Three-dimensional one-to-one pickup and delivery routing problem with loading constraints.

Oleksandr Lytvynenko[1], Oleksij Baranov[1], Remy Dupas[2], Igor Grebennik[1]
[1] Kharkiv National University of Radio Electronics, 61000 Kharkiv, Ukraine
[2] Univ. Bordeaux IMS, UMR 5218, F-33400 Talence, France

Abstract. We propose mathematical model and solving strategy for PDP with 3D loading constraints in terms of combinatorial configuration instead of traditional approach that uses boolean variables. We solve traditional one-to-one Pickup and Delivery Problem in combination with problem of packing delivered items into vehicles by means of proposed combinatorial generation algorithm.

Keywords: Pickup and Delivery problem, loading constraints, combinatorial generation, packing of parallelepipeds.

1 Introduction

There are a lot of articles dedicated to finding various algorithms for solving Pickup and Delivery Problem (i.g., [1-3]). Several of them take into account real-world loading constraints describing them as LIFO ([4]) or FIFO ([5]) buffers, or in form of 2D [6] or 3D [7-8] loading constraints. However, usually objective function and all limitations in PDP are described as inequalities with boolean variables. Also, in most of algorithms solving PDP, it's hard or impossible to regulate balance between solving time and result precision. In this article, we describe PDP with 3D loading constraints in terms of combinatorial sets instead of traditional description that use boolean variables. Also, we solve Pickup and Delivery Problem by means of combinatorial generation algorithm. Described algorithm can balance between solution quality and solution time and is very flexible at all. Also, it produces comparatively good results in adjustable time.

2. Problem Formulation

We consider classical Pickup and Delivery Problem (PDP) [1], one-to-one, symmetric case, i.e. every arc (i,j) is equal to the arc (j,i) and could be replaced by one edge. Pickup and delivery problem is modeled on complete graph $G=(V,A)$ where V is the set of all vertices, $V=\{0,1,...,2n+1\}$, where 0 and $2n+1$ denote the depot, and A the set of all arcs. There are v identical vehicles available; each vehicle has a weight capacity Q and a three-dimensional rectangular loading space defined by width W, height H and length L. Each client i, $i \in J_n$, $J_n=\{1,2,...,n\}$, requires the pickup or delivery of one three-dimensional item having width w_i, height h_i and length l_i with total weight q_i (pickup nodes are associated with a positive value q_i, delivery nodes with a negative value $-q_i$). We assume that *all items are rectangular boxes*. There are next *limitations* on loading items into a vehicle (*3D-constraints*): (1) The items can only be placed orthogonally inside a vehicle; however, items can be rotated by $90°$ on the width–length plane; (2) The stability of the packed items is important; one method to ensure stability is to require items that are placed on top of other items to have sufficient supporting areas. A packing is feasible if all items are either placed directly on the floor of the vehicle or on top of other items with total supporting area of at least some percent of their base areas; (3) All items can be easily unloaded in appropriated delivery point. When delivery client i is visited, its corresponding item must not be stacked beneath nor be blocked by items of clients that are to be visited later.

The *objective* is to find a set of at most v routes (one per vehicle) such that: (1) Every vehicle starts from the depot, visits a sequence of clients and returns to the depot; (2) All clients are served, and every client is served by exactly one vehicle; (3) No vehicle carries a total weight that exceeds its capacity; (4) All items demanded by all the clients served by a vehicle can be orthogonally packed into that vehicle while satisfying 3D-constraints; (5) The total cost of all edges included in the routes is minimized.

3. Mathematical Model

3.1. Designations

P ... set of backhauls or pickup vertexes (clients), $P = \{1, 2, ..., n\}$

D ... set of line hauls or delivery vertexes (clients), $D = \{n+1, n+2, ..., 2n\}$

q_i ... load at vertex i; pickup nodes are associated with a positive value q_i, delivery nodes with a negative value $-q_i$, $i \in J_{2n}, J_{2n} = \{1, 2, ..., 2n\}$,

w_i, h_i, l_i … width, height and length of the loaded (unloaded) item at vertex i respectively, $i \in J_{2n}$,

v ... number of vehicles,

Q ... capacity of each vehicle,

W, H, L … width, height and length of the vehicle loading space respectively,

C …set of pairs of corresponding pickup and delivery clients, $C = \{(p_1, d_1), (p_2, d_2), ..., (p_n, d_n)\}$,

