A Branch-and-Cut Approach for the Vehicle Routing Problem with Loading Constraints

Pedro Hokama
Instituto de Computação, IC, UNICAMP,
13084-971, Campinas, SP
e-mail: hokama@ic.unicamp.br

Flávio K. Miyazawa,
Instituto de Computação, IC, UNICAMP,
13084-971, Campinas, SP
e-mail: fkm@ic.unicamp.br

Eduardo C. Xavier,
Instituto de Computação, IC, UNICAMP,
13084-971, Campinas, SP
e-mail: ecx@ic.unicamp.br

ABSTRACT

We consider a transport system problem which involves route planning for vehicles with containers attached, and cargo accommodation. The specific problem considered is the vehicle routing problem with loading constraints. Its input consists of a weighted undirected graph where vertices corresponds to clients and edges costs represents the cost to move the vehicle along that edge. Each vertex has a demand for a set of items (which are two or three dimensional rectangles). There is also a special vertex that represents a depot where all items are stored at first. The problem is to determine a set of routes, each one leaving the depot, with minimum total cost, such that all clients are visited exactly once. The corresponding cargo for the clients of each route must fit into the container. As it would be too expensive to remove and rearrange the cargo at each point in the route, it is interesting that the goods to be unloaded in a client must be removed without moving the remaining cargo. These are called unloading constraints. In this paper we describe a branch-and-cut algorithm for the vehicle routing problem with loading constraints, using several techniques to prune the search tree. The problem of finding feasible loadings into the bins are solved using techniques adapted from the literature. These techniques involves branch-and-bound, constraint programming and metaheuristics. The efficiency of the algorithms are evaluated by an experimental analysis with instances from the literature, and compared with other algorithms.

KEYWORDS. Vehicle routing. Two-dimensional packing. Three-dimensional packing.

Branch-and-cut. Combinatorial Optimization.

1 Introduction

Several problems in transport systems involve route planning for vehicles with containers attached and accommodation of cargo into these containers. The route planning problem and the packing problem are well known problems in the research literature, and they were largely explored separately. However, in recent years there has been some interest in considering both problems combined, leading to better global solutions.

In the Vehicle Routing Problem with D-Dimensional Unloading Constraints (DL-CVRP), clients have a demand for goods stored into a depot, and k vehicles must be used to deliver these goods. Each travel from the depot to a client, or from a client to a next one has a cost. The problem is to find k routes leaving the depot, one route for each vehicle, such that all clients are visited exactly once, and such that the items of clients of a route can be packed in the vehicle's container. The objective function of the problem is to minimize the total cost of the routes. As it would be too expensive to rearrange the cargo at each visit, we add a condition that the goods to be unloaded in a client must be removed without moving the remaining goods, these are the so called *unloading constraint*. We consider two and three-dimensional unloading constraints.

The capacitated vehicle routing problem with two-dimensional unloading constraints (2L-CVRP) was first addressed by Iori et al. [17]. They introduced an exact method to solve the problem. Azevedo et al. [1] presented an exact method with different cut strategies and packing algorithms obtaining better solutions in most cases. Due to the difficulty of solving this problem exactly, several heuristics were also proposed. Gendreau et al. [11] presented a tabu search method to the 2L-CVRP. Fuellerer et al. [9] employed an ant colony method. Zachariadis et al. [28] introduced a guided tabu search method. Duhamel et al. [8] presented a GRASP approach for the case without unloading constraints. Approximation algorithms for the associated packing problem that considers unloading constraints were proposed by Silveira et. al. [5, 7, 6].

The capacitated vehicle routing problem with three-dimensional unloading constraints (3L-CVRP) was first considered by Gendreau et al. [10]. They presented a tabu search method to solve the problem. Junqueira et al. [18] presented an exact method for the 3L-CVRP with practical constraints where an integer linear programming model was able to solve instances of moderate size. For more references on routing and loading problems we refer to a survey by Iori et al. [16].

In the 2L-CVRP and 3L-CVRP problems we face a packing subproblem of determining if a set of items can be packed in a bin, this is the so called Orthogonal Packing Problem (OPP). In this problem it is given a set of items and a bin with bounded dimensions, and the objective is to find a placement of these items in the bin, or prove that it is unfeasible. When the items and the bins are two-dimensional objects (respectively three), we have the two-dimensional orthogonal packing problem - 2OPP (respectively the three-dimensional orthogonal packing problem - 3OPP). Clautiax et al. [3] presented an efficient method

for solving the 2OPP using Constraint Programming. Their work was further extended to include new bounds by Mesyagutov et al. [22], and to consider the three-dimensional case by Mesyagutov et al. [23]. Recently the problem was considered with the unloading constraint by Côté et al. [4], who presented an exact algorithm using branch-and-cut and a set of lower bounds.

The Constraint Programming (CP) paradigm has been proved to be a very efficient method for solving many different problems [26]. This paradigm was well known by other areas but just in the last decades was rediscovered by researchers in the combinatorial optimization community. The Integer Linear Programming (ILP) is probably the most used method for solving combinatorial optimization problems [27]. The use of both, CP and ILP together is a hot topic and has obtained good results for many problems [15].

In this paper we propose an algorithm to solve the 2L-CVRP and the 3L-CVRP. The Routing Problem is solved by a branch-and-cut algorithm using different cut strategies, including those presented by Lysgaard et al. [19] and Naddef et al. [24]. The demand of items for each client is used to add minimal infeasible cuts, that reduce branches, which can not lead to feasible solutions. The search for these cuts is done by solving the OPP. To this purpose, we tested a number of different originals and adapted algorithms. Among the algorithms to find feasible loadings into the bins, branch-and-bound, and constraint programming techniques were explored. The efficiency of the algorithms are evaluated by an experimental analysis with instances from the literature, and compared with other algorithms.

2 Orthogonal Packing Problem With Unloading Constraints

In this section we formally describe the Orthogonal Packing Problem with Unloading Constraint (OPPUL). We present exact algorithms, some heuristics, and lower bounds for the two and three-dimensional version of this problem.

2.1 Problem Description

The orthogonal packing problem with unloading constraint can be defined as follows: It is given a D-dimensional container B of dimensions $(W^1, \ldots, W^D)$ with total volume $\mathcal{V}(B) = \prod_{d=1}^{D} W^d$, where $W^d \in \mathbb{Z}^+$, $1 \leqslant d \leqslant D$; n sets of D-dimensional items $(I_1, \ldots, I_n)$, let $I = \bigcup_{v=1}^{n} I_v$. Each item $i \in I_v$ has dimensions $(w_i^1, \ldots, w_i^D)$, where $w_i^d \in \mathbb{Z}^+$. The volume of an item i is denoted by $\mathcal{V}(i) = \prod_{d=1}^{D} w_i^d$ and the volume of a set of items I is denoted by $\mathcal{V}(I) = \sum_{i \in I} \mathcal{V}(i)$. The problem is to find a *packing* $\mathcal{P}_I$ of the items I in the bin B that respects the *unloading constraints* in the direction of the last dimension D.

More precisely, a packing $\mathcal{P}_I$ of items I in a container $B = (W^1, \ldots, W^D)$ that satisfy unloading constraints is a function $\mathcal{P}_I : I \rightarrow [0, W^1) \times \ldots \times [0, W^D)$ such that:

(i) The packing must be orthogonal, i.e. the edges of the items must be parallel to the respective container's edges.

(ii) The packing must be oriented, i. e., the items must be packed in the original orientation given in I.

(iii) Items of I must be packed within the container's boundaries. That is, if the position where the item is packed is given by $\mathcal{P}_I(i) = (x_i^1, \ldots, x_i^D)$, for each $i \in I$, then

$$0 \leqslant x_i^d \leqslant x_i^d + w_i^d \leqslant W^d, \text{ for } 1 \leqslant d \leqslant D. \tag{1}$$

(iv) Items must not overlap. That is, if the region occupied by the item i is given by $\mathcal{R}(i) = [x_i^1, x_i^1 + w_i^1) \times \ldots \times [x_i^D, x_i^D + w_i^D)$ then

$$\mathcal{R}(i) \cap \mathcal{R}(j) = \varnothing, \text{ for all pairs } i \neq j \in I. \tag{2}$$

(v) Items belonging to a set I_v are not blocked by any item belonging to a set I_u if $u > v$. That is, if item i must be unloaded before item j, j can not be packed in the region between i and the end of the container, in the unloading dimension D. More precisely consider the region that includes the item i and it's way to the exit of the container defined by $\mathcal{R}^e(i) = \bigtimes_{d=1}^{D-1} [x_i^d, x_i^d + w_i^d) \times [x_i^D, W^D)$. A packing $\mathcal{P}_I$ of $I = I_1 \cup \ldots \cup I_n$ in the bin B respects the unloading constraints if it satisfy

$$\mathcal{R}^e(i) \cap \mathcal{R}(j) = \varnothing \quad \text{for all } i \in I_v \text{ and } j \in I_u \text{ with } 1 \leqslant v < u \leqslant n, \tag{3}$$

2.1.1 Definitions

An important concept used in several algorithms for packing problems is the *Envelope* of a packing or a partial packing. Let I be a set of items, each item $i \in I$ with dimensions $(w_i^1, \ldots, w_i^D)$, and $\mathcal{P}_I$ a packing of these items in a bin of dimensions $(W^1, \ldots, W^D)$. An envelope of $\mathcal{P}_I$ is the region defined by

$$S(\mathcal{P}_I) = \{(x^1, \ldots, x^D) \in \mathbb{R}_+^D : \exists i \in I \text{ with } x^1 < x_i^1 + w_i^1 \wedge \ldots \wedge x^D < x_i^D + w_i^D\}.$$

The complement of the envelope $\bar{S}(\mathcal{P}_I)$ is the region of the Bin B that is not in the envelope. The complement of the envelope is often considered as a feasible region to pack new items in different algorithms. The volume of the envelope $S(\mathcal{P}_I)$ and its complement $\bar{S}(\mathcal{P}_I)$ is denoted respectively by $\mathcal{V}(S(\mathcal{P}_I))$ and $\mathcal{V}(\bar{S}(\mathcal{P}_I))$. Figure 6 show an example of a two-dimensional packing and it's envelope.

