61**	2.87	**698.61**	0.0	700.80	1.20	700.80	0.1
17	861.80	8.58	861.79	1.2	**861.79**	7.69	861.79	2.2
18	**1084.26**	1893.69	**1078.41**	2030.8	1091.61	2502.66	1091.28	4137.8
19	670.44	3322.67	**658.34**	3429.6	**665.45**	2489.16	662.96	1810.9
20	**510.95**	2892.97	**503.30**	1469.7	515.95	2340.49	513.28	3166.3
21	**943.05**	4173.74	**921.25**	4697.4	947.30	2503.90	940.72	2945.8
22	**1029.87**	3561.80	**1009.45**	3348.3	1039.64	2249.58	1021.02	3552.4
23	987.06	3120.66	976.46	1889.1	**980.63**	2194.71	**972.12**	3146.3
24	1056.33	2610.20	**1047.75**	682.8	**1053.60**	3964.98	1048.55	3697.1
25	1232.73	4489.01	1219.77	4658.4	**1224.56**	2898.75	**1209.81**	2564.6
26	**1415.15**	3484.63	**1393.76**	3066.6	1427.58	3053.36	1421.35	4711.4
27	**1317.38**	3372.87	1304.82	2422.3	1322.02	2449.77	**1298.84**	3617.4
Avg. Cost	**847.04**		**841.96**		**848.88**		**845.48**	
Avg. Time		**1522.56**		**1189.44**		**1229.18**		**1298.87**

Table A2: Solution values (with and without rotations)

Appendix 2

		GRASP×ELS			
instance	**county**	$\bar{s}$	$\bar{t}$	s_{best}	t_{best}
DLT_3LCVRP_01	Ain	1461.63	5361.81	1388.27	5418.4
DLT_3LCVRP_08	Ardennes	1065.51	2764.31	1041.54	2461.0
DLT_3LCVRP_10	Aube	1232.45	3685.94	1200.33	2580.6
DLT_3LCVRP_11	Aude	1565.7	4491.45	1517.85	4854.3
DLT_3LCVRP_36	Indre	2085.69	4594.17	2042.01	5324.0
DLT_3LCVRP_39	Jura	1828.25	4174.64	1758.5	5076.3
DLT_3LCVRP_43	Haute-Loire	1361.26	4117.87	1317.93	5301.0
DLT_3LCVRP_52	Haute-Marne	1176.91	3635.08	1146.78	2618.0
DLT_3LCVRP_55	Meuse	1103.22	2458.72	1078.8	1526.2
DLT_3LCVRP_70	Haute-Saône	1328.27	3777.12	1289.68	5052.7
DLT_3LCVRP_75	Paris	71.6	78.82	71.6	1.3
DLT_3LCVRP_82	Tarn-et-Garonne	1027.84	3704.84	1006.8	5135.5
DLT_3LCVRP_92	Hauts-de-Seine	256.88	2454.46	254.56	133.4
DLT_3LCVRP_93	Seine-Saint-Denis	219.44	3020.38	216.35	4722.1
DLT_3LCVRP_94	Val-de-Marne	254.01	3614.27	242.04	2581.4
Average		**1069.24**	**3462.26**	**1038.20**	**3519.08**

Table A3: GRASP×ELS performances on French counties ($n \leq 100$)

