TRUCK LOADING PROBLEM AND HEURISTIC METHODS

Tailai Song
Politecnico di Torino
Corso Duca degli Abruzzi, 24 10129 TO, ITALY
s287288@studenti.polito.it

Zhiqiang Zhao
Politecnico di Torino
Corso Duca degli Abruzzi, 24 10129 TO, ITALY
s277962@studenti.polito.it

Di Wang
Politecnico di Torino
Corso Duca degli Abruzzi, 24 10129 TO, ITALY
s275509@studenti.polito.it

Yu Cheng
Politecnico di Torino
Corso Duca degli Abruzzi, 24 10129 TO, ITALY
s288483@studenti.polito.it

July 10, 2021

ABSTRACT

This report will analyze and solve the truck loading problem through both the exact method using Gurobi and heuristic methods. Moreover, based on the comparison in terms of computational time and gap of solution, we gain a deeper understanding of operational research and more experiences in the field of applied mathematics.

1 Introduction

1.1 Introduction to the problem

The truck loading problem states that a vehicle with m compartments transports q different products of various sizes from a source to n different destinations. A compartment can be loaded with different products at the same time and all compartments can have the same or different capacity. The destinations have different demands for each kind of products. The demands of all the destinations must have to be satisfied all the time, which means as long as there is a lack of any kind of products at any destinations, the replenishment has to be delivered. The replenishment time is the time interval between two deliveries. The operational problem is how to load the compartments of the vehicle so that the replenishment time is maximized.

1.2 Review of similar models

The family of truck loading problems, composed of many different types of loading problem branches, has been studied by many researchers. Ümit Yüceer and Arif Özakça(2010) have developed a mixed-integer linear programming model, from which they obtained weighted distribution formed sub-problems. Then the corresponding sub-algorithms and a main algorithm to limit the uncertainty interval are proposed to maximize the replenishment time. Balaji and Mukund Nilakantan(2019) studied the fixed charge transportation problem with truck load constraints (FCT-TLC) problem, and they proposed a Genetic Algorithm (GA) and a Simulated Annealing Algorithm (SAA) to minimize the cost. Maria Teresa Alonso and Ramon Alvarez-Valdes(2016) studied a truck loading problem considering the products are first loading in pallets, besides the dimension and load constraints, there are many other constraints, related to the maximum weight supported by each axle and the distribution of the load inside the truck. They have developed an improved GRASP algorithm, in which the constructive algorithm is randomized and an improvement phase is added, which including eliminating a percentage of the last trucks in the solution and refilling these trucks using another strategy, for those trucks that have been closed because an axle has reached its maximum weight, they try to obtain better results by swapping pallets between consecutive trucks so that as many pallets as possible can be loaded in the first trucks. Myrna Palmgren(2005) considered a daily transportation problem in forestry which arises when transporting logs from forest sites to customers such as sawmills and pulp and paper mills. In this problem each route has to satisfy a number of constraints concerning time windows, truck capacity, timetable of the driver, lunch breaks, et cetera, they use three solution methods based on the column generation principle, together with a pool strategy which allows them to deal with the feasible routes outside the restricted master problem.

2 Exact solution of the Problem

In order to obtain the exact solution, on one hand, it is required to derive the mathematical model and then generate instances according to the decision variables and relevant parameters. On the other hand, we use Gurobi, a computer software of mathematical optimization solver, as the exact solver.

2.1 Mathematical model

Variable definition $I = \{1, 2, ..., M\}$ is the index set of all m compartments, $J = \{1, 2, ..., N\}$ is the index set of all n destinations and $K = \{1, 2, ..., Q\}$ is the index set of all q products.
x_{ijk}: quantity of product k for destination j to be loaded in compartment i. $i \in I, j \in J, k \in K$
C_i: the capacity of compartment i, $i \in I$
p_k: the size of the package of the product k, $k \in K$
d_{jk}: demand rate of product k at destination j, $j \in J, k \in K$
t: replenishment time should be the minimum loading in all compartments for all products according to a certain destination.

$$t = \min_{j \in J k \in K} \{ \frac{\sum_{i=1}^{M} x_{ijk}}{d_{jk}} \} \tag{1}$$