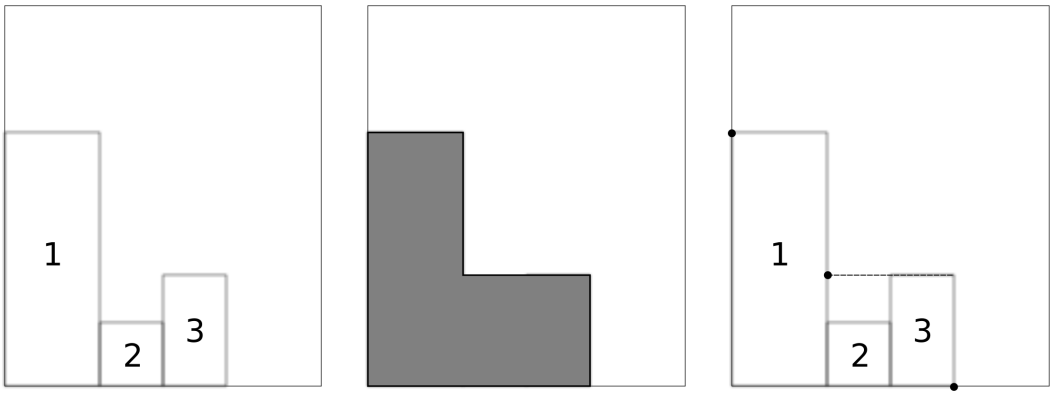
Another important concept is the set of *Corner Points* $\hat{C}(\mathcal{P}_I)$, which is defined by,

$$\hat{C}(\mathcal{P}_I) = \{(x^1, \ldots, x^D) \in \bar{S}(\mathcal{P}_I) : \nexists (x'^1, \ldots, x'^D) \in \bar{S}(\mathcal{P}_I) \backslash \{(x^1, \ldots, x^D)\}, x'^1 \leqslant x^1 \wedge \ldots \wedge x'^D \leqslant x^D\}.$$

Figure 1c presents the set of corner points for the packing, the black dots are the corner points.

2.2 Two-dimensional Orthogonal Packing Problem with Unloading Constraints

To solve the Two-dimensional Orthogonal Packing Problem with Unloading Constraints (2OPPUL), we consider two different exact algorithms found in the literature, which we modified to include unloading



(a) Packing (b) Envelope (c) Corner Points

Figure 1: (a) Example of a two-dimensional packing, (b) it's envelope and (c) the corner points for this packing.

constraints. The first is the OneBin algorithm presented by Martello et al. [20]. The second is a constraint programming algorithm presented by Clautiax et al. [3]. Both algorithms were modified to consider the customer's order of visit and the associated unloading constraints when generating a feasible packing.

To make the notation clear, when considering the Two-dimensional Orthogonal Packing Problems the Bin has dimensions (W, H) and each item i has dimensions (w_i, h_i) and it is packed in the position (x_i, y_i) in a feasible solution.

2.2.1 Branch-and-Bound algorithm

The algorithm OneBin works as follows: let I be the set off all items, J be a set of packed items and $\mathcal{P}_J$ a packing of these items in J. Let $\hat{C}(\mathcal{P}_J)$ be the set of corner points defined by $\mathcal{P}_J$ and let $\bar{J} = I \backslash J$ be the set of items not packed yet. At each iteration, $\hat{C}(\mathcal{P}_J)$ and $\mathcal{V}(S(\mathcal{P}_J))$ are computed. For each item $j \in \bar{J}$ and for each corner point $c \in \hat{C}(\mathcal{P}_J)$, j is assigned at c and OneBin is called recursively. Whenever $\mathcal{V}(B) - \mathcal{V}(S(\mathcal{P}_J)) \leqslant \mathcal{V}(\bar{J})$ happens, backtracking occurs, because the remaining items cannot be packed. Empty space inside the envelope is not used by this algorithm. Our algorithm is an adaptation to the OneBin source code provided by Martello et al. [20].

To deal with the unloading constraint, we modified the algorithm as follows: when an item is assigned to a corner point, the algorithm verifies if the item blocks any item that will be removed first. In this case, the algorithm backtracks. We also modified the initial ordering of items, ordering first by its customer's visiting order and then by nondecreasing volume.

2.2.2 CP model

The second exact algorithm was presented by Clautiax et al. [3], based on the constraint programming paradigm, to which we refer from now on as CP2D. In the CP2D algorithm, variables X_i and Y_i are

associated to the bottom left corner of each item i. The domain of each variable is defined by $X_i \in \{0, \ldots, W - w_i\}$ and $Y_i \in \{0, \ldots, H - h_i\}$. For each pair (i, j) of items the following constraints must be satisfied: $([X_i + w_i \leqslant X_j]$ or $[X_j + w_j \leqslant X_i]$ or $[Y_i + h_i \leqslant Y_j]$ or $[Y_j + h_j \leqslant Y_i])$. These constraints guarantee that two items do not overlap. In order to consider the unloading constraints, we adapted the algorithm as follow: if i will be unloaded before item j, we replace the previous constraints by $([X_i + w_i \leqslant X_j]$ or $[X_j + w_j \leqslant X_i]$ or $[Y_j + h_j \leqslant Y_i])$. With the aim of improving this constraint programming model, Clautiax et al. [3] proposed a redundant constraint programming model for a non-preemptive cumulative-scheduling problem associated with two relaxations of 2OPPUL, which are linked to the original problem. A set of activities $\{A_1^w, \ldots, A_{|I|}^w\}$, and a resource R_w are defined. Each activity A_i^w requires h_i of resource R_w, and has processing time w_i. The resource R_w has maximum capacity H. All activities must be executed by time W, and at any given time all activities being executed must not exceed the resource's maximum capacity. Analogously, we define the set $\{A_1^h, \ldots, A_{|I|}^h\}$ and resource R_h with maximum capacity W. Each activity A_i^h requires w_i of resource R_h, and has processing time h_i and all activities must be executed by time H.

To link these scheduling problems with the packing problem, the start time of an activity A_i^w must be equal to X_i and the start time of an activity A_i^h must be equal to Y_i. Using this model Clautiax et al. [3] proposed a series of pruning and propagation methods.

Reducing the Domain of Variables

In the formulation above, for each variable X_i, Clautiax et al. [3] considered all integer values between 0 and $W - w_I$ and for each variable Y_i the integer values between 0 and $H - h_i$. This can be improved by reducing the domain of each variable, reducing the number of choices where each item can be placed. For each dimension, we must find the *discretization points*, a reduced set of coordinates where items can be placed without changing the feasibility of the instance. The set of discretization points in a certain dimension is defined by all possible combinations of item's sizes in that dimension. Following the notation used by Côté et al. [4], consider $I_i^{\geqslant}$ the set of items to be unloaded with, or after i. Let P_i^H be a set of coordinates over the Y-axis that item i can assume, defined as

$$P_i^H = \left\{ y = \sum_{j \in I_i^{\geqslant} \setminus \{i\}} h_j \xi_j : 0 \leqslant y \leqslant H - h_i, \xi_j \in \{0,1\}, j \in I_i^{\geqslant} \setminus \{i\} \right\}. \tag{4}$$

Let P_i^W be a set of coordinates over the X-axis that item i can assume, defined as

$$P_i^W = \left\{ x = \sum_{j \in I \setminus \{i\}} w_j \xi_j : 0 \leqslant x \leqslant W - w_i, \xi_j \in \{0,1\}, j \in I \setminus \{i\} \right\}, \tag{5}$$

This way, we define for each item $i \in I$:

$$Dom(X_i) = P_i^W, \tag{6}$$

$$Dom(Y_i) = P_i^H. \tag{7}$$

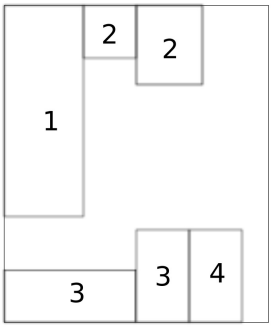


Figure 2: Example of packing using top-bottom mixfill.

Herz [14] shows that any feasible solution for a given instance of the problem, has a corresponding solution over the discretization points, this way we do not lose any solution by reducing the domain to the discretization points.

