3D Truck Loading problem

Ayoub Sbai

April 29, 2023

Abstract

Operations research is a scientific discipline that uses several techniques from fields such as mathematical modeling, simulation, optimization theory, computer science and other fields to solve complex decision making problems concerning processes and operations. In that telling, operations research is linked to decision theory and industrial engineering. Operations research is also known as management science.

The ROADEF/Euro challenge 2022 is a supply chain decision problem. Supply chain decision problems are one of the most famous problems in operations research. They require analytical methods and techniques from a plethora of fields (Queuing theory, Graph theory, Combinatorial Optimization, Discrete modeling...etc)

1 Problem Definition and Statement

How to maximize 3D truck loading efficiency (thus minimizing the number of trucks and inventory cost due to early deliveries) in order to deliver all items using 6000 trucks or less to 40 plants in 17 countries

Related fields:
-Information theory (Conformational entropy, Kolmogorov complexity..etc)
-Extremal combinatorics
-Number theory (Perrin numbers, Padovan sequence, plastic number...etc)
-Graph theory (circle packing representation, independent set..etc)
-Statistical physics (Gibbs entropy)
-Topology
-Measure theory
-Crystallography

2 Problem illustration

2.1 Problem taxonomy

The 3d truck loading problem is an example of packing problem. Packing problems are concerned with packing objects as densely as possible within a single container or packing maximum number of objects in a container as to minimize the number of containers used. There are a lot of variants of packing problems depending on the topology of the objects and the containers.
Packing problems are either geometry-specific or otherwise. The latter case is usually dealt with platonically via abstract data structures. In the case of geometry taken into account, there are two parameters that distinguish between different problems (other than topology or supply collection); boundedness (compact space, Euclidean space..etc) and dimension of the space (3D, 2D..etc)

Complexity-wise, packing problems are NP-hard; meaning that there exists a polynomial-time reduction NP problems of the same nature to packing problems ie that solutions to packing problems can be used to solve NP problems (of the same nature) in polynomial time. However, the fly in the ointment is that polynomial-time algorithms for NP hard problems such as packing problems are not

yet proved to exist. In fact, this is equivalent to stating the famous conjecture P = NP.

Packing problems have recovering problems as their dual optimization problems. This will prove useful later in the problem-solving phase.

2.1.1 Packing density

Packing density is defined as the fraction of the amount of space filled by objects to the space of the container. Packing density is an important notion in crystallography (known as compacity or atomic packing factor) that we usually choose to maximize (Kepler conjecture, Ulam packing conjecture..etc). The formulation of the packing density is measure-specific (continuous,discrete), space-specific (compact, Euclidean space..etc). For our convenience, trucks are compact (bounded and closed) finite-dimensional (3D) measurable spaces X. Let K1,K2,...,Kn be non-overlapping measurable subsets (stacks in our case). The packing density of the stacks packing collection is defined as:

$$\eta = \frac{\sum_{i=1}^{n} \mu(K_i)}{\mu(X)} \tag{1}$$

where μ is the measure defined on our compact space (here taken to be the volume).

Dual to the notion of packing density is the notion of redundant void (redundant because it's not weight-constrained and thus unnecessary void between stacks that need to be exploited by adding more potential stacks in the truck):

$$\text{RV} = 1 - \eta = \frac{\mu(X) - \sum_{i=1}^{n} \mu(K_i)}{\mu(X)} \tag{2}$$

Maximizing the packing density (effect) is the inverse problem of the Gibbs entropy minimization (cause). This approach is recommended because there is no explicit geometry-specific formulation of Gibbs entropy. Inverse problems are usually ill-posed (often violating Hadamard's third criterion for well-posed problems; stability/continuity of the solutions with respect to given parameters) which makes it impossible to infer the causes sometimes. However, Gibbs entropy (cause) is just a representative model that fits our goal (a sort of abstract cause, not a physical cause). Moreover, we are not interested in calculating this Gibbs entropy perse from packing density inverse problem. In other words, this is not an either/or situation between forward and inverse problems as both notions are, in our case, completely replaceable.

2.1.2 Gibbs entropy vs Conformational entropy

The Gibbs entropy refers to the degree of disorder present in a system, which is naturally high in the case of random sampling. This entropy is order-specific and unequally distributed, meaning that some states are more favored than others, as dictated by the second law of thermodynamics. In this context, Gibbs entropy is the measure of weight-unconstrained void that exists between stacks. This void is unnecessary and actually contributes to the high entropy of the system. The analysis assumes that loading constraints have been met, but aims to maximize loading possibilities, or the number of potential future items/stacks to be loaded. This is known as the primal problem of entropy's dual problem. Specifically, the potential loading problem is denoted by K=3, with pre-defined loading problems represented by K=1,2 and the post-loading problem by K=3. This problem can be expressed as the maximization of abs(card(pre-SIs)-card(post-SIs)) using a truck-based approach. The Gibbs entropy is considered the dual of the geometry/packing density/compacity, which is the primal problem. Combinatorial/Conformational entropy, on the other hand, refers to the number of ways in which a system, such as a molecule or any thermodynamic system, can be rearranged. Combinatorial entropy is used to enumerate all exhaustive possibilities of 5 degrees of freedom: translation sx,sy,sz (continuously infinite, discretely finite), horizontal rotation, and forced orientation SOs (which reduces from 4 to 2

