Acta Cybernetica **20** (2011) 17–33.

Pickup and Delivery Vehicle Routing with Multidimensional Loading Constraints*

Tamás Bartók† and Csanád Imreh‡

Abstract

In this paper we introduce a new, pickup and delivery vehicle routing model where weight limits and also packing constraints are taken into account. In the model the vehicles have to transport 3-dimensional boxes from their pickup points into their delivery points. The boxes have weights and the vehicles has to satisfy a weight limit. We present a heuristic algorithm for the solution of the problem. The efficiency of the algorithm is evaluated by an experimental analysis.

Keywords: vehicle routing, multidimensional packing

1 Introduction

In logistics there are two important problems related to combinatorial optimization. One is the routing of the vehicles and the other problem is the container loading. Both areas have huge literature and many different models have been investigated. If both problems are taken into account then more difficult models appear, but they give a more adequate description of the real life problems.

In the area of vehicle routing many models are investigated, one can find a detailed description of the models in the surveys [4] and [16]. Usually the goal is to minimize the cost of the transportation, but in some recent works other cost functions like minimizing pollution are also considered (see [2]). One of the most important subfields of vehicle routing is the area of pickup and delivery problems. In these problems the requests are goods with a pickup point and a delivery point and the vehicles must transport them from the pickup point to the delivery point with minimal cost. These problems are NP-hard and several exponential time exact solution algorithms and metaheuristics are developed for their solution. One can find an overview about pickup and delivery problems in the survey [3].

Loading the vehicles leads to a 3-dimensional bin packing problem. If weight limits are taken into account, then we receive a common generalization of the vector

*Cs. Imreh was supported by the Bolyai Scholarship of the Hungarian Academy of Sciences.

†Institute of Informatics, University of Szeged, E-mail: tbartok@inf.u-szeged.hu

‡Institute of Informatics, University of Szeged, E-mail: cimreh@inf.u-szeged.hu

DOI: 10.14232/actacyb.20.1.2011.3

packing and box packing problems (see [1]). Algorithms for models, where both the routing and loading of vehicles are considered have been published only in the recent years. The capacitated vehicle routing problem (where each request has to be collected into a depot) with two dimensional loading constraints has been investigated first in the paper [9]. An algorithm which gives the exact optimal solution is presented for the case of two dimensional loading constraints in [10]. Improved tabu search algorithms are presented for the solution of that problem in [9] and [17]. An algorithm based on ant colony optimization is presented in [6]. The first paper on capacited vehicle routing with 3-dimensional loading constraints is presented in [8]. An improved tabu search based algorithm for 3-dimensional constraints is presented in [17], and an ant colony based algorithm is given in [7]. Pickup and delivery vehicle routing problems with two dimensional loading constraints are presented in [13]. An overview about the results on vehicle routing problems with loading constraints can be found in [11] and [18].

In this paper we consider a pickup and delivery problem with 3-dimensional loading constraints and with weight limit on the vehicles. As far as we know no algorithm for this model is presented in the literature. In the next section we give the mathematical model which is used in this paper. Then in Section 3 a heuristic algorithm is presented to solve the problem. Section 4 contains the description of the tests, we used to analyze the algorithm.

2 The mathematical model

In this model we are given an undirected simple graph $G = (V, E)$ which describes the road system which can be used by the vehicles. The input of the problem is a list of demands denoted by D and a list of vehicles denoted by R. The demands have the following parameters:

- the size which is a 3-dimensional box given by the sides x_j, y_j, z_j,
- the weight which is a positive real number w_j,
- pickup and delivery points: $s_j \in V$, $e_j \in V$.

The vehicles have the following parameters

- the size of the cargo which is a 3-dimensional box given by the sides X_v, Y_v, Z_v,
- the weight limit which is a positive real number W_v,
- the speed of the vehicle sp_v
- the start and end point of the vehicle s_v and e_v
- a cost function c_v which defines for each edge of G the travelling cost for vehicle v, we suppose that this is proportional to the distance, thus the time spent by the vehicle travelling on edge e is $c_v(e)/sp_v$

- time limit of the vehicle l_v which gives an upper bound on the time which the vehicle can travel before delivering its last demand

Our goal is to transport all of the demands by the given vehicles by a minimal cost solution, while not violating the defined constraints. In this model we do not allow to change the vehicle during the transportation of a demand, the vehicle must pick up the demand at its pick up point and deliver it to the delivery point. A vehicle can transport an unlimited number of freights at the same time as long as the size and weight constraints are not violated. We suppose that the cost of the solution is the total cost of the routes done by the vehicles. During the route of the vehicles the constraints must be satisfied at each time, which means that the demands have to packed into it without overlapping and the total weight is not allowed to be more than the weight limit. We note that in this model rotation of the items is not allowed. This assumption is realistic for automation based container loading. Moreover there is an extra assumption on the routes: each route has to be finished before the time limit of the vehicle. Here we do not take into account the time which is used to return to the end point, the limit is on the last delivery time. On the other hand one can easily modify our algorithm to the case when the time limit is for the arrival time at the end point.