The top-bottom mixfill strategy

To take advantage of the unloading constraints we decided to apply the top-bottom mixfill strategy presented by Côté et al. [4] in the formulation of Clautiax et al. [3]. The unloading order can be explored to fill the bin not only from the bottom to the top, but from the top to bottom too. This can be done by dividing the items into those who will be unloaded first, and the others, to be unloaded later. Let c be the cut point between these two sets, that is, items in $I^T = I_1 \cup \ldots \cup I_c$ will be packed from the top to the bottom, and items in $I^B = I_{c+1} \cup \ldots \cup I_n$ will be packed from the bottom to the top. This way, we can redefine the coordinates in the Y-axis where an item can be placed. To this purpose consider $I_i^{T,\leqslant}$ the set of items from I^T that will be unloaded with or before i, and $I_i^{B,\geqslant}$ the set of items from I^B that will be unloaded with or after i. Let P_i^H be the new set of coordinates y where item i can be packed, if $i \in I^T$:

$$P_i^H = \left\{ y' = H - y : y = \sum_{j \in I_i^{T,\leqslant} \setminus \{i\}} h_j \xi_j, 0 \leqslant y \leqslant H - h_i, \xi_j \in \{0,1\}, j \in I_i^{T,\leqslant} \setminus \{i\} \right\}. \tag{8}$$

If $i \in I^B$:

$$P_i^H = \left\{ y = \sum_{j \in I_i^{B,\geqslant} \setminus \{i\}} h_j \xi_j : 0 \leqslant y \leqslant H - h_i, \xi_j \in \{0,1\}, j \in I_i^{B,\geqslant} \setminus \{i\} \right\}. \tag{9}$$

Figure 2 presents an example of packing using the top-bottom mixfill strategy. Côté et al. [4] proved that if a solution is feasible for a given cut point c, it is feasible for every cut point. This way we can choose the cut point that generates the least number of discretization points.

2.3 Heuristic and Hash

Besides the exact approach, a fast heuristic was employed to help decreasing the time spent solving the 2OPPUL. The exact algorithms are not called if a feasible packing can be found by a Bottom Left

Decreasing Width heuristic (BLDW), modified to consider the unloading constraints. For a description of the BLDW algorithm see [2].

To reduce the computational effort, known routes are stored in a hash table. A route is stored whether it is feasible or not, therefore the same route will not have its feasibility checked twice. Two routes will have the same hash, excluding collisions, if they serve the same customers and have an identical visiting order.

2.4 Metaheuristic for the Two-Dimensional Orthogonal Packing Problem With Unloading Constraints

With the aim to reduce the total processing time of our exact algorithm, we also present some heuristics to the 2OPPUL based on the Biased Random-Key Genetic Algorithm (BRKGA). We first give an overview of the BRKGA, and then we show more details of some heuristics for the 2OPPUL problem under this approach.

2.4.1 The BRKGA

The BRKGA presented by [12] is a general search metaheuristic for finding solutions to combinatorial optimization problems. This algorithm uses a chromosome of fixed size m of random keys over the interval $[0, 1)$, where the value of m depends on the instance of the optimization problem. An evolutionary process involves crossing-over different chromosomes and exchanges among different populations. The BRKGA also introduces new chromosomes called mutants to add variety.

The BRKGA involves the following main parameters:

- m is the size of a chromosome;
- p is the number of individuals (chromosomes) in a population;
- p_e is the percentage of elite individuals in a population;
- p_m is the percentage of new mutants to be introduced in a population at each generation;
- ρ_e is the probability that a gene is inherited from the elite parent;
- K is the number of independent populations;
- $MaxGen$ is the number of generations evolved;
- $Exch$ is the number of generations before Exchange best individuals among populations;
- $NExch$ is the number of best individuals to be exchanged among populations.

The BRKGA initializes each population with p randomly generated chromosomes, each having m random keys. Then it evolves each population by $MaxGen$ generations. The evolution of a population is composed of the following steps:

1. Compute the fitness function for each chromosome. A chromosome when associated with its fitness is called an individual.

2. Producing the next generation includes:

   (a) The elite set of the previous generation,

   (b) p_m new randomly generated mutants.

   (c) Chromosomes produced by matching two individuals from previous generations, one from the elite set and another from the non-elite set. Each gene has a probability ρ_e to be copied from the elite parent.

To use this metaheuristic as a framework to an optimization problem we need the following steps: define the number of genes in a chromosome, define a decoder which maps a chromosome into a solution (feasible or not) to our specific problem, and define the fitness value of the chromosome, which will measure the quality of the solution. In the subsequent subsections we describe these steps for some heuristics developed.

2.4.2 Heuristic Bottom-Left

Chromosome Each item $i \in I$ has an associated gene g_i.

Decoder In this heuristic items are placed in the inverse sequence which they will be unloaded, that is, set I_u is packed before set I_v if $u > v$. Within each set I_v items are sorted according to the corresponding genes' values in the chromosome, that is, if $i \in I_v$ and $j \in I_v$, i is placed before j if $g_i < g_j$.

Let J be the set of all packed items, initially J is empty, and let be $\mathcal{P}_J$ the packing of those items in the Bin. Consider the set of corner points $\hat{C}(\mathcal{P}_J)$ as defined in section 2.1.1. We call an eligible *bottom-left corner point*, the point $p = (x_p, y_p) \in \hat{C}(\mathcal{P}_J)$, such that, y_p is minimum and $x_p + w_i \leqslant W$. Each item i in the order obtained from the chromosome is placed in the eligible *bottom-left corner point*. The order which items are placed guarantee that the unloading constraints are respected. We allow items to overflow the height H of the bin.

Fitness Value The fitness value of an individual is given by the height used by the packing in the heuristic. If at any time we have an individual with fitness less than or equal to H we have a feasible packing.

2.4.3 Heuristic Bottom-Left and Left-bottom

Chromosome Each item $i \in I$ has two associated genes g_i^1 and g_i^2.

Decoder In this heuristic items are placed in the order given by the genes, item i is placed before j if $g_i^1 < g_j^1$. Based on the heuristic proposed by Gonçalves et al. [13], items can be placed in two different positions. Let J be the set of all packed items, initially J is empty, and let $\mathcal{P}_J$ be the packing of those items in the Bin. An item i is placed in the eligible bottom left corner point $p = (x_p, y_p)$ or in the eligible *left-bottom corner point* denoted by $q = (x_q, y_q) \in \hat{C}(\mathcal{P}_J)$, such that x_q is minimum. Item i is placed in point p or q according to the gene g_i^2, if $g_i^2 < 0.5$ it is placed in p, otherwise, it is placed in point q. We allow items to overflow the height H of the bin.

Fitness Value The fitness value of an individual is given by the height used in the packing given in the heuristic, plus a penalty of value H for each unloading constraint violated, that is, for each pair of items, if the unloading constraint is violated one penalty is applied. At the end of the process if we have an individual with fitness less than or equal to H we have a feasible packing.

2.4.4 Heuristic Tetris

Chromosome Each item $i \in I$ has two associated genes g_i^1 and g_i^2.

Decoder In this heuristic items are placed in the inverse sequence which they will be unloaded, that is, set I_u is packed before set I_v if $u > v$. Within each set I_v items are sorted according to the genes g^1, that is, if $i \in I_v$ and $j \in I_v$, i is placed before j if $g_i^1 < g_j^1$.

The gene g_i^2 represents the position x where the item i will be packed, and y is the lowest possible value, respecting the overlapping constraints. That is, item i will have position x_i defined by:

$$x_i = \lfloor g_i^2(W - w_i + 1) \rfloor. \tag{10}$$

The order in which items are placed guarantee that the unloading constraints are respected.

Fitness Value The fitness value of an individual is given by the height used in the packing given in the heuristic. At the end of the process if we have an individual with fitness less than or equal to H we have a feasible packing.

2.4.5 Heuristic Double-Layer Tetris

This heuristics has two stages. In the outer stage the order that each item will be placed is decided using the BRKGA, and then for a given sequence, in the inner stage the x positions are decided also using a BRKGA.

Chromosome In the outer stage each chromosome has size equal to n. Each item $i \in I$ has an associated gene g_i^1. We will call this a sequence chromosome.

Decoder Items are placed in the inverse sequence which they will be unloaded, that is, set I_u is packed before set I_v if $u > v$. Within each set I_v items are sorted according to the chromosome, that is, if $i \in I_v$ and $j \in I_v$, item i is placed before item j if $g_i^1 < g_j^1$.

For each sequence chromosome we search for the packing positions by executing another BRKGA. In this inner stage we call a chromosome by position chromosome, where each item has an associated gene g_i^2 representing the position x where the item i will be packed, that is, item i will have position x_i defined by:

$$x_i = \lfloor g_i^2(W - w_i + 1) \rfloor, \tag{11}$$

and y is the lowest possible value, respecting the overlapping constraints. The fitness value in this step is given by the height used by the packing obtained. The order which items are placed guarantee that the unloading constraints are respected.

Fitness Value The fitness value is the height used by the packing given by the best position chromosome found in the inner stage. At the end of the process if we have an individual with fitness less than or equal to H we have a feasible packing.

2.4.6 Heuristic Bottom-Left Penalty

Chromosome Each item $i \in I$ has an associated gene g_i.

Decoder The packing order of item i is given exclusively by the gene g_i, that is, if $g_i < g_j$, i will be packed before j independently of the unloading order. Items are packed in this order using the bottom left heuristic. This may lead to an unfeasible solution, so for each violation a penalty of value H is added to the fitness value.

Fitness Value The fitness value of an individual is given by the height used in the packing given by the heuristic, plus a penalty for each violated unloading constraint, that is, for each pair of items, if the unloading constraint is violated one penalty is applied. At the end of the process if we have an individual with fitness less than or equal to H we have a feasible packing.

2.4.7 Heuristic Absolute Position

Chromosome Each item $i \in I$ has two associated genes g_i^x and g_i^y.

Decoder In this heuristic an item i is placed, if possible, on the position (x_i, y_i) given by:

$$x_i = \lfloor g_i^x(W - w_i + 1) \rfloor, \tag{12}$$

$$y_i = \lfloor g_i^y(H - h_i + 1) \rfloor. \tag{13}$$

Items are packed on the reverse order which they will be unloaded. Items from a same set I_v are packed in the order given in the input. If item i violates the overlap constraint or the unloading constraint we do not pack the item i. At the end there might be items not packed. Let I be the set of all items, J be the set of packed items, and c the total number of conflicts.