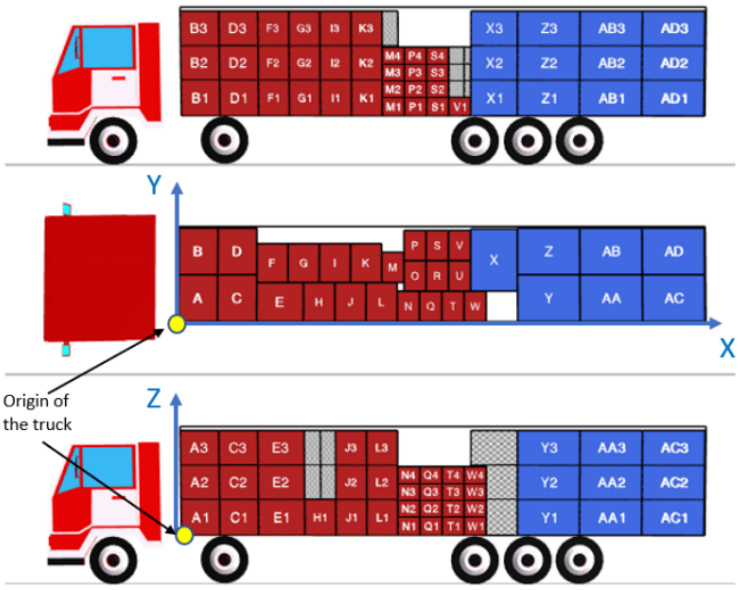

Figure 1: Truck

due to structural symmetry). This includes lengthwise and widthwise rearrangements, which leads to individual void and, eventually, collective void through recursion.

3 Challenge description

Renault has a complex supply chain problem, with more than 40 factories in 17 countries and 1,500 suppliers, where 6,000 trucks deliver parts every week. To minimize the cost of transport and inventory, it is crucial to maximize the number of items per truck ratio and deliver the items to the factories as late as possible (at least, that being the case with Renault. It usually holds true in general as well when the inventory turnover ratio is low), while respecting the delivery time windows of 1 to 5 days. However, early deliveries can be made, but this incurs inventory costs for factories. The scheduled trucks are hired to deliver the items to the factories, but if their number is insufficient (for whatever reason, intentionally left equivocal), additional/extra trucks costing more can be called upon. The problem is to organize the stacks of items in the three-dimensional trucks to minimize the total cost of transport and inventory, with constraints on the dimensions and weight of the stacks and respecting the delivery time windows.

The three-dimensional trucks contain stacks packed on the two-dimensional ground. Each stack contains items that are packed on top of each other.

3.1 Items

An item i is characterized by its spatial dimensions length, width and height(IL_i, IW_i, IH_i) , its weight IM_i and its stackability code IS_i , the latter ensures that elements with the same code share the same stack, length, width, plant dock, supplier, supplier dock. Each item has a maximum stackability ISM_i. An item i can have a forced orientation IO_i and it must be delivered to a factory IP_i within a time interval between its earliest arrival time $IDE_i$and its latest arrival time IDL_i. An item i has a unit inventory cost.

3.2 Trucks

A planned truck t is characterized by its length/width/height dimensions (TL_t, TW_t, TH_t), its maximum authorized loading weight TM_t^m , its transport cost TC_t. The arrival of a truck at the factory TP_t is scheduled at the arrival time TDA_t,It can recover the set of products TR_t from a set of suppliers TU_t. In case of lack of scheduled trucks, additional trucks can be called from the carrier. An additional truck $\hat{t_j}$ has the same characteristics as a scheduled truck, except for the transport cost. A truck is associated with a "stack with several ports" indicator TF_t If (TF_t = yes), a stack packed in truck t can include items associated with different factory ports (consecutive in iterms of loading/unloading). If (TF_t = no), all stacks packed in truck t must have a single factory port among its items.

3.3 Stacks

A stack contains items, stacked on top of each other, and loaded onto a truck. It is characterized by its length/width/height dimensions (sl_s, sw_s, sh_s) and its weight sm_s. The stack is characterized by the coordinates of its point of origin (sx_s^0, sy_s^0, sz_s^0) and of its points (sx_s^e, sy_s^e, sz_s^e). Stacks can be rotated horizontally either lengthwise or widthwise. Stack's forced orientation is dictated by the forced orientation of one of its items.