Objective function The objective is how to load the compartments of the vehicle so that the replenishment time is maximized. We can convert the Max-Min problem into a Max problem with an additional constrain:

$$max \quad t \tag{2}$$

subject to

$$\sum_{j=1}^{N} \sum_{k=1}^{Q} p_k x_{ijk} \le C_i \quad i \in I \tag{3}$$

$$t d_{jk} \le \sum_{i=1}^{M} x_{ijk} \quad j \in J, k \in K \tag{4}$$

$$x_{ijk} \ge 0, \; x_{ijk} \in \mathbb{N}^+ \quad i \in I, j \in J, k \in K \tag{5}$$

$$t \ge 0 \tag{6}$$

2.2 Instance generation

To generate reasonable instances, fisrtly, the number of compartments m, the number of destinations n and the number of products q are fixed and can be varied manually. They are assumed to be constant. Secondly, C_i is generated uniformly and randomly with constrains $C_L \le C_i < C_H$, where C_L and C_H are lower limit and higher limit of the capacity of compartments, they are both assumed constant and can be modified manually. Furthermore, p_k is generated in the same way, with constrains $p_L \le p_k < p_H$, where p_L and p_H are lower limit and higher limit of the size of the packages. The demand of product k at destination j is denoted as d_{j_k} and for each destination j, we generate a local limit d_{j_L} and d_{j_H} randomly, with constrains $0 \le d_{j_L} < d_L$ and $d_L + 1 \le d_{j_H} < d_H$, then we generate d_{j_k} randomly with constrains $d_{j_L} \le d_{j_k} < d_{j_H}$. Note that all the parameters will be integers.

2.3 Results based on various inputs

We observe the various results based on modification of values in terms of compartment capacities, product sizes and demands.

Decreasing of compartment capacity In table 1, the values of objective function according to decreasing values of compartment capacity are presented and it is intuitive that compartment capacity plays an important role in the determination of objective function result since with a less amount of capacity, the number of loading products will be less accordingly and thus, in order to satisfy the demand, we need more deliveries, in other words, higher replenishment time.

Table 1: Objective function results based on different compartment capacity

Capacity of 3 comparments	Objective function result
810, 843, 821	1.25
405, 510, 410	0.591
270, 281, 273	0.391

Distinct product size The original product sizes are uniformly generated while in table 2, the objectibe function result of products with distinct sizes is shown. The second result is smaller, which means in order to have a higher replenishment time, it is suggested to avoid products that are extremely diverse in size.

Table 2: Objective function results based on distinct product size

Size of 3 products	Objective function result
20, 19, 21	1.25
1, 19, 40	1.118

Distinct demand According to table 3 , demand variation does not contribute a lot when it comes to the objectibe function result. Logically speaking, it is easier to satisfy the demand for product with smaller requirement by loading less amount of it, which leaves more room for others with larger demand. Consequently, the demands are fulfilled in an average way regardless the variation among products.

Table 3: Objective function results based on distinctdemand

Demand of 3 products for 2 destinations	Objective function result
16, 22, 23 and 8, 17, 12	1.25
1, 22, 38 and 1, 17, 19	1.211

3 Heuristic methods

Based on the review of literatures and the analysis during the development, we derive two key points that can be utilized to develop heuristic methods:

- In order to maximize the replenishment time, we need to load products as much as possible to not only satisfy but also exceed the demands and thus, have less deliveries. It implies that we can find a feasible solution which try to fill up all the compartments.
- When we have a result of minimum value for a certain destination among all the replenishment time, the requirements for other destinations are also respected with higher demands. It implies that we only need to try to modify the minimum time to improve the lower limit of the requirements.

In this section we introduce three heuristic algorithms and make comparisons with exact solution.

3.1 Partial Dynamic Programming

3.1.1 Introduction

The first method we have developed is composed by two parts: 1. find a possible time that can be used to derive a feasible solution; 2. according to the value of time, try to obtain a feasible solution and at the end of each iteration, update the value of time.