$(p_i, d_i) \in C$ … pair of corresponding pickup and delivery clients, $p_i \in P$, $d_i \in D$, $d_i = p_i + n$, $i \in J_n$,

μ … number of loaded vehicles, $\mu \le \nu$,

$C_1, C_2, ..., C_\mu$ … a partition of C, $C = \bigcup_{j=1}^{\mu} C_j$, $C_i \cap C_j = \varnothing$, $i \in J_\mu$, $j \in J_\mu$;

there is a one-to-one correspondence between each set C_j of pickup and delivery clients and loaded vehicle j which serves this set of clients, $j \in J_\mu$,

$n_j = Card\, C_j$, $j \in J_\mu$, $\sum_{j=1}^{\mu} n_j = n$,

$c(i, j)$ . . . cost to traverse arc or edge (i, j),

$V_j = \{i_1^j, i_2^j, ..., i_{2n_j}^j\}$ … set of all vertexes included in C_j.

$P(V_j)$ … set of permutations generated by elements of V_j; this set describes all possible paths of vehicle *j*, i.e. all possible sequences of visiting all vertexes served by vehicle.

$Q(i_k^j)$ ... load of vehicle j when arriving at vertex $i_k^j, k \in J_{2n_j}$.

$u_0^j = (x_0^j, y_0^j, z_0^j)$ … coordinates of the pole of an placement area in a vehicle j.

3.2. Decision variables of the problem.

$U = (U^1, U^2, ..., U^\mu)$, $U^j = (u_1^j, u_2^j, ..., u_{n_j}^j)$, where

$u_i^j = (x_i^j, y_i^j, z_i^j)$ … coordinates of the pole of an item i in a vehicle j corresponding to set C_j, $i \in V_j$, $j \in J_\mu$;

$\pi^j \in P(V_j)$ … path (i.e. the sequence of visiting vertexes) of vehicle *j;*

$v_i^j = (lw_i, h_i), lw_i \in \{(l_i, w_i), (w_i, l_i)\}$ … orientation of an item i in a vehicle j, $i \in V_j$, $j \in J_\mu$,

We have *n* items which have a form of parallelepipeds

$\Pi_i = \{x \in R^3 : x = (x_1, x_2, x_3) \,|\, 0 \le x_1 \le l_i, 0 \le x_2 \le w_i, 0 \le x_3 \le h_i\}$, $i \in J_n$,

ν identical placement areas D_j (also having a form of parallelepipeds) are given, $j \in J_\nu$:

$D_j = \{x \in R^3 : x = (x_1, x_2, x_3) \,|\, 0 \le x_1 \le L, 0 \le x_2 \le W, 0 \le x_3 \le H\}$

3.3. Φ-functions

Φ-functions are used for mathematical modeling and solving of wide classes of problems of placement various geometrical objects (see [9,10]). They allow to describe conditions of touching, intersection and non-intersection of geometrical objects. Also a big advantage of Φ-functions is that they allow to describe problem of placement of geometrical objects as a mathematical programming problem.In this article, we use Φ-functions to describe formally conditions of mutual non-intersection for two parallelepipeds and condition of correct placement of parallelepiped into the placement area. Article [10] contains more detail information about such application of Φ-functions.To describe 3D constraints, we use 2 types of Φ-functions:

$$\Phi_{il}^{j}(u_i^j, u_m^j, v_i^j, v_m^j) = \max\{x_m^j - x_i^j - v_{1i}, -x_m^j + x_i^j - v_{1m}, \quad (1)$$
$$y_m^j - y_i^j - v_{2i}, -y_m^j + y_i^j - v_{2m}, z_m^j - z_i^j - v_{3i}, -z_m^j + z_i^j - v_{3m}\}$$

– Φ-function of pair of parallelepipeds – for checking whether item *i* (determined by pole coordinates u_i^j, linear dimensions x_i^j, y_i^j, z_i^j and orientation v_i^j) intersects with item *m* (determined by pole coordinates u_m^j, linear dimensions x_m^j, y_m^j, z_m^j and orientation v_m^j) (if (1) ≥ 0 then items don't intersect);

$$\Phi_{0m}^{j}(u_0^j, u_m^j, v_m^j) = \min\{x_m^j - x_0^j, -x_m^j + x_0^j + L - v_{1m}, \quad (2)$$
$$y_m^j - y_0^j, -y_m^j + y_0^j + W - v_{2m}, z_m^j - z_0^j, -z_m^j + z_0^j + H - v_{3m}\}.$$