Fitness Value The fitness value is given by

$$|I| - |J| + (c/n^2). \tag{14}$$

If an individual has fitness value 0, then all items are packed and the solution is feasible. The last term of the fitness value helps to compare two individuals with the same number of packed items. If one solution has less conflicts, then it is closer to a feasible solution.

2.4.8 Heuristic Best Corner Point

Chromosome Each item i has an associated gene g_i.

Decoder Each gene represents the order of the packing, that is, if $g_i < g_j$, i will be packed before j. For each item i the heuristic packs it in the corner point that generates the minimum amount of waste, and do not violate the unloading constraints. For each corner point s, it verifies if i can be placed on s without exceeding the limits of the Bin, and without violating the unloading constraints with the items already packed. If no violation occurs, s is a candidate point to i. For each candidate s, the heuristic computes the volume of the complement of the envelope with item i packed on s, and chooses the one with the biggest volume. An item may have no candidates, in this case it is not packed.

Fitness Value Let I be the set of all items, J be the set of packed items, and S the total volume of the envelope. The fitness value is given by

$$|I| - |J| + (W\,H - S)/(2WH). \tag{15}$$

If an individual has fitness value less than 1, them all items were packed and the solution is feasible. The last term of the fitness value helps to compare two individuals with the same number of packed items, preferring those who let more remaining space to be used.

2.5 Lower Bounds for the Orthogonal Packing Problem

Some lower bounds presented by Côté et al. [4], are used to obtain a minimum dimension size to pack a set of items in a Bin. If the lower bound is greater than the dimension size of the bin, then the packing is unfeasible.

2.5.1 Lower Bound L_1

The first Lower Bound, L_1, considers the minimum height required do pack all items.

$$L_1 = \left\lceil \frac{\sum_{i \in I} w_i h_i}{W} \right\rceil. \tag{16}$$

2.5.2 Lower Bound L_2

To present the second lower bound, L_2, we need the following definitions:

- $I_i^{>,W-w_i}$ is the set of all items to be delivered after i, with width greater than $W-w_i$, and $\sum_{(i)}^{>,W-w_i}$ is the total volume of these items.
- $\sum_{(i)}^{>}$ is the total volume of items to be delivered after i.
- $I_i^{\leqslant,w_i}$ is the set of items to be delivered before or with i and has width greater than w_i.
- $I_i^{<,W-w_i}$ is the set of all items to be delivered before i, with width greater than $W - w_i$, and $\sum_{(i)}^{<,W-w_i}$ is the total volume of these items.
- $\sum_{(i)}^{<}$ is the sum of volumes of items to be delivered before i.
- $I_i^{\geqslant,w_i}$ is the set of items to be delivered after or with i and has width greater than w_i.
- P_i^H is the set of discretization points where item i can be placed.

The bound L_2 also uses the values of $y_i^{\min}$ and $y_i^{\max}$, which are respectively the minimal position that can be occupied by the bottom of item i and the maximal position that can be occupied by the top of item i. To calculate $y_i^{\min}$ and $y_i^{\max}$ the following values must be computed:

$$y_i^{\min,1} = \max\left\{ \left\lceil \frac{\sum_{(i)}^{>,W-w_i}}{W} \right\rceil, \max\{h_j : j \in I_i^{>,W-w_i}\} \right\}, \tag{17}$$

$$y_i^{\min,2} = \left\lceil \frac{[\sum_{(i)}^{>} - H(W-w_i) + \sum_{j \in I_i^{\leqslant,w_i}} h_j(w_j - w_i)]^+}{w_i} \right\rceil, \tag{18}$$

$$y_i^{\max,1} = H - \max\left\{ \left\lceil \frac{\sum_{(i)}^{<,W-w_i}}{W} \right\rceil, \max\{h_j : j \in I_i^{<,W-w_i}\} \right\}, \tag{19}$$

$$y_i^{\max,2} = H - \left\lceil \frac{[\sum_{(i)}^{<} - H(W-w_i) + \sum_{j \in I_i^{\geqslant,w_i}} h_j(w_j - w_i)]^+}{w_i} \right\rceil. \tag{20}$$

$$\tag{21}$$

This way $y_i^{\min}$ and $y_i^{\max}$ are computed as follow:

$$y_i^{\min} = \max\{y_i^{\min,1}, y_i^{\min,2}, \min_{j \in I_i^{>,W-w_i}}\{y_j^{\min} + h_j\}, \min\{t | t \in P_i^H\}\}, \tag{22}$$

$$y_i^{\max} = \min\{y_i^{\max,1}, y_i^{\max,2}, \min_{j \in I_i^{<,W-w_i}}\{y_j^{\max} - h_j\}, \max\{t | t \in P_i^H\} + h_i\}. \tag{23}$$

Finally using definitions of $y_i^{\min}$ and $y_i^{\max}$, L_2 can be computed as follow:

$$L_2 = \max_{i \in I}\{y_i^{\min} + h_i + (H - y_i^{\max})\}. \tag{24}$$

2.5.3 Lower Bound L_3

Based on the Cutting Stock Problem we can obtain the lower bound L_3^H. A cutting pattern is defined as a subset $I' \subset I$, such that $\sum_{i \in I'} w_i \leqslant W$. Let $\mathcal{K}^W$ be the set of all feasible patterns, and $a_{ik} = 1$ if item i belongs to pattern k, and $a_{ik} = 0$ otherwise. The following Integer Linear Program is constructed, where v_k is the variable that represents the number of times pattern k appears in the solution.

$$\min \sum_{k \in \mathcal{K}^W} v_k \tag{25}$$

$$\sum_{k \in \mathcal{K}^W} a_{ik} v_k \geqslant h_i \qquad i \in I \tag{26}$$

$$v_k \geqslant 0, v_k \in \mathbb{Z} \quad k \in \mathcal{K}^W. \tag{27}$$

The Cutting Stock Problem is usually solved by column generation, solving a series of knapsack problems with reduced costs. The knapsack problem is efficiently solved by Dynamic Programming. The optimal solution of the Cutting Stock Problem (CSP) gives a lower bound on the minimum height of the Bin necessary to pack all items. Let L_3^H be the value of the optimal solution of CSP, if $L_3^H < H$ we know that the packing of items I into bin $B = (W, H)$ is unfeasible.

Analogously, the bound L_3^W can be defined by doing the same process over the transversal axis. The resulting Cutting Stock Problem will give a lower bound on the width necessary to pack the items.

2.6 Three-dimensional Orthogonal Packing Problem With Unloading Constraints

To solve the Three-dimensional Orthogonal Packing Problem With Unloading Constraints (3OPPUL), we used two different exact algorithms. The first is the OneBin_General algorithm presented by Martello et al. [21]. The algorithm was slightly modified to consider the customer's items and their unloading order when generating a feasible packing. The second is a natural CP model to the problem.

2.6.1 CP based algorithm - relative positions

The first algorithm used to solve the 3OPPUL is the OneBin_General algorithm, introduced by Martello et al. [21]. This algorithm is based on constraint programming, and uses the following idea, two items (i, j) do not overlap if item i is *left of*, *right of*, *under*, *above*, *behind*, or *in front of* item j. To represent this relation between items i and j, it is created a variable r_{ij} for each pair of items, this variable can assume one of the following values $\{\texttt{l}, \texttt{r}, \texttt{u}, \texttt{a}, \texttt{b}, \texttt{f}\}$.

The problem is solved recursively by the algorithm OneBin_General. At each call two items i, j are considered and one of the relative positions are assigned to r_{ij}. The new assignment must be checked for

its feasibility with the previous assignments made.

In each call, after assignments were made in the variables r, the algorithm needs to find the positions (x, y) for the placement of each item. To this purpose it initializes the position (x_i, y_i) of each item i with $x_i = 0$ and $y_i = 0$. For each relation already assigned, one of the following assignments is performed: if $r_{ij} = \texttt{l}$ and $x_j < x_i + w_i$, the item j is "pushed" left by doing $x_j = x_i + w_i$; otherwise if $r_{ij} = \texttt{r}$ and $x_i < x_j + w_j$, item i is "pushed" left by doing $x_i = x_j + w_j$. A similar assignment is made for the other possible relations. The algorithm repeats this procedure until no modifications are made. If at any point an item is positioned such that it exceeds the size of the bin, the last assignment has been proved to be unfeasible.

We have adapted the original algorithm to consider the unloading constraints. For a certain variable r_{ij}, we remove from its domain the value `f`(*front*) if i must be unloaded after j, or remove `b`(*behind*) if i must be unloaded before j.

2.6.2 A natural CP model - absolute positions

We now describe a natural constraint programming formulation for the Three-dimensional Loading problem. Similarly to the CP2D, variables X_i, Y_i and Z_i are defined for each item $i \in B$, where (X_i, Y_i, Z_i) is the position where item i will be packed. The initial domains are defined as $X_i \in \{0, \ldots, W - w_i\}, Y_i \in \{0, \ldots, H - h_i\}$ and $Z_i \in \{0, \ldots, D - d_i\}$.

For each pair of items i, j, we include a constraint $[X_i + w_i \leqslant X_j]$ or $[X_j + w_j \leqslant X_i]$ or $[Y_i + h_i \leqslant Y_j]$ or $[Y_j + h_j \leqslant Y_i]$ or $[Z_i + d_i \leqslant Z_j]$ or $[Z_j + d_j \leqslant Z_i]$. To attend the unloading constraint, if i must be unloaded before j, we replace this constraint by $[X_i + w_i \leqslant X_j]$ or $[X_j + w_j \leqslant X_i]$ or $[Y_i + h_i \leqslant Y_j]$ or $[Y_j + h_j \leqslant Y_i]$ or $[Z_j + d_j \leqslant Z_i]$. We call this algorithm by CP3D.