Binary search algorithm, finding a value of time Through this sub-algorithm, it is allowed to minimize the interval of uncertainty and further maximize the replenishment time t in a fast manner. At the begining, it is required to define the bounds, in which the lower bound t_{lb} is initialized as 0, and since the maximum loading will be reached when all the rooms are filled up, so that we can consider all the compartments as a whole and therefore upper bound with relaxation is obtained by:

$$t_{ub} = \frac{\sum_{i=1}^{M} C_i}{\sum_{j=1}^{N} \sum_{k=1}^{Q} p_k d_{jk}} \tag{7}$$

Moreover, define $t_l = (t_{lb} + t_{ub})/2$, which is used to derive a feasible solution in the following sub-algorithm. At the end of each iteration, the knowledge of feasiblility will be provided. If the solution is a feasible one, which means we may still have improvement for current time t_l, update the lower bound with value of t_l. Otherwise, if the solution is infeasible, which means there's no more feasible solutions when the value of time is higher than current time, update the upper bound with t_l. In addition, update t_l accordingly. The algorithm does not stop until the dfiierence between bounds are smaller than a threshold of 0.03 and then output the last feasible solution.

Dynamic programming algorithm, find a feasible solution In practice, it does not matter how to load a certain product in compartments for a given destination as long as its demands has been satisfied because a destination can retrieve the same product either from, for example compartment A or compartment B, which means we are able to only take into acoount the loading stradegy of overall products (demand) for all destinations. As a consequence, we can derive a total product requirement based on the time provided by the previous algorithm: $D_k = \sum_{j=1}^{N} d_{jk} t_l$. Then the problem becomes that try to find a way to load all these products in compartments. Despite the fact that there're corrlations among compartments, we can still deal with comparments one by one instead of analyzing all compartments together. Therefore, for a certain compartment, we can grab products from the total quantity to try to fill it up, which resembles the classical 0-1 knapsack problem with item value being equal to item size, that can be solved by dynamic programming. And the recursive formula is:

$$value_{i,j} = max[value_{i-1,j}, value_{i-1,j-VS_{i,0}} + VS_{i,1}], i \in \{1, ..., D\}, j \in \{1, ..., C\} \tag{8}$$

in which D is the total amount of all demands: $D = \sum_{k=1}^{Q} D_k$, C is the current compartment capacity, $value_{i,j}$ is the total value (total size of loaded products) at stage i with occupied volume of j and VS stores the size (index 0) and value (index 1) at stage i.

Now, for a compartment, we are able to find the optimal way to load products to fill the void as much as possible and we can perform the same procedure for the rest products and the next compartment. Finally, by exhausting all the compartments, if we are able to load all the products, it means there must be a feasible solution under the current condition of time. And with the output of result and feasiblity, we are able to update the time bounds interation by iteration. In conclusion, the algorithm follows the flowchart in picture 1.

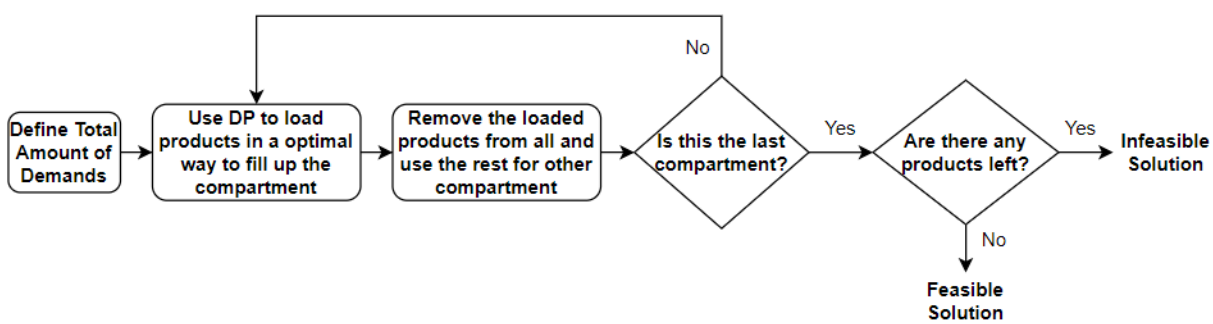

Figure 1: Flowchart for dynamic programming

3.1.2 Comparison with exact solution

In order to evaluate the performance of this heuristic method, the gap in terms of results and computational time are taken into account.