		GRASP×ELS			
instance	**county**	$\bar{s}$	$\bar{t}$	s_{best}	t_{best}
DLT_3LCVRP_03	Allier	2139.75	4603.22	2052.27	5330.8
DLT_3LCVRP_05	Hautes-Alpes	2075.94	5099.58	2026.35	5430.4
DLT_3LCVRP_06	Alpes Maritimes	3529.92	4613.60	3444.20	4476.5
DLT_3LCVRP_07	Ardèche	2484.92	4229.79	2434.31	4389.0
DLT_3LCVRP_12	Aveyron	2329.36	4804.46	2268.78	4844.2
DLT_3LCVRP_13	Bouches-du-Rhône	2352.33	4739.08	2296.61	4062.9
DLT_3LCVRP_16	Charentes	1932.30	5141.70	1872.63	5180.4
DLT_3LCVRP_17	Charentes Maritimes	1978.44	3841.00	1929.74	4671.5
DLT_3LCVRP_2A	Corse du Sud	2864.00	4926.72	2821.78	4917.4
DLT_3LCVRP_2B	Haute-Corse	3735.49	3626.75	3665.83	4552.5
DLT_3LCVRP_21	Côte d'Or	1977.07	5353.86	1885.21	5375.1
DLT_3LCVRP_25	Doubs	3266.13	5246.27	3159.12	5429.8
DLT_3LCVRP_26	Drôme	2700.61	4845.18	2648.82	5409.2
DLT_3LCVRP_28	Eure-et-Loire	2935.09	5322.84	2832.90	5414.4
DLT_3LCVRP_30	Gard	3538.71	5335.48	3436.54	5424.6
DLT_3LCVRP_31	Haute-Garonne	1818.05	4705.95	1776.06	4792.8
DLT_3LCVRP_34	Hérault	2765.50	4951.93	2736.49	5347.4
DLT_3LCVRP_40	Landes	3990.96	5308.65	3912.77	5438.5
DLT_3LCVRP_41	Loir-et-Cher	2852.58	5370.68	2759.98	5400.7
DLT_3LCVRP_47	Lot-et-Garonne	1809.77	5289.63	1778.14	5402.5
DLT_3LCVRP_48	Lozère	2961.16	4719.93	2875.28	5407.1
DLT_3LCVRP_51	Marne	2367.75	5259.18	2217.50	5400.4
DLT_3LCVRP_53	Mayenne	1564.51	4675.93	1528.12	5440.7
DLT_3LCVRP_60	Oise	2256.10	5100.09	2231.22	5300.0
DLT_3LCVRP_61	Orne	2022.89	4310.26	1954.09	5309.2
DLT_3LCVRP_66	Pyrénées Orientales	2916.57	5229.04	2872.76	5470.1
DLT_3LCVRP_68	Haut Rhin	2109.04	5279.39	2042.68	5038.4
DLT_3LCVRP_73	Savoie	3163.53	4655.41	3085.56	5188.6
DLT_3LCVRP_74	Haute-Savoie	3499.17	5096.06	3442.30	5427.2
DLT_3LCVRP_79	Deux-Sèvres	3156.48	5283.64	3077.39	4672.8
DLT_3LCVRP_81	Tarn	2159.38	5321.56	2097.33	5390.7
DLT_3LCVRP_83	Var	2882.99	5376.17	2811.55	4558.9
DLT_3LCVRP_84	Vaucluse	1720.03	5283.00	1662.31	5384.2
DLT_3LCVRP_85	Vendée	2449.78	5038.93	2361.79	5557.1
DLT_3LCVRP_87	Haute-Vienne	1540.89	5400.42	1498.54	5404.8
DLT_3LCVRP_88	Vosges	3005.10	5075.91	2919.94	5317.4
DLT_3LCVRP_89	Yonne	2362.85	5354.09	2314.52	5400.0
DLT_3LCVRP_90	Territoire de Belfort	658.54	4277.14	642.05	4674.6
Average		**2522.99**	**4949.80**	**2457.20**	**5148.23**