3 Capacitated Vehicle Routing Problem with Loading Constraints

In this section we formally describe the Capacitated Vehicle Routing Problem with Loading Constraints (DL-CVRP). We first formally describe the problem, then we present the formulation been considered.

3.1 Problem Description

The Capacitated Vehicle Routing Problem with Loading Constraints (DL-CVRP) can be defined as follows: It is given a complete undirected graph $G = (V, E)$, such that, V is a set of $n + 1$ vertices, where vertex 0 corresponds to the depot, and vertices $1, \ldots, n$ that corresponds to the n customers, and E is the set of edges. For each edge $e \in E$ there is an associated cost $c_e \in \mathbb{Q}^+$. It is also given a set of K identical vehicles, each having a weight capacity of M and a container of dimensions $W^1, \ldots, W^D$ to carry the customer's items, with D been the dimension of the problem. Let $V_+ = V \setminus \{0\}$ be the set of customers. Each customer $v \in V_+$ has a demand of a set I_v of items with total weight m_v. Each item i in $I = \bigcup_{v=1}^{n} I_v$ has dimensions $(w_i^1, \ldots, w_i^D)$.

We need to find routes for each vehicle, and each route must have a feasible Packing that respects the unloading constraints. A feasible route C is a cycle in G that contains the depot and satisfies the following conditions:

(i) The total weight of all items of customers in C must not exceed the vehicle load capacity. That is, if V_C is the set of customers in C then $\sum_{v \in V_C} m_v \leqslant M$.

(ii) There is a packing $\mathcal{P}_C$ for the items of the clients in C in one of the vehicles. The packing $\mathcal{P}_C$ must respect the unloading constraints.

The DL-CVRP is to find a set of K feasible routes $\mathcal{C} = \{C_1, \ldots, C_K\}$, that minimizes the total routing cost, that is, minimize $c(\mathcal{C}) = \sum_{i=1}^{k} \sum_{e \in C_i} c_e$.

3.2 Formulation

In this section we present the integer linear programming formulation used to model and solve the DL-CVRP.

Let q_e be a variable that indicates the use of an edge e in the solution, that is, q_e is equal to one if a vehicle travels along the edge e, and zero otherwise. Given a subset of customers $S \subseteq V_+$, $m(S)$ is the total weight of all items of the customers in S, i.e., $m(S) = \sum_{i \in S} m_i$. We denote by $\mathcal{V}(B)$ the volume of the container and $\mathcal{V}(S)$ the total volume of the items of the customers in S, i.e, $\mathcal{V}(S) = \sum_{v \in S} \mathcal{V}(I_v)$. Let $\delta(S)$ be the set of edges in G with exactly one vertex in S. Let $r(S)$ be the minimum number of vehicles needed to supply the demand of S and K be the number of available vehicles, considering the weight of the demands of the items. Also let $\mathcal{R}$ be the set of routes that are infeasible to be packed respecting the unloading constrains. The integer programming formulation is:

$$\text{minimize} \sum_{e \in E} c_e q_e \tag{28}$$

$$\text{s.a.} \sum_{e \in \delta(\{i\})} q_e = 2 \qquad \forall i \in V_+ \tag{29}$$

$$\sum_{e \in \delta(S)} q_e \geqslant 2r(S) \qquad \forall S \subseteq V_+, |S| \geqslant 2 \tag{30}$$

$$\sum_{e \in \delta(\{0\})} q_e = 2K \tag{31}$$

$$\sum_{e \in R} q_e \leqslant |R| - 1 \qquad \forall R \in \mathcal{R} \tag{32}$$

$$q_e \in \{0, 1\} \qquad \forall e \in E. \tag{33}$$

Constraints (29) ensure that each customer is visited exactly once. Constraints (30), the Capacity Inequalities, impose connectivity and capacity conditions. If $r(S)$ is replaced by $k(S) = max\{\lceil m(S)/M \rceil, \lceil \mathcal{V}(S)/\mathcal{V}(B) \rceil\}$, a valid lower bound is achieved. These are known as Rounded Capacity Inequalities (see [24]).

Constraints (31) ensure that exactly K vehicles are used. Constraints (32) ensures that there exists a packing respecting unloading constraints for each route. Note that to compute if a route R belongs do $\mathcal{R}$ is an NP-Hard problem, since it is equivalent to finding the optimal solution of the Orthogonal Packing Problem with Unloading Constraints (OPPUL), given the vehicle container and the customers items. Constraints (33) impose that the variables must be binary. Following [17], we do not allow routes with only one customer.

4 Branch-and-Cut Algorithm for the DL-CVRP

The DL-CVRP consists of a routing and a packing problem, therefore routing and packing strategies are needed. Following Azevedo et al. [1] our routing separation routine consists of separating Capacity Inequalities to ensure connectivity and capacity constraints, and using other families of inequalities to try reducing the feasible region. The packing is used when no more routing cuts can be found at the current node, but we can extract some information from the packing problem.

4.1 Routing Separation Routine

The formulation has an exponential number of constraints. The algorithm then starts with a small set of constraints and add others by separation routines. Constraints (29), (31) and the bounds give us a relaxation that can be easily solved, generating a weak lower bound that can be strenghtened by adding cutting planes. For this reason, the following separation routines were used: Capacity Inequalities, Framed Capacity Inequalities, Multistar Inequalities, 2-Edges Extended Hypotour Inequalities and Strenghtened Comb Inequalities. Observe that separation for Capacity Inequalities take into account not only the capacity of the vehicle, but also its container's area. Details on the used cuts can be found in [19] and [24].

Separating Capacity Inequalities is NP-Hard (see [24]), consequently separation is done on Rounded Capacity Inequalities, obtained by replacing $r(S)$ by $k(S)$ on (30). To separate different CVRP families of inequalities we used code provided by [19].

Following Azevedo et al. [1], to allow better chances of finding more cutting planes at each iteration at the root node, two cyclical approaches based on a separation strategy proposed by [19] are used. At each iteration, the following separation routines can be called: Rounded Capacity Inequalities, Framed Capacity Inequalities, Multistar Inequalities, Strenghtened Comb Inequalities and 2-Edges Extended Hypotour Inequalities. For each separation attempt, every inequality that was found is added to the LP.

Initially, the algorithm tries to separate Rounded Capacity Inequalities. If there is at least one cut violated by less than a certain limit, the algorithm attempts the Framed Capacity separation. If no Framed Capacity Inequality is found, two different approaches are used: if the current iteration is a multiple of five, the algorithm tries to separate Multistar and Strenghtened Comb Inequalities. Otherwise, a circular strategy is put into action: separation of Multistar, Strenghtened Comb and Hypotour occurs every third iteration, e.g., if Multistar Inequalities are found, re-optimization occurs and the algorithm will not try to

separate this family again until both Strenghtened Comb and Hypotour Inequalities are found. The order used was Multistar, followed by Strenghtened Comb and then Hypotour Inequalities. It should be noted that separation is pursued only if the algorithm finds at least one Rounded Capacity Inequality violated by a predefined limit. This limit is different for each family.

Non-root nodes of the branching tree are treated differently than the root node. First the algorithm calls the routine to separate Capacity Inequalities, then we try to separate Framed Capacity Inequalities, followed by Multistar Inequalities and Strenghtened Comb Inequalities. Unlike our approach for the root node, these procedures are called without any conditions (limits). For each subsequent iteration, only Capacity separation is attempted.

4.2 Packing Separation Routine

When none of the Routing Separations mentioned are found, we can use the information given by the packing problem to add some extra cuts. Those cuts will also eliminate unfeasible routes, the ones with clients such that theis items cannot be packed in a single vehicle. Given a partial solution for the integer linear program (28-33) described in section 3.2, it is possible that not all variables are integer. Then we propose two methods that can be used.

In the first method, the Branch-and-Bound is done until all values are integers, so the solution is a set of routes. To each of these routes, the algorithm searches for a feasible packing. Figure 3 illustrates a partial solution where all variables are integers. We will call this method *BranchFirst*.

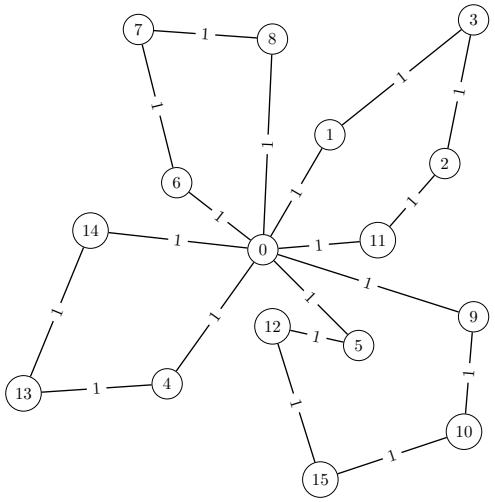


Figure 3: Example of a partial solution, packing can be tested for each route.

The second strategy searches for possible cuts even with fractional values, as illustrated in Figure 4. We define the problem of finding cuts in this partial solution as *Unpackable Path Problem*.

4.2.1 Unpackable Path Problem

The Unpackable Path Problem is the problem of given a fractional partial solution, find a path whose customer's items cannot be packed in that specific order. This path can be removed from the solution, by adding a new cut.