Therefore the solution can be described as follows: Each vehicle v has a travel plan P_v which contains a list $p_1, \ldots, p_{k(v)}$ of vertices which - with the graph being simple - is a walk and for each vertex two sets are given: $in(p_i) \subseteq S$ contains the demands packed into the vehicle at vertex $p(i)$ and $out(p_i) \subseteq S$ contains the demands packed out from the vehicle at vertex $p(i)$, where S is a set of (a, b) couples ($a, b \in V$), where there is at least one transport request from a to b.

The solution is feasible if the following conditions are satisfied:

- $s_j = p_i$ for each $j \in in(p_i)$ and $v_j = p_i$ for each $j \in out(p_i)$, which means that a demand can be packed into a vehicle at its pickup point and can be packed out of the vehicle at its delivery point,

- $(c_v(s_v, p_1) + \sum_{i=1}^{k(v)-1} c_v(p_i, p_{i+1}))/sp_v \leq l_v$ is valid for each vehicle v, which means that the route without the return trip has to be finished within the time limit,

- at each point p_i the items which are in the vehicle ($j \in S$ with $j \in in(P_r)$ and $j \in out(P_q)$ for some $r \leq i < q$) must satisfy the loading constraints (the weight limit and the 3-dimensonal packing constraint)

Then the objective function is

$$\sum_{v \in R} (c_v(s_v, p_1) + c_v(p_{k(v)}, e_v) + \sum_{i=1}^{k(v)-1} c_v(p_i, p_{i+1}))$$

3 The heuristic algorithm

After initialization we build routes on the given graph for each vehicle, considering pairs of vertices, where there are requests in between. We are repeating this step as long as we have any unassigned pair of vertices. After this step we apply a simple local search method, with which we are swapping vertices along a route as long as we can decrease the total distance of a route, while keeping the linear ordering of the vertices. After the simple local search step, comes the execution of the packing step. The algorithm used here, is what is described in [1]. Then, we repeat the steps above, as long as we have any uninitialized vehicle and at least one untransported item. The last step is an advanced local seach method, with which we are changing vertices between routes, while keeping the feasibility of the packing.

Detailed description of the proposed algorithm

Example

In order to make the algorithm easier to understand, we use a simple example to demonstrate the steps of the algorithm.

In our example we have a simple graph, which consists of 7 vertices and 10 symmetrical edges. The vertices are named 1,2,...,7. The edges are as follows: 1-3, 1-4, 1-5, 2-3, 2-5, 3-4, 4-5, 4-7, 5-6, 6-7. The weight of the edges in the same order: 10, 10, 15, 15, 10, 5, 10, 5, 5, 15. We have 2 vehicles (v_1 and v_2), and 3 demands to fulfill. The vehicles are defined as follows: s_{v_1}=1, e_{v_1}=7, s_{v_2}=2, e_{v_2}=7, sp_{v_1}=sp_{v_2}=1, X_{v_1}=Y_{v_1}=Z_{v_1}=W_{v_1}=X_{v_2}=Y_{v_2}=Z_{v_2}=W_{v_2}=1, l_{v_1}=l_{v_2}=30. The items are defined as follows: x_j=z_j=0.75, y_j=0.5, w_j=0.5 for all demands. s_1=1, e_1=4, s_2=3, e_2=4, s_3=5, e_3=6. The cost of travelling on edge e: $c_v(e)$ = weight of e for both vehicles, thus it will be easier to follow the examples.

Phase 1 (Initialization)

First of all, given a graph $G(V, E)$, we create the set S (defined in previous chapter). Moreover, let S_i be the set of (a, b) couples, that are already inserted, initialized as an empty set. Let us assign a vehicle (taken from the initial set of vehicles R) to all couples in S, where $c_v(s_v, a) + c_v(b, e_v)$ is minimal.

Note: The assigment does not mean, that a certain demand can only be satisfied by v, it only means, if (a, b) is chosen as a first pair of vertices for a new route, then v is assigned to this route.

Example: S={ $(1, 4)$, $(3, 4)$, $(5, 6)$ }. The assigned vehicle will be v_1 for $(1, 4)$ and $(3, 4)$, and v_2 for $(5, 6)$.