On one hand, we have evaluated the gap with different instances generated by 200 random seeds under the same resonable and relatively normal dimensional parameters[1]. According to the histogram presented in Fig. 2, we get gaps distributing from 0% to 1.6% except for one and most cases are located around 0% and 0.7%, which indicates that the

[1]3 compartments, 3 products, 2 destinations, $C_L = 700$, $C_H = 900$, $p_L = 15$, $p_H = 25$, $d_L = 10$, $d_H = 30$ and the gap figures for all the heuristic methods are evaluated with this configuration.

Partial Dynamic Programming method works substantially well from a statistical point of view. Although we cannot eliminate the correlation among compartments and only get a sub-optimal solution for a certain compartments, the performance is more than acceptable.

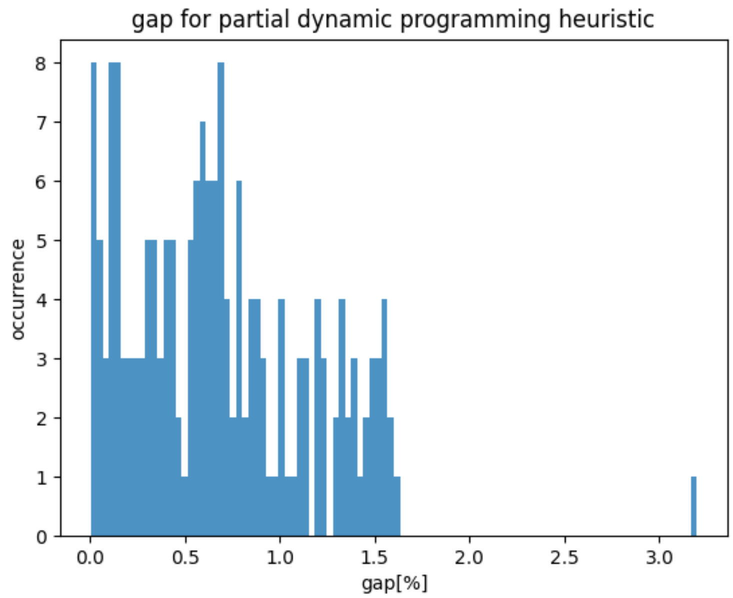

Figure 2: histogram of gap between heuristic (Partial Dynamic Programming) result and exact result for different random seeds

On the other hand, we also have considered the time consumption[2] as the dimension of parameters[3] increases and the result is in Table 4. In terms of gap, the excellent performance still holds and it has nothing to do with the complexity of the model. On contrary, the computational time increases proportionally and dramatically as the dimension of parameters increases and the values are generally unacceptable, which means it is not affordable to apply this method when we have a large scale of parameters. From a mathematical perspective, the overall time complexity is around O($n^3 \log n$), which indicates that dynamic programming is not a good way to solve the problem in terms of time consumption.

Table 4: Compression of solution: Exact solver v.s Partial Dynamic Programming

	Execution time		
Index	Gurobi[s]	PartialDP[s]	Gap[%]
1	0.048	0.425	1.28
2	0.066	4.78	0.62
3	0.052	31.66	1.22
4	0.5	116.73	0.76
5	0.627	266.76	0.63

3.2 Simulation with Fair-Supply Strategy

3.2.1 Introduction

Since we are dealing with a loading problem, it is also possible to derive a heuristic method to simulate the loading procedure based on a certain stradegy. In particular, the goal of this problem is to increase the minimum replenishment

[2] in some cases, it's not possible for Gurobi to derive the exact solution within seconds and the presented time are for exact solution with a gap smaller than 0.1%.

[3] changed parameters: number of products: [3, 6, 9, 12, 15], number of destinations: [2, 4, 6, 8, 10], $C_L = [700, 2800, 6300, 11200, 17500]$, $C_H = [900, 3600, 8100, 14400, 22500]$ and the rest doesn't change.

time as much as possible, in other words, increase the minimum supply for the corresponding destination. Meanwhile, it is also not desirable that other non-minimum supplies are so many that occupy much more spaces and leave less for the minimum. To sum up. a stradegy can be seen as a good one if we can balance the supplies for all destinations and therefore, we have developed a heuristic method based on this stradegy to simulate the process. The algorithm is divided into 2 stages:

First stage, adding one We traverse all the combination of loading following the order of destination, compartment and product. In particular, For a given product required by certain destination, we add a quantity of one to the given compartment and repeat this procedure iteration by iteration until we satisfy the most basic demand for all destination in one period (t=1) or all the compartments are filled up. In addition, during this process, if a demand of a specific product required by a specific destination is met, for the following iteration, we will skip the combination of this product and destination. As a result, we give all the compartments initial occupancies satisfying the fundamental demand in an average way.