– Φ-function of parallelepiped and placement area – for checking whether item *m* can be placed into a placement area (if (2) ≥ 0 then item can be placed);

3.4. Objective function and constraints

$$\sum_{j=1}^{\mu}[c(0, i_1^j) + \sum_{k=1}^{2n_j-1} c(i_k^j, i_{k+1}^j) + c(i_{2n_j}^j, 2n+1)] \to \min, \quad (3)$$

$$Q(i_k^j) = \sum_{k=1}^{s} f(i_k^j) \leq Q, \ \forall s \in J_{2n_j}, \ j \in J_\mu, \quad (4)$$

$$f(i) = \begin{cases} q_i, & \text{if } i <= n \text{ (vertex is a pickup)}, \\ -q_i, & \text{if } i > n \text{ (vertex is a delivery)}, \end{cases}$$

$$\begin{cases} \Phi_{im}^{j}(u_i^j, u_m^j, v_i^j, v_m^j) \geq 0, \ i, m \in J_n, i < m, \\ \Phi_{0m}^{j}(u_0^j, u_m^j, v_m^j) \geq 0, \ m \in J_n. \end{cases}, \quad j \in J_\mu. \quad (5)$$

Here $c(0, i_1^j)$ is a distance from the depot (vertex 0) to the first visited vertex, $c(i_{2n_j}^j, 2n+1)$ – from the last visited vertex to the depot. As mentioned above, vertexes *0* and *2n+1* are different designations for a single depot. Other designations are already described in previous sections.

4. Decision Strategy

We propose a two-level strategy for solving of the problem. On the *upper* level we implement a *partitioning* of set C of pickup-delivery pairs to subsets $C_1, C_2, ..., C_\mu$. Methods of the partitioning may be different, for example heuristic, something like clustering when solving large scale traveling salesman problem. We have to put to each cluster C_j pairs (p_i, d_i) of corresponding pickup and delivery clients which must be served by one vehicle. Making clusters for that problem is not the main purpose of our work, that's why we chose the simplest *k*-means clustering algorithm. Classical k-means algorithm [11] deals with single points, but we want to make clusters of pairs (p_i, d_i). We substitute pair (p_i, d_i) with single point k_i, which is the middle point between p_i and d_i.

On the *lower* level, we deal with single cluster C_j served by single vehicle and construct a path (sequence of vertex visiting) for it. We consider a permutation of elements of set V_j: $\pi^j \in P(V_j)$, which describes the path of vehicle. The permutation also defines an order of loading and unloading of items for vehicle j and thus order of unloading of items to/from the vehicle (inverse order to π^j) and an order of visits of clients corresponding to order of loading/unloading. Path π^j should satisfy all described limitations. To take into account item rotations, we substitute each element in π^j (i.e. vertex) by vector $lw_i \in \{(l_i, w_i), (w_i, l_i)\}$, $i \in V_j$. So finally we obtain the composition of permutations. So, to construct a path for single vehicle *j* we should choose an optimal (according to (3)) permutation of its vertexes V_j + determine orientation $lw_i \in \{(l_i, w_i), (w_i, l_i)\}$ of each vertex in a path. Below we solve a problem of generating optimal (according to (1)) path π^j for single vehicle *j*.

5. Solving Algorithm For Lower Level

Exact solution. To solve the problem of generating permutations π^j, we use the algorithm *GenBase* described in [12]. This algorithm is quite universal because of its ability to generate various combinatorial sets with given properties. For the convenience of further presentation, let us denote the path of the current vehicle *j* as $t = \pi^j$; also first *i* vertexes of path *t* will be called *partial path* $t^i = (t_1, t_2, ..., t_i)$. The algorithm has a recursive nature: at each recursion level $i \in J^0_{2n-1}$ $, J^0_{2n-1} = \{0, 1 ... 2n-1\}$ it expands the current partial path $t^i = (t_1, t_2, ..., t_i)$ by adding the next vertex t_{i+1} at the end of the path and thus obtaining a new partial path $t^{i+1} = (t_1, t_2, ..., t_{i+1})$ of length $i+1$ at the next level. Consequently, at level *i=2n*, the desired path $t^{2n} = t$ is obtained. In other words, on each iteration the algorithm just adds a new vertex to the current partial path. Elements t_{i+1} must satisfy some restrictions arising from specific features of each combinatorial set. Let us denote a tuple of all those elements as $F^i = (f_1, f_2, ..., f_k)$ at each level $i \in J^0_{2n-1}$. Now we can say that for each $j \in J_k, J_k = \{1, 2 ... k\}$ the algorithm adds a new element