Heuristic for Unpackable Path Problem

Initially all edges e whose value of x_e is below a certain value $p > 0.5$ are removed. In the example of figure 4 we considered $p = 0.7$, and removed the dashed edges. The edges incident to the depot are also removed. We obtain a set of disjoint paths and isolated vertices, these paths are called sub-routes. Let P be one of these paths, where $\sum_{e \in P} x_e > m - 2$. The algorithm then check if the packing needed to the path P is feasible. If it is not, we add the following cut $\sum_{e \in P} x_e \leqslant |P| - 1$, pruning this sub-route from the set of solutions. We will call this strategy *CutFirst*.

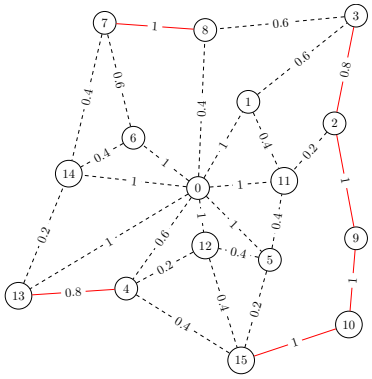


Figure 4: Example of a partial solution with fractional variables, packing can also be used to find cuts in this case.

For each route or sub-route of a solution or partial solution, we perform the following steps: first, we check if it can be found in the hash table, if so, it means that this route's packing was solved previously and its feasibility is known. Otherwise, we try our packing heuristic. The heuristic attempts to pack the route items in both customer sequences of the route. If a feasible packing is found, we update the hash table. If it fails, one of the exact algorithms is called and the hash table is updated with the results found. Thus we avoid multiple calls to identical packing instances. If there is no feasible packing for the corresponding route a cut equivalent to the route is added to remove this route.

One can also conclude the unfeasibility of routes or sub-routes by looking for unpackable subsequences. For instance, consider we wish to know if the route (3, 2, 9, 10, 15) is feasible or not, and we already have concluded and stored that (3, 9, 10) is unfeasible, we can conclude that the route (3, 2, 9, 10, 15) is also unfeasible. Obviously, the search by this subsection may take some time and do not always compensate. Therefore we tested with and without this search. Combined with strategy *CutFirst*, we call this search by *SubSeqSearch*.

5 Computational Results

The proposed algorithm was implemented in c++ language, and compiled with g++ 4.6.4 in a computer running Linux. The experiments were run on a 2.93GHz Intel Xeon Cpu. With the exception of the metaheuristic the algorithm only runs in a single core. The integer linear programing solver used was the Cplex 12.5.1, and the constraint programing solver was the CP Solver 1.7, both from IBM ILOG.

Following Lysgaard et al.[19] and Azevedo et al. [1] the root node separation strategy tries to separate different families of cuts only if there is at least one Rounded Capacity Inequality violated by less than a certain limit. These limits are: 0.2 for Framed Capacity Inequalities, 0.05 for Multistar Inequalities, 0.1 for Strenghtened Comb Inequalities and 0.1 for the 2-Edges Extended Hypotour Inequalities.

5.1 Instances

We tested our algorithm for the 2L-CVRP on the 60 instances used by Iori et al. [17], from the 180 available at `http://www.or.deis.unibo.it/`. These instances were based on the CVRP instances from literature, provided by Reinelt [25]. For the 12 CVRP instances considered, Iori et al. [17] take the complete graph connecting the clients, the weights m_v for each client v and the total capacity M of each vehicle. The distances from each pair of clients is a integer obtained by truncating the euclidian distances.

For each one of the 12 CVRP instances, 5 classes were created by Iori, corresponding to different ways to generate the items for each client. First class is the original instance, creating a single item for each client with both sizes equals 1, and $W = H = n$. For the remaining classes the vehicles were considered with sizes $W = 20$ and $H = 40$. and the number of items per client is a uniformly random value in the interval $[1, r]$, where r is the number of the class. For each item it's shape is selected with equal probability among: *Vertical* , *Homogeneous* and *Horizontal*, and it's values are randomly generated in the intervals given in Table 1.

The number of vehicle available in each instance were obtained by Iori, heuristically solving a Two-dimensional Bin Packing Problem considering all items, but not considering Unloading Constraints or gathering items of a same customer in the same container. Then K is set to the maximum value between this number and the number of vehicles in the original CVRP instance. These could lead to a infeasible solution, as the number of vehicles available could be less than the number of vehicles required, but the authors have reported that this do not occurs.

To test the efficiency of the modifications in CP2D we use the instances provided by Côté et al. [4]. These instances were obtained first creating extra instances based on the 2L-CVRP instances available at `http://www.or.deis.unibo.it/`, by simply modifying the dimensions of the container according with 5 different types:

- Type 1 : H = 40, W = 20

Table 1: Classes for the 2L-CVRP instances

Class	Itens per Client	*Vertical*		*Homogeneous*		*Horizontal*	
		h	w	h	w	h	w
1	1	1	1	1	1	1	1
2	$[1,2]$	$[\frac{4H}{10}, \frac{9H}{10}]$	$[\frac{W}{10}, \frac{2W}{10}]$	$[\frac{2H}{10}, \frac{5H}{10}]$	$[\frac{2W}{10}, \frac{5W}{10}]$	$[\frac{H}{10}, \frac{2H}{10}]$	$[\frac{4W}{10}, \frac{9W}{10}]$
3	$[1,3]$	$[\frac{3H}{10}, \frac{8H}{10}]$	$[\frac{W}{10}, \frac{2W}{10}]$	$[\frac{2H}{10}, \frac{4H}{10}]$	$[\frac{2W}{10}, \frac{4W}{10}]$	$[\frac{H}{10}, \frac{2H}{10}]$	$[\frac{3W}{10}, \frac{8W}{10}]$
4	$[1,4]$	$[\frac{2H}{10}, \frac{7H}{10}]$	$[\frac{W}{10}, \frac{2W}{10}]$	$[\frac{H}{10}, \frac{4H}{10}]$	$[\frac{W}{10}, \frac{4W}{10}]$	$[\frac{H}{10}, \frac{2H}{10}]$	$[\frac{2W}{10}, \frac{7W}{10}]$
5	$[1,5]$	$[\frac{H}{10}, \frac{6H}{10}]$	$[\frac{W}{10}, \frac{2W}{10}]$	$[\frac{H}{10}, \frac{3H}{10}]$	$[\frac{W}{10}, \frac{3W}{10}]$	$[\frac{H}{10}, \frac{2H}{10}]$	$[\frac{W}{10}, \frac{6W}{10}]$

- Type 2 : H = 32, W = 25
- Type 3 : H = 50, W = 16
- Type 4 : H = 80, W = 14
- Type 5 : H = 130, W = 14

They end up with 900 2L-CVRP instances, in which they heuristically generate routes of clients saving those that could not been proved unfeasible by some Lower Bounds. They end up with a total 2183 2OPPUL instances. Which we use to test the efficiency of our packing algorithms.

The heuristics presented in section 2.4 aim to solve some instances were the exact algorithm would take to much time. As the instances provided by Côté et al. [4] does not reflect this kind of instance, we create a new set of instances for the two-dimensional orthogonal packing problem. These instances were generated by running our algorithm for the 2L-CVRP, and when the algorithm faces a 2OPPUL that cannot be solved by CP2D within 1 second, we save this route as an new 2OPPUL instance. We let this algorithm runs for 10 seconds. We end up with a total of 296 instances. These new instances are available at the Laboratory of Optimization and Combinatorics website `http://www.loco.ic.unicamp.br`.

5.2 Efficiency of CP and Discretization Points

The use of the discretization points and the top-bottom mixfill approach were tested in the instances of [4], and the time limit was set to 600 seconds, we compare the efficiency of CP with the Branch-and-Bound algorithm OneBin adapted to consider unloading constraints. Column S_{OB} presents the number of instances solved by the adapted OneBin algorithm, column T_{OB} presentes the average time used in the instances that were solved. The column S and T indicates respectively the number of instances solved, and the average time used by the CP2D with complete domain. Analogously columns S_{dp} and T_{dp} indicates numbers of instances solved, ant average time by CP2D with reduced domain, considering

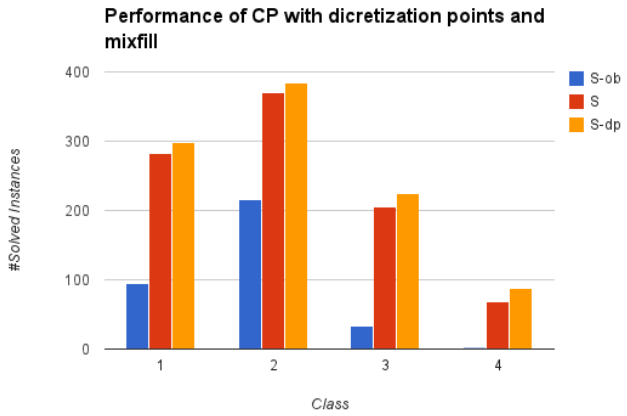


Figure 5: Performance of CP with dicretization points and mixfill.

only discretization points as presented in section 2.2.2. Finally columns S_{dp_mix} and T_{dp_mix} indicates respectively the number of instances solved, and the average time used by CP2D using the discretization points and the top-bottom mixfill presented in section 2.2.2. Those combinations of Class and type where all the algorithms were able not able to solve any instance were ommited.