Second stage, adding more Instead of adding one item iteratively, it is more resaonable to add more items based on various demands since a higer demand requires a higher supply to increase the related replenishment time. Therefore, in this stage, we still follow the traversal process but with a increment of a product to a compartment varying according to its the demand. The increment can be determined by the ratio of corresponding demand divided by the minimum demand of all products for a certain destination, e.g. we use the minimum as a base, a Carrefour store needs 2 kg apples, 6 kg bananas and 10 kg oranges, then the increment (supply) should be 2/2=1 kg apples, 6/2=3 kg bananas and 10/2=5 kg oranges. In addition, if a compartment cannot hold a certain product with a specifc increment anymore, we decrease the increment by one until the increment reaches one, which means this compartment is full. In the previous stage, we satisfy the demand in an average way while in this stage, we increase the supply (demand) in an average manner.

In the end, we stop the simulation when all compartments are full and by averaging the product demand, we balance not only the distribution of products in compartments but also all the supplies to avoid penalizing a certain destination.

3.2.2 Comparison with exact solution

The comparison has been done in the same way with respect to the previous method.

In Fig. 3, obviously, the performance in terms of gap is worse than the previous method but with relatively acceptable results ranging from 0% to 6%. Moreover, there exist few gaps with value greater than 10% and up to 20% due to some instances with large variation in demands, which means this method doesn't perform well versatilely.

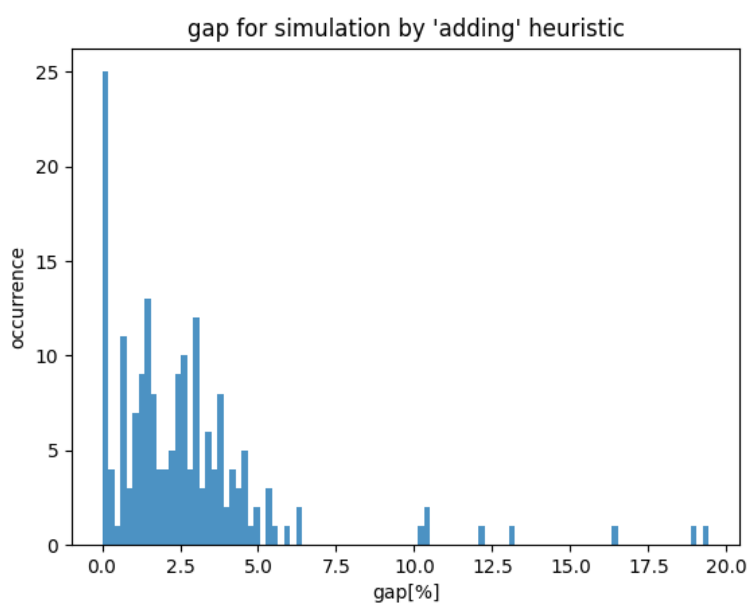

Figure 3: histogram of gap between heuristic (Simulation with Fair-Supply Strategy) result and exact result for different random seeds

Being different from the performance for gaps, Table 5 demonstrates a huge improvement when it comes to the computational time. As the increasing of complexity, the time comsumption doesn't increase with a large amount but it cannot be ignored that the gap is still an issue.

Table 5: Compression of solution: Exact solver v.s Partial Dynamic Programming

	Execution time		
Index	Gurobi[s]	PartialDP[s]	Gap[%]
1	0.048	0.001	5.45
2	0.066	0.025	2.12
3	0.052	0.353	7.41
4	0.5	1.82	1.72
5	0.627	6.67	4.88

3.3 Simulation with greedy algorithm

Similar to PartialDP method, we have also used half-interval search algorithm to obtain the optimal value of t, and similar to the second heuristic method, this method computes the optimal solution through simulation iterations as well, but combined with greedy algorithm.