Phase 2 (Route Building)

During this phase we repeatedly do the following steps:

Step 2.1 Choose a couple (a, b) from $S \backslash S_i$, for which $c_v(s, a, b, e)$ is minimal, where S_i denotes the set of investigated couples and $c_v(s, a, b, e) = c_v(s, a)+c_v(a, b)+c_v(b, e)$ with the vehicle v assigned to the pair (a, b) in the initialization. Let

Figure 1: Routes after Phase 2

r denote the route starting at s and continued by a and b, and ending at e. Insert (a, b) into S_i.

Step 2.2 Iteration: While the current route r can be continued with any couple from $S \setminus S_i$: Insert the couple (a, b) from $S \setminus S_i$, where total distance after inserting (a, b) into r before the position of e is minimal. Insert (a, b) into S_i. A route can be continued by a vertex, if the total time travelled (length of r from s to the last delivery point after the insertion divided by sp_v, where sp_v is the speed of v) is not greater than l_v, where l_v is the time limit for v.

Step 2.3 If vehicle v has been assigned to other couples in $S \setminus S_i$ before, repeat the vehicle assignment for those couples, allowing only unused vehicles. We have to do this reassignment since v cannot be used to serve these couples.

Step 2.4 If $S \setminus S_i$ is not empty and there is at least one unused vehicle that can be continued with any couple from $S \setminus S_i$, go to step 2.1.

Note: If there are no unused vehicles left, and no route can be continued by any item from $S \setminus S_i$, we still can not report the actual problem to be unsolvable, because the two local search methods in the later phases can still make it solvable.

Example: We start with couple $(1, 4)$, and we use v_1, as v_1 is the assigned vehicle for couple $(1, 4)$. We can insert both $(1, 4)$ and $(3, 4)$ into the route of v_1, at this point $c_{v_1}(r)$=25, but the total time, which is checked with l_{v_1} is 20. We can not insert the last couple $(5, 6)$ as it would increase the total time to 35, which is $>l_{v_1}$. After these steps: $S \setminus S_i$={ $(5, 6)$ }. (No reassignment of the vehicles is needed in Step 2.3, because v_1 is not assigned to $(5, 6)$). After the second iteration, the two routes are demonstrated on Figure 1.

Phase 3 (Simple Local Search)

This simple local search phase consists of many internal swaps within a single route, and no swaps are allowed between different routes. This is an essential step, if we consider that the vertices are inserted into the routes without any ordering. We repeat the following steps for route $r = p_1, \ldots, p_{k(v)}$ assigned to vehicle v:

Step 3.1 Let M be a set of pairs of positions in the route, initialized as an empty set. We will use this set for memorizing the already investigated pairs of vertices.

Step 3.2 Choose a (i, j) pair of positions, where $0 \leq i < k(v)$ and $p_i \neq p_j$ and (i, j) or (j, i) is not in M. If there is no such pair of positions, go to Step 3.5.

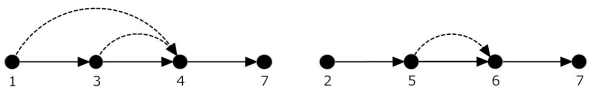

Figure 2: Routes after Phase 3

Step 3.3 Check if the swap at positions i and j would cause any of the requests, that are assigned to route r, unsatisfiable. This happens, if there is at least one request, for which, the start point does not precede the possible end points along r. If this swap would cause unfeasibility, insert (i, j) into M, go to Step 3.2. Otherwise, let us denote the new route r', go to Step 3.4.

Step 3.4 Let $c_v(r)$ denote the travelling cost for r, using v. If $c_v(r') < c_v(r)$, let $r := r'$, and delete every occurrence of i and j from M. Doing so, we allow further swaps of p_i and p_j in their new position. Continue with Step 3.2.

Step 3.5 If $S \setminus S_i$ is not empty, try to insert a pair of vertices from $S \setminus S_i$ into r, using the same method that was described in Phase 2. If any pair from $S \setminus S_i$ was successfully inserted, go to Step 3.2, otherwise proceed to the final step of phase 3.

After the iteration the length of the route is not greater than before the iteration, and no demand has been made unsatisfiable, which had been satisfiable before.

Final step of Phase 3

We may have the same vertex on r multiple times, and we also note that we had a constraint that $c_v(a, a)$=0, for every $v \in R$, where $a \in V$, therefore in many cases after the iteration we may have more instances of a vertex along r next to each other. In this final step, we iterate through r, and remove all surplus instances of a, so that a can not be followed by an other instance of a.

Note: We note that Phase 3 is indispensable in the algorithm. Without this phase the routes might contain several instances of a vertex and this would increase the cost a lot.