Table 2: Performance of CP with dicretization points and mixfill

Class	Type	#inst	S_{OB}	T_{OB}	S	T	S_{dp}	T_{dp}	S_{dp_mix}	T_{dp_mix}
1	3	30	27	80.37	29	9.12	30	10.39	30	8.56
1	4	198	64	185.66	185	33.81	190	28.58	190	25.85
1	5	200	3	384.01	68	153.37	79	147.92	81	151.66
2	2	1	1	0.21	1	0.06	1	0.04	1	0.00
2	3	128	122	62.47	128	18.43	128	9.88	128	8.42
2	4	198	89	195.56	187	50.90	194	47.43	194	43.37
2	5	200	5	405.06	55	162.25	61	152.17	63	159.46
3	3	2	2	32.35	2	0.36	2	0.22	2	0.19
3	4	155	30	194.35	144	30.89	148	22.45	148	22.22
3	5	206	2	345.03	59	147.54	75	124.07	75	116.87
4	4	187	2	218.11	68	152.84	85	118.86	92	114.50
4	5	182	0	0.00	1	136.231	3	332.42	4	363.55

The use of Discretization Points increased the number of solved instances in more than 8% for some types, for example, Class 4 Type 4 and the use of the top-bottom mixfill increased up to 12% for the same type, if compared to the CP2D with complete domain. And in none of the types the number of solved instances has decreased when using these strategies.

5.3 Efficiency of metaheuristics

The proposed heuristics to solve the two-dimensional orthogonal packing problem, were implemented using the brkgaAPI framework, that implements the BRKGA described in section 2.4.1. We use the brkgaAPI provided at http://www2.research.att.com/ mgcr/src/brkgaAPI/. The objective of these heuristics are to improve the time spent to solve each route, found when solving the original set of instances to the 2L-CVRP, where the exact algorithm may take too long to compute the feasibility. Unfortunately, the set of instances provided by Côté et al. [4] does not reflect the routes that we desire do solve with these heuristics when solving the 2L-CVRP.

To evaluate the possible performance of these heuristics, we generate a new set of instances as follow: We execute the branch-and-cut algorithm for 10 seconds for all the original 2L-CVRP instances provided by Iori et al. [17], and for each route found we first check its feasibility with the lower-bounds. Than if lower bounds does not prove its unfeasibility, we run the basic BLDW presented in section 2.3, and if it doesn't found a packing, we run the CP2D algorithm with 1 second limit. If feasibility or unfeasibility has not been proved by these methods we store this route in our set of instances. The result is a set of 296 instances to the orthogonal packing problem.

We evaluate the performance of the heuristics based on BRKGA with this new set of instances. Table 3 presents the performance of the Heuristic Absolute Position presented in section 2.4.7, and Heuristic Better Corner Point presented in section 2.4.8. The column "Type" indicates the type of the instance, column "#inst" indicates the number of instances of that type and column "avg. it." the average number of items in these instances. Columns S_{cp} and T_{cp} represents respectively the number of feasible instances of that type, and the average time spent by algorithm CP2D to solve the instances. Columns S_{abs} and T_{abs} represents respectively the number of feasible solutions found by Heuristic Absolute Position and the average time that the best solution was founded. Analogously, columns S_{bcp} and T_{bcp} represents respectively the number of feasible solutions found by Heuristic Better Corner Point and the average time that the best solution was founded.

Table 3: Performance of metaheuristics to orthogonal packing problem

Type	#inst	avg. it.	S_{cp}	T_{cp}	S_{abs}	T_{abs}	S_{bcp}	T_{bcp}
1	0							
2	0							
3	22	13.05	1	2.32	0	0.13	0	0.02
4	67	14.94	9	31.11	0	0.21	2	0.03
5	207	19.84	112	62.19	0	0.39	36	0.09
All	296	18.23	122	50.7	0	0.33	38	0.07

Table 4 presents the performance of the remaining heuristics based on BRKGA introduced in section 2.4. For each heuristic h three columns S_h, T_h and H_h indicates respectively, the number of feasible solutions found by heuristic h, the average time that the best solution was founded, and the average best

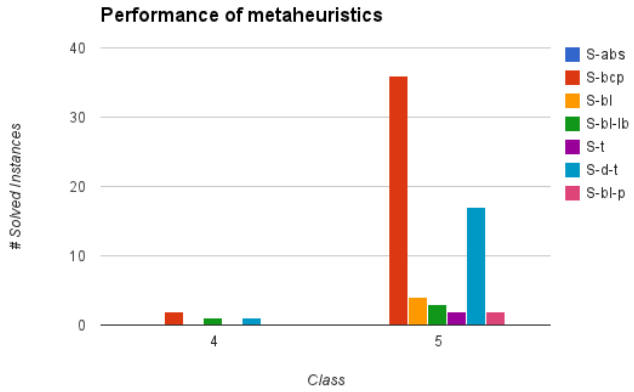


Figure 6: Performance of metaheuristics.

fitness function achieved by heuristic h. The heuristics are

- bl: Heuristic Bottom-Left presented in section 2.4.2
- $bl - lb$: Heuristic Bottom-Left and Left-bottom presented in section 2.4.3
- t: Heuristic Tetris presented in section 2.4.4
- $d.t$: Heuristic Double-Layer Tetris presented in section 2.4.5
- $bl - p$: Heuristic Bottom-Left Penalty presented in section 2.4.6

Table 4: Performance of metaheuristics to strip packing problem

Type	S_{bl}	T_{bl}	H_{bl}	S_{bl-lb}	T_{bl-lb}	H_{bl-lb}	S_t	T_t	H_t	$S_{d.t}$	$T_{d.t}$	$H_{d.t}$	S_{bl-p}	T_{bl-p}	H_{bl-p}
1															
2															
3	0	0	53.82	0	0.06	47.05	0	0.16	44.36	0	0.17	43.68	0	0.03	50.18
4	0	0	51.84	1	0.09	47.07	0	0.15	45.03	1	0.22	44.19	0	0.04	50.49
5	4	0.02	49.46	3	0.21	46.79	2	0.27	44.51	17	0.47	43.9	2	0.1	49.67
All	4	0.01	50.32	4	0.17	46.88	2	0.23	44.61	18	0.39	43.95	2	0.08	49.9

Heuristics Better Corner Point and Double-Layer Tetris presented in section has showed the best results, for this reason we choose to include only these two in the branch-and-cut algorithm, besides the basic BLDW heuristic.

5.4 Comparation between different algorithms

We compare our algorithm with the exact approach presented by Iori et al. [17]. We transcribed the results presented in their paper. We run our code in a single core of a Intel Xeon 2.93GHz, while Iori et al. used a Intel Pentium 3Ghz.

The table 5 presents the instance Name, Class, number of clients (n), total number of items (M) and number of available vehicles (K). For the results of Iori et al. table 4 shows the total number of

Cuts, the gap, the value of obtained solution (z) and The total time spent by its algorithm (T_{tot}. The last 7 columns present the results of our algorithm, #Heu is the number of routes that have been proved feasible by heuristics or by the CP2D limited by 1 second, #Ex is the number of routes that have been proved feasible by CP2D and #ExUn if the number of unpackable routes proved by the exact algorithm. The remaining columns presents the total number of cuts based on the packing problem ($Cuts_{pack}$), and the total number of cuts based on the routing problem ($Cuts_{rout}$), the total time spent (T_{BNC}) and the gap.

As we limit our algorithm to 3600 seconds to make the results more compatibles we remove those lines in which both algorithm take more than an hour to return a solution, remaining 51 instances. Our approach generates 160% more cuts than the previous approach. Both algorithms could not solve only one of the remaining 51 instances, while Iori could not solve the instance E030-03g class 1, our approach did not solve E030-03g class 3. Leaving out these instances our approach has been showed to be in average 200% faster than the previous Branch-and-Cut.

Obviously no packing cut has been added in the instances of class 1, since all combinations of clients is packable in this class. We also observe that in all instances of Class 1 (even excluding instance E030-03g) our algorithm is very effective (more than 600% faster). Which indicates that is a good algorithm even when only routing is considered.

The heuristics were very effective been able to find the feasible solutions of almost every route. Only 10 feasible routes in 11733 routes founded failed to be found by the heuristics. Noted that the CP2D limited by 1 second is considered as a Heuristic. While a large number of routes were unpackable, and discovered only by the exact algorithm, which indicates that a investment in lower bounds may be worthwhile.

6 Conclusions and Future work

In this paper we presented a branch-and-cut algorithm for the vehicle routing problem with loading constraints. Our algorithm has showed to be more robust and faster than the compared algorithm.

We also presented adapted exact algorithms to solve the Two and Three-Dimensional Orthogonal Packing Problem With Unloading Constraints. In particular we adapted the algorithm presented by Clautiax et al. [3], to consider the Unloading Constraints, and improved by 9% with the use of discretization points and the top-bottom mixfill approach.

We also presented several heuristics for the Two-Dimensional Orthogonal Packing Problem With Unloading Constraints, those heuristics were evaluated by a new set of instances. Heuristics Best Corner Point and Double-Layer Tetris were very effective and in fact solved few OPPUL instances where the exact algorithms were taking much more time.

Several improves yet can be made, the use of an initial primal heuristic, the use of betters algorithms (exacts and heuristics) for OPP, improved lower bounds. It is also possible to consider variants of this problem with new pratical constraints. As time windowns, stability.

Acknowledgements: This research was partially supported by CAPES, CNPq and FAPESP proc(2011/13382-3).

References

[1] B. L. P. Azevedo, P. Hokama, F. K. Miyazawa, and E. C. Xavier. A branch-and-cut approach for the vehicle routing problem with two-dimensional loading constraints. *XLI Simposio Brasileiro de Pesquisa Operacional*, 2009.