3.3.1 Introduction

3.3.2 Comparison with exact solution

The same comparisons have been proceeded and results are presented in Fig.4 and Table.6. In terms of gaps, there is a huge amount of gaps with value of 0%, which means it is possible to reach a extremely precise result with respect to the exact solution, but the distribution of other gaps is relatively dispersive indicating a unstable performance. On top of that, the time comsumption is so small that can be ignored, even for a high compexity, which results in a hegemony in heuristic method choice if the computational time is the paramount consideration.

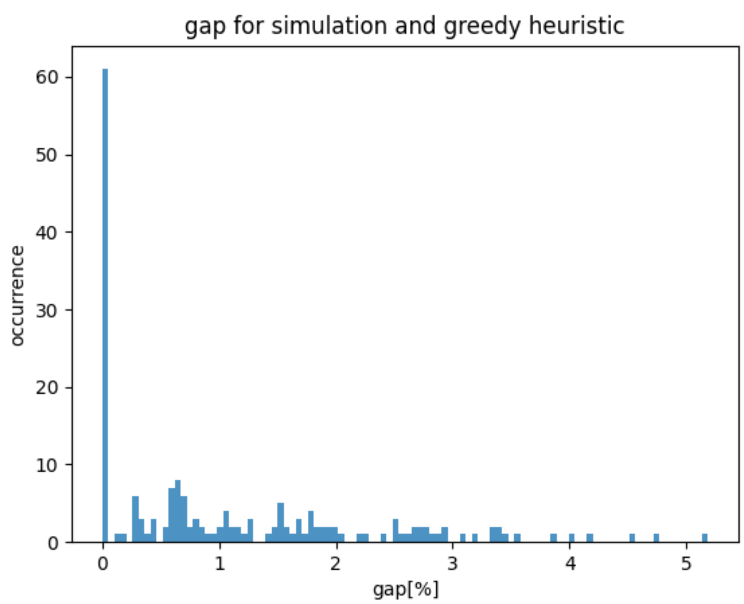

Figure 4: histogram of gap between heuristic (Simulation with greedy algorithm) result and exact result for different random seeds

Table 6: Compression of solution: Exact solver v.s Simulation with greedy algorithm

	Execution time		
Index	Gurobi[s]	greedy[s]	Gap[%]
1	0.048	0.001	0.00
2	0.066	0.002	0.00
3	0.052	0.002	0.97
4	0.498	0.004	0.41
5	0.627	0.004	0.35

3.4 Summary of heuristic methods

The ultimate goal is to find a fast and good heurist method to solve the truck loading problem. Besides the comparison between certain heuristic mtehod and the exact solver, the worst case performance is indeed an important indicator when it comes to the evaluation of heuristic methods. Futhermore, a general comparison taking into acount all the factors, among all the heuristic methods can be conducted to draw a conclusion.

3.4.1 Worst case analysis

To begin with, four relatively extreme scenarios with different parameters are considered. All the results in the following are derived based on instances generated by 100 different random seeds but with certain parameters being fixed. In addition, two kinds of results are presented: 1. number of times when a certain heuristic method gives the highest objective function result among all three. Note that there might be more than one heuristic methods providing same highest value; 2. average gap with respect to exact solver.

Distinct comparment capacity

Distinct product size

Distinct demand

Relatively small compartment capacity

3.4.2 Comparsion among heuristic methods

3.5 Conclusion

References

[1] Ümit Yüceer & Arif Özakça. (2010). A truck loading problem. Computers & Industrial Engineering 58, 766–773.

[2] A. N. Balaji, J. Mukund Nilakantan, Izabela Nielsen, N. Jawahar & S. G. Ponnambalam (2019). Solving fixed charge transportation problem with truck load constraint using metaheuristics. Annals of Operations Research 273, 207–236.

[3] Maria Teresa Alonso, Ramon Alvarez-Valdes, Francisco Parreño & Jose Manuel Tamarit. (2016). Algorithms for Pallet Building and Truck Loading in an Interdepot Transportation Problem. Mathematical Problems in Engineering, 2016, Article ID 3264214.

[4] Myrna Palmgren (2005). Optimal Truck Scheduling: Mathematical Modeling and Solution by the Column Generation Principle. Linköping University.