Example: Considering the route of v_1, which is $1 \to 1 \to 4 \to 3 \to 4 \to 7$, the only swap, which is possible, does not create unfeasibility and shortens the route (thus decreasing total cost) is the swap at positions 3 and 4 (p_3=4, p_4=3). After processing the swap the resulting route will be $1 \to 1 \to 3 \to 4 \to 4 \to 7$, thus the length of the route decreases from 25 to 20. For v_2 we can not process any swaps, which could decrease the total length of the route and retain feasibility at the same time. During the final step the duplicate instances are deleted, resulting in the routes demonstrated on Figure 2.

Phase 4 (Packing)

During this phase we try to load as many items into the vehicle along the route as many is possible. We treat the items with priority, that are more difficult to be

packed. An item can be transported by vehicle v, if and only if it can be loaded in the loading area of the vehicle for every vertex along the route r between the item's pickup and delivery point. Hereby we note that the same vertex can be on r multiple times, so we need to check all feasible possibilities. If the item can be loaded in the loading area of v for any of these possibilities, it is considered transportable. The input for this phase is the list of vertices for route r, and a list of items, item j is determined by its size x_j, y_j, z_j, by its weight w_j and by its pickup and delivery points, s_j and e_j, where s_j and e_j are vertices along the route r. We note the loading area of v (X_v, Y_v, Z_v) and weight limit W_v, specific for v. To pack the items we use the following alteration of the Block algorithm which is defined in [1]. In this application of the Block algorithm we do not aim to pack all the items in a given set to a minimal number of bins, but we pack the given items into one single vehicle's loading area. At each vertex in the route we use the following algorithm to pack the items.

Block algorithm

Step 4.1 (Classifying Phase) In this phase the items are divided into classes by their x-coordinates. Let $f_0 = L < f_1 < f_2 < \cdots < f_k = X$ be the list of the border points let C_i be the set of the boxes having the x-coordinate between f_{i-1} and f_i. The border points are determined by algorithm IPM given below.

Step 4.2 (y-Block Building Phase)

Step 4.2.1 For each class C_i first order the elements by the y-coordinate.

Step 4.2.2 If the list in the class is empty then we move to the next class. Otherwise we take the longest prefix of this list, which still fits into the container and does not exceed the maximal weight of the container.

Step 4.2.3 We remove these items from the list and replace them with a single item (y-Block), which has the x-value of the greatest x-value and a z value of the greatest z-value in the list, and a height(y) of Y. Go to Step 4.2.2. using the new actual list.

Step 4.3 (z-Block Building Phase) We take only those boxes into account in each iteration, which are in the i-th interval according to its x-coordinate. These are only y-Blocks.

Step 4.3.1 We sort these boxes by their z-coordinates into decreasing order.

Step 4.3.2 We form z-Blocks from this list using first-fit strategy, we always choose the first element which fits into the container and does not exceed the maximal weight of the whole container. If no more such element exists we move to Step 4.3.3.

Step 4.3.3 We remove these y-Blocks from the list and replace them with a single item (z-Block), which has the x value of the greatest x-value, a z-value of Z, and a y-value of Y.

Step 4.4 (The packing phase) After the z-Block building phase there are blocks with size of the form (x_i, Y, Z, w_i).

Step 4.4.1 Order the elements (z-Blocks) into decreasing order by $x_j w_j$. Go to Step 4.4.2.

Step 4.4.2 Pack the elements into the bins using a greedy strategy. This algorithm packs an element into the bin when it fits. If it does not fit then we skip the block. Note that in this case an element fits into the bin if it can be packed without overlapping and it does not violate the weight limit.

Considering the values of $f_1, \ldots, f_k$ we use the following algorithm to determine them. The algorithm puts at most K elements into each class, moreover it ensures that the sum of the elements is at least K for each consecutive pair of intervals.

Interval Preparation Method

Initialization part: Let $I = \{1, X\}$ be an ordered list of border points. Let K be the number of elements, we aim to have in each interval.

Step IPM1 Let f_i be the border point, for which the $[f_i, f_{i+1})$ interval contains the most elements. If this amount is at most K, then proceed with Step IPM2. Else, we divide this interval, with inserting a new border point into the list between f_i and f_{i+1}, with value of $(f_i + f_{i+1})/2$. Refresh the interval assignments and repeat Step 1.

Step IPM2: Let i be an index of I, for which the sum of elements in intervals $[f_i, f_{i+1})$ and $[f_{i+1}, f_{i+2})$ is minimal. If this sum is larger than K, then exit. Else concatenate these intervals by deleting border point f_{i+1} from the list I. Refresh element assignments and repeat Step IPM2.