[2] B. Baker, E. Coffman, and R. Rivest. Orthogonal packings in two dimensions. *SIAM Journal on Computing*, 9(4):846–855, 1980.

[3] F. Clautiaux, A. Jouglet, J. Carlier, and A. Moukrim. A new constraint programming approach for the orthogonal packing problem. *Computers and Operations Research*, 35(3):944 – 959, 2008.

[4] J.-F. Côté, M. Gendreau, and J.-Y. Potvin. An exact algorithm for the two-dimensional orthogonal packing problem with unloading constrains. Technical Report CIRRELT-2013-26, Centre interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et le transport, April 2013.

[5] J. L. M. da Silveira, F. K. Miyazawa, and E. C. Xavier. Heuristics for the strip packing problem with unloading constraints. *Computers & OR*, 40(4):991–1003, 2013.

[6] J. L. M. da Silveira, F. K. Miyazawa, and E. C. Xavier. Two-dimensional strip packing with unloading constraints. *Discrete Applied Mathematics*, 164, Part 2(0):512 – 521, 2014.

[7] J. L. M. da Silveira, E. C. Xavier, and F. K. Miyazawa. A note on a two dimensional knapsack problem with unloading constraints. *RAIRO - Theor. Inf. and Applic.*, 47(4):315–324, 2013.

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

[9] G. Fuellerer, K. Doernera, R. Hartla, and M. Iori. Ant colony optimization for the two-dimensional loading vehicle routing problem. *Computers and Operations Research*, 36(3):655–673, 2009.

[10] M. Gendreau, M. Iori, G. Laporte, and S. Martello. A tabu search algorithm for a routing and container loading problem. *Transportation Science*, 40(3):342–350, 2006.

[11] M. Gendreau, M. Iori, G. Laporte, and S. Martello. A tabu search heuristic for the vehicle routing problem with two-dimensional loading constraints. *Networks*, 51(1):4–18, 2008.

[12] J. Gonçalves and M. Resende. Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics*, 17(5):487–525, 2011.

[13] J. Gonçalves and M. Resende. A biased random key genetic algorithm for 2d and 3d bin packing problems. *International Journal of Production Economics*, 145(2):500 – 510, 2013.

[14] J. C. Herz. A recursive computational procedure for two-dimensional stock-cutting. *ibm*, pages 462–469, 1972.

[15] J. Hooker. *Integrated Methods for Optimization*. International series in operations research & management science. Springer, 2007.

[16] M. Iori and S. Martello. An annotated bibliography of combined routing and loading problems. *YUGOSLAV JOURNAL OF OPERATIONS RESEARCH*, 23(3), 2013.

[17] M. Iori, J. Salazar-González, and D. Vigo. An exact approach for the vehicle routing problem with two-dimensional loading constrains. *Transportation Science*, 41(2):253–264, 2007.

[18] L. Junqueira, J. F. Oliveira, M. A. Carravilla, and R. Morabito. An optimization model for the vehicle routing problem with practical three-dimensional loading constraints. *International Transactions in Operational Research*, 20(5):645–666, 2013.

[19] J. Lysgaard, A. Lechtford, and R. Eglese. A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming*, 100(2):423–445, 2003. `http://www.hha.dk/~lys/`.

[20] S. Martello, D. Pisinger, and D. Vigo. The three-dimensional bin packing problem. *Operations Research*, 48(2):256–267, 2000. `http://www.diku.dk/~pisinger/`.

[21] S. Martello, D. Pisinger, D. Vigo, E. D. Boef, and J. Korst. Algorithm 864: General and robot-packable variants of the three-dimensional bin packing problem. *ACM Trans. Math. Softw.*, 33, March 2007.

[22] M. Mesyagutov, G. Scheithauer, and G. Belov. {LP} bounds in various constraint programming approaches for orthogonal packing. *Computers and Operations Research*, 39(10):2425 – 2438, 2012.

[23] M. Mesyagutov, G. Scheithauer, and G. Belov. New constraint programming approaches for 3d orthogonal packing. 2012.

[24] D. Naddef and G. Rinaldi. Branch-and-cut algorithms for the capacitated vrp. toth p. vigo d., eds. the vehicle routing problem. *SIAM Monographs on Discrete Mathematics and Applications*, 9:53–84, 2002.

[25] G. Reinelt. *TSPLIB - a traveling salesman problem library*. Report. Inst. für Mathematik, 1990.

[26] F. Rossi, P. v. Beek, and T. Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006.

[27] L. A. Wolsey. *Integer programming*. Wiley-Interscience, New York, NY, USA, 1998.

[28] E. Zachariadis, C. Tarantilis, and C. Kiranoudis. A guided tabu search for the vehicle routing problem with two-dimensional loading constraints. *European Journal of Operational Research*, 195(3):729–743, 2009.

Table 5: Performance different packing strategies

Instance					Iori					BNC						
Name	Class	n	M	K	z	$Cuts$	T_{tot}	Gap	#Heu	#Ex	#ExUn	$Cuts_{pack}$	$Cuts_{rout}$	T_{BNC}	Gap
E016-03m	1	15	15	3	273	209	0.92	0	101	0	0	0	154	0.80	0
	2	15	24	3	285	885	24.49	0	271	0	150	177	810	18.36	0
	3	15	31	3	280	641	22.05	0	459	0	225	250	1328	89.36	0
	4	15	37	4	288	28	2.69	0	21	0	6	6	28	0.16	0
	5	15	45	4	279	4	39	0	8	0	1	1	12	1.76	0
E016-03m	1	15	15	5	329	145	0.39	0	49	0	0	0	132	0.53	0
	2	15	25	5	342	397	4.27	0	99	0	28	29	526	2.95	0
	3	15	31	5	347	379	3.98	0	57	0	24	28	196	1.06	0
	4	15	40	5	336	212	19.09	0	103	0	7	8	392	2.15	0
	5	15	48	5	329	128	25.34	0	49	0	0	0	132	0.57	0
E021-04m	1	20	20	4	351	352	4.75	0	69	0	0	0	198	0.70	0
	2	20	29	5	389	1748	72.64	0	247	0	279	412	1052	19.07	0
	3	20	46	5	387	144	5.05	0	145	0	133	172	432	13.57	0
	4	20	44	5	374	86	59.36	0	129	0	38	42	222	5.95	0
	5	20	49	5	369	46	0.25	0	53	1	2	2	85	15.66	0
E021-06m	1	20	20	6	423	93	0.19	0	93	0	0	0	316	0.92	0
	2	20	32	6	434	238	2.08	0	90	0	28	28	533	1.91	0
	3	20	43	6	432	432	6.16	0	131	0	29	29	902	4.81	0
	4	20	50	6	438	368	6.08	0	104	1	25	25	659	17.57	0
	5	20	62	6	423	154	46.31	0	85	2	8	9	319	213.92	0
E022-04g	1	21	21	4	367	41	0.05	0	38	0	0	0	169	0.12	0
	2	21	31	4	380	126	2.34	0	126	0	45	50	349	2.69	0
	3	21	37	4	373	52	1.95	0	47	0	16	17	219	21.19	0
	4	21	41	4	377	383	35.31	0	58	0	22	24	211	0.75	0
	5	21	57	5	389	39	1865.66	0	29	0	4	4	59	2.24	0
E022-06m	1	21	21	6	488	519	5.73	0	151	0	0	0	545	2.63	0
	2	21	33	6	491	1010	26.52	0	132	0	42	52	852	3.77	0
	3	21	40	6	496	1081	35.45	0	319	0	100	110	2948	38.39	0
	4	21	57	6	489	491	7.19	0	181	0	54	67	1308	10.94	0
	5	21	56	6	488	509	4.67	0	144	2	5	5	545	236.87	0
E023-03g	1	22	22	3	558	30	0.02	0	14	0	0	0	16	0.06	0
	2	22	32	5	724	801	23.19	0	734	0	336	638	405	32.96	0
	3	22	41	5	698	47	3.25	0	329	0	166	201	158	18.98	0
	4	22	51	5	714	238	1668.05	0	734	0	354	680	362	172.79	0
	5	22	55	6	742	20	782.42	0	52	0	18	20	28	1.70	0
E023-05s	1	22	22	5	657	6	0.02	0	9	0	0	0	3	0.05	0
	2	22	29	5	720	980	34.42	0	611	0	430	889	389	41.26	0
	3	22	42	5	730	357	69.05	0	345	0	73	143	1327	79.65	0
	4	22	48	5	701	37	17.16	0	587	2	217	389	257	171.24	0
	5	22	52	6	721	8	956.05	0	11	0	0	0	3	0.10	0
E026-08m	1	25	25	8	609	727	19.34	0	114	0	0	0	541	1.89	0
	2	25	40	8	612	1199	56.49	0	78	0	21	23	677	0.97	0
	3	25	61	8	615	1244	156.97	0	153	0	33	35	2287	15.39	0
	4	25	63	8	626	1440	246.75	0	220	0	38	41	3455	47.73	0
	5	25	91	8	609	527	42.48	0	136	1	18	21	917	89.46	0
E030-03g	1	29	29	3	524	6207	55700.07	0	2526	0	0	0	8912	1537.01	0
	3	29	49	6	637	1620	999.16	0	657	1	42	56	1467	>3600	-
E033-03n	1	32	32	3	1991	319	6	0	50	0	0	0	42	0.75	0
E036-11h	1	35	35	11	682	1962	2128.13	0	319	0	0	0	12899	275.48	0
	2	35	56	11	682	1857	1427.5	0	237	0	50	51	10919	223.92	0
	3	35	74	11	682	1589	987.77	0	219	0	18	18	8886	365.75	0