Table A4: GRASP×ELS performances on French counties ($100 < n \leq 150$**)**

		GRASP×ELS			
instance	**county**	$\bar{s}$	$\bar{t}$	s_{best}	t_{best}
DLT_3LCVRP_02	Aisne	3483.94	5376.81	3411.68	5407.4
DLT_3LCVRP_04	Alpes de Haute Provence	4001.26	5413.66	3898.77	5479.7
DLT_3LCVRP_09	Ariège	3660.89	3677.88	3457.07	5416.9
DLT_3LCVRP_14	Calvados	3404.01	5424.41	3341.09	5535.1
DLT_3LCVRP_15	Cantal	4692.26	5177.38	4597.80	5426.0
DLT_3LCVRP_24	Dordogne	5368.17	5323.03	5261.64	5274.6
DLT_3LCVRP_29	Finistère	5499.76	5070.86	5249.17	5425.4
DLT_3LCVRP_33	Gironde	4173.51	5537.61	4112.79	5700.6
DLT_3LCVRP_35	Ille-et-Vilaine	2746.49	5198.73	2680.55	5404.7
DLT_3LCVRP_37	Indre-et-Loire	3244.50	5231.54	3178.33	5232.7
DLT_3LCVRP_42	Loire	4126.93	5340.25	4007.88	5598.0
DLT_3LCVRP_44	Loire Atlantique	3614.01	5290.52	3513.68	5484.1
DLT_3LCVRP_45	Loiret	3243.62	5303.97	3208.11	5552.8
DLT_3LCVRP_50	Manche	6216.28	5369.54	6084.57	5458.6
DLT_3LCVRP_54	Meurthe-et-Moselle	3888.63	5372.69	3810.19	5476.1
DLT_3LCVRP_56	Morbihan	4415.26	5294.84	4347.93	5405.4
DLT_3LCVRP_57	Moselle	4710.03	5341.14	4599.94	5535.2
DLT_3LCVRP_59	Nord	3124.63	5538.69	3069.32	5422.7
DLT_3LCVRP_63	Puy-de-Dôme	3655.75	5281.63	3569.56	5430.4
DLT_3LCVRP_64	Pyrénées Atlantique	4137.24	5100.74	4077.55	5044.2
DLT_3LCVRP_67	Bas-Rhin	2266.82	5072.56	2201.15	4696.7
DLT_3LCVRP_69	Rhône	2056.44	5114.28	2031.52	4472.4
DLT_3LCVRP_71	Saône-et-Loire	14494.27	5390.60	4093.28	5378.3
DLT_3LCVRP_72	Sarthe	2632.41	5234.74	2548.62	5456.2
DLT_3LCVRP_76	Seine-Maritime	3057.31	4985.63	2977.96	5159.4
DLT_3LCVRP_77	Seine-et-Marne	2754.45	5086.19	2705.95	5424.8
DLT_3LCVRP_78	Yvelines	2744.03	5489.90	2671.51	5244.6
DLT_3LCVRP_80	Somme	2481.82	5349.43	2407.48	5555.6
DLT_3LCVRP_86	Vienne	4012.49	5069.05	3962.59	5418.2
DLT_3LCVRP_91	Essonne	2227.53	5114.22	2181.66	5231.5
DLT_3LCVRP_95	Val d'Oise	1908.77	5252.83	1868.38	5319.8
Average		**3936.89**	**5220.17**	**3520.25**	**5357.04**

Table A4: GRASP×ELS performances on French counties ($150 < n \leq 200$**)**

		GRASP×ELS			
instance	**county**	$\bar{s}$	$\bar{t}$	s_{best}	t_{best}
DLT_3LCVRP_18	Cher	4612.07	5464.63	4557.44	5463.3
DLT_3LCVRP_19	Corrèze	5255.32	5436.26	5081.43	5485.2
DLT_3LCVRP_22	Côtes d'Armor	4654.87	5563.20	4522.52	5666.8
DLT_3LCVRP_23	Creuse	3237.73	5379.35	3161.36	5403.6
DLT_3LCVRP_27	Eure	3775.79	5366.59	3652.64	5450.0
DLT_3LCVRP_32	Gers	4479.75	5527.63	4414.94	5477.3
DLT_3LCVRP_38	Isère	5482.05	5468.13	5313.02	5598.1
DLT_3LCVRP_46	Lot	5317.21	5625.37	5170.43	5532.8
DLT_3LCVRP_49	Maine-et-Loire	5968.21	5660.49	5832.03	5619.2
DLT_3LCVRP_58	Nièvre	4016.08	5540.16	3905.81	5791.7
DLT_3LCVRP_62	Pas-de-Calais	4593.95	5480.41	4484.43	5336.7
DLT_3LCVRP_65	Hautes-Pyrénées	13053.68	5412.83	2992.35	5570.3
Average		**5370.56**	**5493.75**	**4424.03**	**5532.92**

Table A4: GRASP×ELS performances on French counties ($n \geq 200$)