Final step of Phase 4

If an item j is not transportable by v, and $(s_j, e_j) \in S_i$, where s_j and e_j are the pickup and delivery points for j, then delete (s_j, e_j) from S_i. This step is needed to ensure, that only those pair of vertices are prohibited from insertion into further routes, for those there are not unsatisfied demands. If $S \setminus S_i$ is not empty (means that there are untransported items), and there are unused vehicles, continue with Phase 2, Step 2.1. If there are not any unused vehicles, but there are untransported items, we still can not state, that we can not provide a feasible solution, because the local search method, described in Phase 5, can still make space for the untransported items.

Example: During this step the loading is trivial at all positions, and all items are transportable. Items 1 and 2 are packed to v_1 (picked up at positions 1 and 2), item 3 is packed to v_2 (picked up at position 2).

Phase 5 (Advanced Local Search)

The input for this phase is a set of routes, an assigned vehicle for each route, and a list of transported items for each route. During this phase, we will investigate all possible vertices, and we will try to move a vertex into an another route, keeping in mind to maintain the feasibility of the transportation.

Step 5.1 Choose a point a in route r_1, which cannot be the start or end point of r_1, and has not been investigated before, and choose a route r_2. If all possibilities are investigated, go to Step 5.7.

Step 5.2 Generate the "must-move neighbourhood" of the designated point. This contains the vertices, which have to be moved to the other route together with the designated point. This "must-move neighbourhood" is a sublist of positions of a route, vertices included are: the designated point, and all points that are start points of a request fulfilled during this route, and ending at the designated point, or endpoints of such request, starting at the designated point. Let us denote this by L_a. The ordering of the points must be also kept within L_a. Note, that this could also be used recursively, generating the Kleene closure of the designated point, using it, we would not need the following step, but practically, this closure is usually close the whole route, and exchanging almost whole routes with one-another would not end up in decrease of total distance.

Step 5.3 Determine the sublist L'_a of L_a. L'_a represents those positions of L_a, that can be deleted from their original route r_1. A vertex at a position can be deleted from its previous route if there are no such demands assigned to this route, that have their start point in L_a and their endpoint not in L_a or vice versa. $L_a \setminus L'_a$ represents the set of points in the "must-move neighbourhood" of a, where the points are tied to the original route and to L_a at the same time.

Step 5.4 Insert L_a into r_2, while keeping the linear order of points from L_a, delete L'_a from r_1.

Step 5.5 Execute phases 2, 3 and 4 for the new r_1 and r_2 to determine, whether all the items, that were previously transportable, are still transportable. If no, discard the changes, go to Step 5.1.

Step 5.6 If the new total sum of costs has been decreased by the swaps, save the changes, else discard them. Go to Step 5.1.

Step 5.7 If the total result value has been improved during the previous run of this local search procedure, restart the procedure (go to Step 5.1), otherwise proceed to Step 5.8.

Note: It may also happen, because of the various speeds of vehicles, that the total amount of distance travelled increases, but the total amount time consumed decreases.

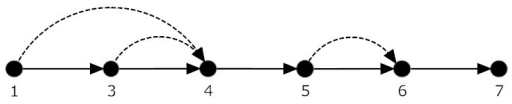

Figure 3: Solution after Phase 5

Step 5.8 If $S \setminus S_i$ is not empty, reexecute the algorithm, starting at Phase 2 Step 2.2, for each route. This reexecution places untransported items, whereever it may be possible. If there are still untransported items, report them as items not transported and a possible infeasibility of the input.

Example: Let us investigate the case, when r_1 denotes the route of v_2 and r_2 denotes the route of v_1. We can not choose vertex 2 from r_1, because it is a starting point, so let us choose vertex 5 (at position 2). The "must-move neighbourhood" L_a of vertex 5 will be: {2, 3}, representing vertices {5, 6}. L'_a will be set to {2, 3}, because there aren't other demands in r_1, which could retain an instance of vertex 5 or 6 in r_1. As we proceed to Step 5.4, we try to insert vertices 5 and 6 to r_2, while keeping the order of them. One possibility is to insert them between positions 2 and 3 (in r_2), but this would increase the length of r_2 from 20 to 50, and the total time would be 45 (the length of the route minus the last edge, which has a weight of 5), which is $> l_{v_1}$, therefore the insertion at these points is not feasible. We refrain from presenting all infeasible possible insertions, so we continue to check insertion between positions 3 and 4. This increases the length of the route to 40, but the time consumed is only 30. (the last edge on this route is 6-7, which has a weight of 10). This can be accepted, because l_{v_1}=30. As we can not find any better insertion, we proceed to Step 5.5. Fortunately all demands can be fulfilled (trivial), so we can continue with Step 5.6. The new total sum of costs can be decreased from 50 to 40, so we decide to keep the changes. We note that vehicle v_2 will not transport any items, therefore it will not be included in the solution. We demonstrate the final solution in Figure 3.