$t_{i+1} = f_j$ to the current path $t^i = (t_1, t_2, ..., t_i)$ and makes a recursive call of itself with an extended partial path $t^{i+1} = (t_1, t_2, ..., f_j)$. To generate all the paths, algorithm *GenBase* should be called with an empty path $t^0 = (\ )$. For PDP paths, tuple F^i contains only vertexes satisfying the following restrictions: 1) vertexes in t^i are unique ($t_{i+1} \neq t_z, z = 1..i$); 2) each "pickup" vertex must be visited *before* a corresponding "delivery" vertex. It means that delivery vertex can be added to path t^i only when it already has a corresponding "pickup" vertex ($t_{i+1} > n \Rightarrow \exists z : (t_{i+1} - n) = t_z$); 3) restrictions (2) upon the maximum payload capacity of the vehicle; 4) if t_{i+1} is a pickup, then new box will be placed into a vehicle. This requires to *check 3D constraints if* t_{i+1} *is a pickup*. To validate 3D-constraints, we use the algorithm described in [10]. Note that the algorithm for validating 3D-constraints can rotate each item in the horizontal plane if required. This leads to determining vectors lw_i, $i \in V_j$. Described algorithm products a recursive tree, where each vertex at levels i < 2n-1 is a *partial path* and vertexes at last level i = 2n-1 are *full paths*. While running algorithm, at levels i < 2n-1 we *expand* each tree leaf, i.e. add new vertex from F^i to a partial path. When all paths generated (tree is complete), we choose a solution: it's a path with best value of (1).

Heuristic solution. As there are a large number of pickup-delivery pairs in real PDP problems, we need an heuristic for generating not all, but some good paths. For this we'll modify a described procedure for obtaining exact solution. Because of algorithm *GenBase* is universal, it is easy to inject any heuristics into it at the step of *expanding tree leaves* (adding points from F^i to partial solutions). We can select 'best' points from F^i and use only them for expanding tree leaves excluding other, 'bad' points. For the considered problem, the heuristic can be as follows: (*H1*) Vertexes in F^i are sorted by ascending the distance from: the depot, if i=0 (it's easy to assume that the first vertex of the path should be near the depot); the last vertex in the current path (t_i) (if i >0). (*H2*) Recursive calls of *GenBase* are made only for the first *RBW%* of elements of the tuple F^i, where *RBW* is the predefined constant. So we expand *RBW%* tree leaves. Other vertexes are excluded from further consideration. (*H3*). As checking 3D constraints is a complex procedure, we check them *not every time* pickup is added but with predefined probability *check_prob* ∈ [0;1] at all levels except the last i=2n-1. At last level i=2n-1, we *always* check 3D constraints to prevent invalid path to be a final solution. Combination of *H1* and *H2* of described heuristic a modification of well-known *beam search* procedure [13]. In regular beam search, *beam width* is a constant while our "*beam width*" is a *relative* value – some *percent* of tree vertexes. So let us denote it as *relative beam width (RBW)*. By changing the value of *RBW,* we can balance between time of work and the *accuracy* of results: we can include more or less elements of F^i to the further path generation.

6. Computational Experiments

We used described heuristic to solve one-to-one PDP problems for *n* up to 50. To evaluate the efficiency (result quality and solution time) of proposed heuristic, we obtained exact and heuristic solutions for the bunch of instances for *n*=3,4,5,6 (total 120 instances). For each instance, we tried to obtain *exact* solution and *3 heuristic solutions* for *RBW*=10,30,50%. For all instances, *check_prob* was 0.2. After obtaining heuristic solutions, we compared resulting cost (1) with cost of optimal path obtained in exact solution, so we obtained relative *cost increase*. We analyzed how *cost_increase* (Fig.1), heuristic solution time (Fig.2) and relative frequency of 'jack pot' (heuristic produces an optimal solution, Fig.3) depend on *RBW* for various *n*.