4 Description of tests

To our knowledge pickup and delivery vehicle routing with 3-dimensional loading constraints has not been investigated before, therefore no test instances have been published yet. We generated our test instances, combining the following methods:

- Graph: For the graph we used a part of the public roadsystem of county capitals of Hungary. Below we can see the visualization of the graph, reflected on the map of Hungary:

Figure 4: The graph

- Items: We extended the method with weights, which was used in [14] to generate the test demands (items), both uniform and gaussian (standard deviation: 1.0, the mean exactly halves the given interval) distributions were investigated. The pickup and delivery points were generated uniformly random among points from the input graph.
- Vehicles: Loading area generated as in [14], start- and endpoints are (uniformly) random points from the graph. Time limit is uniformly random from the 1000-2000 interval.

We were using the following testbed: Intel Core i5 750@3.15Ghz, Kingston 2x2 GB DDR3 1600Mhz, Gigabyte P55-UD3, MS Win7 x64, Java 1.6.0_23 x64

Testcases

For both distributions we performed 10 types of tests (the same intervals are used for generating the size as in [14]). We define in each test a maximal loading area for the vehicles it is $X \times Y \times Z$ and a weight limit denoted by W. In all testcases the sizes of the loading area of the vehicles are chosen as follows: $50\% : 1.0 \cdot MS$, $10\% : 0.9 \cdot MS, 10\% : 0.8 \cdot MS \ldots 10\% : 0.5 \cdot MS$ respectively, where MS is the maximal possible size of the loading area.

In the first 5 types of tests $X = Y = Z = 100$ and $W = 10000$. The intervals which are used for the uniform distribution are the following:

- Type 1: $x \in [1, 1/2X]$, $y \in [2/3Y, Y]$, $z \in [2/3Z, Z]$, $w \in [1, 2/3W]$.
- Type 2: $x \in [2/3X, X]$, $y \in [1, 1/2Y]$, $z \in [2/3Z, Z]$, $w \in [1, 2/3W]$.
- Type 3: $x \in [2/3X, X]$, $y \in [2/3Y, Y]$, $z \in [1, 1/2Z]$, $w \in [1, 2/3W]$.
- Type 4: $x \in [1/2X, X]$, $y \in [1/2Y, Y]$, $z \in [1/2Z, Z]$, $w \in [1, 2/3W]$.
- Type 5: $x \in [1, 1/2X]$, $y \in [1, 1/2Y]$, $z \in [1, 1/2Z]$, $w \in [1, 2/3W]$.

We note that the loading in the case of type 4 is obvious, each vehicle can only transport one item at a time. In the next 3 tests the sizes are $X = Y = Z = 10$, $X = Y = Z = 40$ and $X = Y = Z = 100$ respectively, the weight limit is $W = 10000$. The intervals which are used for the uniform distribution are the following:

- Type 6: $x \in [1, X]$, $y \in [1, Y]$, $z \in [1, Z]$, $w \in [1, 2/3W]$.
- Type 7: $x \in [1, 35]$, $y \in [1, 35]$, $z \in [1, 35]$, $w \in [1, 2/3W]$.
- Type 8: $x \in [1, X]$, $y \in [1, Y]$, $z \in [1, Z]$, $w \in [1, 2/3W]$.

The following two testcases were added to simulate the behaviour on many small items. Size of the maximal loading area for both: $X = Y = Z = 100$, $W = 10000$.

- Type 9: $x \in [1, 1/4X]$, $y \in [1, 1/4Y]$, $z \in [1, 1/4Z]$, $w \in [1, 1/4W]$.
- Type 10: $x \in [1, 1/8X]$, $y \in [1, 1/8Y]$, $z \in [1, 1/8Z]$, $w \in [1, 1/8W]$.

In the first class of tests the maximal number of vehicles: 30 for 100 items, 275 for 1000 and 1250 for 5000 items. We note that in most cases only a portion of these vehicles were actually used. We expect more calculation time needed for the last two test cases, as in these test cases, the items are much smaller, therefore the packing is more difficult.

We generated inputs of size 100, 1000 and of size 5000. All testcases on all sizes and distributions were executed 100 times, except for the largest testcases (N=5000), which were executed 20 times each. We executed the algorithm on the test cases and also considered the algorithm without the last most time consuming local search phase. As far as the running time is concerned, the presented algorithms are fast, we summarize the running times in Table 1. The running times are very similar for the two investigated distributions. If we consider the algorithm without the last local search phase, it is faster, the running time is decreased with approximately 25 percent. The experienced results were very similar in case of both investigated distributions, therefore we only present here one of them.

Table 1: The average running time of the full algorithm (msec)

N	T1	T2	T3	T4	T5
100	698.40	633.19	661.95	698.58	632.85
1000	13349.9	11901.8	12598.6	18715.5	12136.2
5000	199479	182741	190315	318304	180405
N	T6	T7	T8	T9	T10
100	666.02	652.07	651.58	863.79	872.12
1000	13753.8	12602.7	12477.3	16311.1	19223.5
5000	206946	186038	182971	194910	218564

We also considered the value of the cost function for the algorithm. Table 2 and 3 contain the cost for the case where the expansion of the items is generated by uniform and gaussian distribution. The first three lines contain the results of the full algorithm, the next three lines show the results when the last local search phase is omitted.

Table 2: The average result value on gaussian distribution (km)

N	T1	T2	T3	T4	T5
100	14237.96	14238.44	14098.83	22360.46	14213.81
1000	124915	124825	124775	186269	125460
5000	583884	588904	586226	813913	583893
100	17615.72	17632.82	17479.83;	34355.57	17655.96
1000	149285	149624	149656	287612	149873
5000	690341	693146	693911	1214585	689334
N	T6	T7	T8	T9	T10
100	17490.91	14378.77	15381.34	9670.62	9522.66
1000	146063	126528	133402	87861	86065
5000	670274	594189	602684	435602	429330
100	23293.37	17775.66	19323.8	12492.3	12290.18
1000	189296	151233	161949	97865	96426
5000	828233	703710	719073	470851	462743

Table 3: The average result value on uniform distribution (km)

N	T1	T2	T3	T4	T5
100	14475.28	14356.12	14604.09	22307.21	14066.24
1000	125852	126217	126756	185120	123222
5000	574761	580987	582221	816104	571117
100	18242.5	18247.02	18346.11;	34219.99	17889.46
1000	152161	152508	153411	287123	150197
5000	676405	681878	687960	1222230	675340
N	T6	T7	T8	T9	T10
100	15807.48	14633.3	15146.15	9615.78	9398.92
1000	135009	127794	130426	88013	86410
5000	620470	581959	594788	434717	424375
100	20460.28	18418.25	19245.48	12527.19	12345.06
1000	167217	154564	159579	97967	96756
5000	742652	688937	704292	467698	457200

We saved the actual result value of each run of the local search procedure. In Figure 5 we can see the average result value in the ratio of the result value prior to the execution of the last local search phase (which is marked as 1.0). Each bar represents one run of the local search procedure for the actual testcase. We can observe, that the most improvements were made (to 67% of the starting result

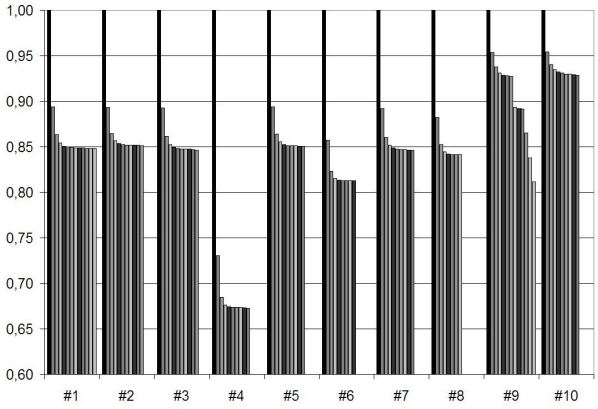

Figure 5: Percentages of result value, by each optimization cycle for the 10 testcases

value) in those testcases, when the initial result value had been worse, and the least improvement was made in testcase 10, in which the initial result value was previously the best. We also note that we can observe different number of bars for some testcases, it refers to the different number of local search runs needed for the testcase.

If we observe Table 4, we can see that the last local search has not only improved the solution value, but slightly decreased the number of required vehicles in all cases. Which can also be another reason for the decreased solution value. The first three lines contain the results of the full algorithm, the next three lines show the results when the last local search phase is omitted.

Table 4: The average number of vehicles in the solution (pcs)

N	T1	T2	T3	T4	T5
100	14.48	14.49	14.48	29.39	14.29
1000	136.8	137.29	137.04	272.85	137.70
5000	698.17	705.38	702.25	1247.5	700.77
100	14.9	14.89	14.84;	30.00	14.75
1000	137.155	137.69	137.35	274.57	138.12
5000	698.75	705.83	702.62	1250.0	701.22
N	T6	T7	T8	T9	T10
100	19.93	14.71	15.93	9.07	9.04
1000	172.61	138.91	148.71	83.96	82.28
5000	820.1	712.87	724.83	453.14	441.62
100	20.38	15.05	16.37	9.98	9.86
1000	173.24	139.18	149.04	85.2	84.13
5000	820.72	713.27	725.0	456.0	444.9

We also investigated the algorithm in the case where there are many items and fewer vehicles are available and it is not possible to serve all of the demands. In this case we investigated the number of the unsatisfied demands with the number of vehicles decreased to 60% of the previous tests. The results are summarized in table 5.