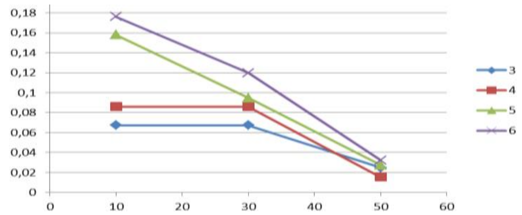

Fig. 1. How *cost_increase* (*y* - axis) depends on *n* (see legend) and *RBW* (*x* - axis)

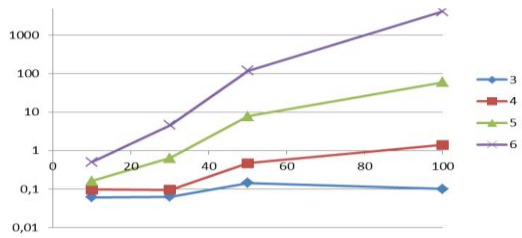

Fig. 2. How heuristic solution time (*y* - axis) depends on *n* and *RBW* (*x* - axis)

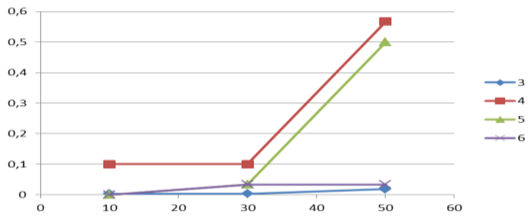

Fig. 3. Relative frequency of 'jack pot' (*y* - axis) for various *n* and *RBW* (*x* - axis)

7. Conclusions

We proposed mathematical model and solving strategy for PDP with 3D loading constraints in terms of combinatorial configuration instead of traditional approach that uses boolean variables. We solved traditional one-to-one Pickup and Delivery Problem in combination with problem of packing delivered items into vehicles by means of proposed combinatorial generation algorithm. The main advantages of proposed approach and solution strategy are: (a) ability to balance between solution quality and time (by varying *RBW* parameter); (b) ability to get not one best solution but a set of good solutions (because we generate many admissible solutions in our algorithm); (c) flexibility of solution algorithm: by changing way of determining F^i one can easily change algorithm logic. Proposed algorithm is quite universal: it has been already used in [12] for solving other optimization and combinatorial problems.

References

1. D. Pisinger, S. Ropke. A general heuristic for vehicle routing problems. Computers & Operations Research, Vol. 34, Issue 8 (2007), 2403-2435.
2. Renaud J, Boctor FF, Ouenniche J (2000) A heuristic for the pickup and delivery traveling salesman problem. Comput Oper Res 27:905–916
3. S. Ropke, J.-F. Cordeau (2008). Branch-and-Cut-and-Price for the Pickup and Delivery Problem with Time Windows
4. J.-F. Côté, M. Gendreau, J.-Y. Potvin. Large Neighborhood Search for the Pickup and Delivery Traveling Salesman Problem with Multiple Stacks. Wiley Periodicals, Inc. NETWORKS, Vol. 60(1). – 2012, pp. 19-30.
5. G. Erdoğan, J.-F. Cordeau, and G. Laporte. The pickup and delivery traveling salesman problem with first-in-first-out loading. Computers & Operations Research, 36:1800-1808, 2009.
6. A. Malapert, C. Guerét, N. Jussien, A. Langevin, and L.-M. Rousseau. Two-dimensional pickup and delivery routing problem with loading constraints. In Proceedings of the First CPAIOR Workshop on Bin Packing and Placement Constraints (BPPC'08), May 2008, Paris, France
7. E.E. Zachariadis, C.D. Tarantilis, and C.T. Kiranoudis. The pallet-packing vehicle routing problem. Transportation Science, 46:341-358, 2012
8. 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 (2010) 751–759
9. G. Scheithauer, Yu. G. Stoyan, T. Ye. Romanova Mathematical Modeling of Interactions of Primary Geometric 3D Objects // Cybernetics and Systems Analysis, 41(3), 2010, pp. 332–342.
10. I. V. Grebennik, A. V. Pankratov, A. M. Chugay, A. V. Baranov Packing n-dimensional parallelepipeds with the feasibility of changing their orthogonal orientation in an n-dimensional parallelepiped // Cybernetics and Systems Analysis, 46(5), 2010.– P. 793–802.
11. J.B. MacQueen. Some Methods for classification and Analysis of Multivariate Observations. 5th Berkeley Symposium on Mathematical Statistics and Probability 1. University of California Press, 1967.
12. I. Grebennik, O. Lytvynenko. Generating combinatorial sets with given properties. Cybernetics and Systems Analysis. Volume 48, Issue 6, November 2012, pp 890-898.
13. Bruce T. Lowerre. "The Harpy Speech Recognition System", Ph.D. thesis, Carnegie Mellon University, 1976.