Table 5: The average number of unfulfilled demands (pcs)

N	T1	T2	T3	T4	T5
100	0.6	1.0	0.6	2.7	0.45
1000	0.75	2.5	3.25	28.5	1.0
5000	10.6	11.3	28.0	104.4	16.5
N	T6	T7	T8	T9	T10
100	0.45	0.45	0.95	0.0	0.0
1000	2.5	1.0	0.5	0.0	0.0
5000	27.5	4.1	8.0	0.0	0.0

5 Conclusions

In this paper we have presented a new vehicle routing model which gives an adequate description of practical problems. We presented a multi level heuristic algorithm for the solution of the problem which has a reasonable running time even for inputs of large time. Concerning the last more time consuming local search phase we could observe that it makes in average a $15 - 20\%$ improvement in the solution given by the first part.

Acknowledgment

The authors would like to thank the anonymous referees for their helpful advice and suggestions concerning this paper.

References

[1] T. Bartók, Cs. Imreh, Heuristic algorithms for the weight constrained 3-dimensional bin packing model, submitted for publication

[2] M. Bárány, B. Bertók, Z. Kovács, F. Friedler, and L. T. Fan, Optimization Software for Solving Vehicle Assignment Problems to Minimize Cost and Environmental Impact of Transportation, *Chemical Engineering Transactions*, **21**, 499–504, 2010.

[3] G. Berbeglia, J. F. Cordeau, I. Gribkovskaia, G. Laporte, Static pickup and delivery problems: a classification scheme and survey, *TOP*, **15(1)**, 1–31, 2007

[4] J.F. Cordeau, G. Laporte, M.W.P. Savelsbergh, D. Vigo, Vehicle routing, in: *Transportation, Handbooks in Operations Research and Management Science*, **vol. 14**, 367-417, 2007.

[5] K.F. Doerner, G. Fuellerer, M. Gronalt, R. Hartl, M. Iori, Metaheuristics for vehicle routing problems with loading constraints, *Networks*, **49(4)**, 294–307, 2007.

[6] M. Fuellerer, K.F. Doerner, R. Hartl, M. Iori, Ant colony optimization for the two-dimensional loading vehicle routing problem, *Computers & Operations Research*, **36**, 655-673, 2009.

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

[8] M. Gendreau, M. Iori, G. Laporte, S. Martello, A tabu search algorithm for a routing and container loading problem, *Transportation Science*, **40**, 342-350, 2006

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

[10] M. Iori, G. Salazar, D. Vigo, An exact approach for the vehicle routing problem with twodimensional loading constraints *Transportation Science*, **41**, 253-264, 2007

[11] M. Iori, S. Martello, Routing problems with loading constraints, *TOP* **18(1)**, 4–27, 2010

[12] A. Lodi, S. Martello, D. Vigo, Heuristic algorithms for the three-dimensional bin packing problem, *European Journal of Operational Research*, **141**, 410-420, 2002.

[13] Malapert A, Guert C, Jussien N, Langevin A, Rousseau L-M Two-dimensional pickup and delivery routing problem with loading constraints. *In: Proceedings of the first CPAIOR workshop on bin packing and placement constraints (BPPC08)*, http://contraintes.inria.fr/CPAIOR08/BPPC/bppc08submission10.pdf

[14] S. Martello, D. Pisinger, D. Vigo, The Three-Dimensional Bin Packing Problem, *Operations Research*, **48(2)**, 256-267, 2000.

[15] C.D. Tarantilis, E.E: Zachariadis, C.T. Kiranoudis, A hybrid metaheuristic algorithm for the integrated vehicle routing and three-dimensional container-loading problem *IEEE Trans Intell Transp Syst*, **10**, 255-271, 2009

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

[17] E.E. Zachariadis, C.D. Tarantilis, C.T. Kiranoudis, A guided tabu search for the vehicle routing problem with two-dimensional loading constraints, *European Journal of Operational Research*, **195**, 729-743, 2009.

[18] F. Wang, Y. Tao, N. Shi, A survey on vehicle routing problem with loading constraints. *Proceedings of the 2009 International Joint Conference on Computational Sciences and Optimization*, **2**, 602-606, 2009