3.4 Objective Function

The objective is to simultaneously minimize the total cost of transport and inventory. The transport cost is the total cost of the planned and additional trucks. This is a fixed cost that depends only on the number of trucks, not the number of items loaded. The storage cost is a variable cost due to early deliveries of items, calculated on the interval between the time of arrival of the items at the factory and the latest time of arrival of these items. The storage cost does not include a fixed part.

3.5 Constraints

the constraints relate primarily to constraints of the articles which guarantee the compatibility between the articles and the trucks, that is to say which articles can be loaded in which trucks, in the second place on the constraints of stack which control the composition of the stacks, mainly which items can be packed in the same stack third on placement constraints related to the placement of the stacks in the trucks, in particular the order of placement of the stacks according to the docks of the suppliers and the factory and finally on weight constraints which impose the maximum authorized weights of the batteries in the truck.

4 Mathematical model

In the EURO/ROADEF 2022 Challenge document, several parameters, decision variables and constraints are described in detail. While the mathematical model is not unique, it's convenient to adopt the model proposed by the authors of the challenge (Alain Nguyen, Mohamed-Amine Khatouf, Christian Serrano; Challenge EURO/ROADEF 2022 Truck Loading)

4.1 Parameters

Below are the parameters that will appear in the mathematical modeling:
$\tilde{I}$: set of items i
IL_i, IW_i, IH_i, IM_i : length/width/height/weight of item i
ISM_i : maximum storage capacity of item i
IO_i : forced orientation of item i
IDE_i, IDL_i : earliest and latest arrival time of item i
IP_i : destination plant of item i
IG_i : factory port of item i
IU_i : supplier of item i
IK_i : vendor port of item i
IC_i : inventory cost of item i
TL_t, TW_t, TH_t, TM_t^m : length/width/height/maximum truck loading weight t
TP_t : destination plant of truck t
TDA_t : arrival time of truck t at T Pt
$TMM_t\gamma$: maximum total weight of all packed items above the bottom item associated with product γ in any stack of truck t
TF_t : "stack with multiple ports" flag for truck t
TEM_t: maximum density of the piles in the truck t
TC_t : truck cost t
TE_t: supplier loading order for truck t
TGE_{pt}: port loading order of factory p for truck t
TKE_{ut}: loading order from ports of supplier u for truck t.
TGE_{pt}: port loading order of factory p for truck t.
α_T: the transport cost coefficient
α_I: the inventory cost coefficient

4.2 Decision Variables

The decision variables represent the choices we have to make in this problem. In this problem, the decision variables are:
$\tilde{T}$:the set of trucks t
$\tilde{TS_t}$:the set of batteries loaded into a truck t
$\tilde{SI_s}$: the set of items wrapped in the stack s
sl_s, sw_s, sh_s, sm_s:length/width/height/stack weight s
$\tilde{SG_s}$:the set of factory ports in stack s.
ida_i: time of arrival of item i
sx_s^o, sy_s^o, sz_s^o: the coordinates of the origin point of the stack s
sx_s^e, sy_s^e, sz_s^e: the coordinates of the end point of the stack s
su_s :the stack provider s

4.3 Objective function (Fitness)

Our goal is to minimize the cost of transportation and inventory, hence the following objective function:

$$Min \ \ \alpha^T \sum_{t \in \tilde{T}} TC_t + \alpha^I \sum_{i \in \tilde{I}} IC_i * (IDL_i - ida_i) \tag{3}$$

The difference $(IDL_i - ida_i)$ is calculated in days.

4.4 Constraints

- All items must be packed and loaded onto trucks: To ensure proper organization and safety of items during transport, it is important to stack and load them properly onto trucks. This is done by sorting and grouping items by size:

$$\sum_{s \in \tilde{TS}_t} \sum_{i \in \tilde{I}} 1_{i \in \tilde{SI}_s} (IL_i * IW_i * IH_i) \leq \sum_{s \in \tilde{TS}_t} (sl_s * sw_s * sh_s) \tag{4}$$

- The product arrives at the factory during the time window:

$$IDE_i \leq TDA_t \leq IDL_i \tag{5}$$

- All items packed in a stack must share the same vendor, factory, overlay code, and vendor port:

$$\forall (i,j) \in \tilde{SI}_s \quad IU_i = IU_j, IP_i = IP_j, IK_i = IK_j, ISM_i = ISM_j \tag{6}$$

- For any stack s loaded in truck t and if (TFt = no), all items in stack s must share the same factory port:

$$\forall s \in \tilde{TS}_t \;\; \forall (i,j) \in \tilde{SI}_s \;\; TF_t = 0; \quad IG_i = IG_j \tag{7}$$

- For any stack s loaded in truck t, if (TFt = yes), s can contain items with 2 factory ports with consecutive load orders. Provided that for each overlay code SC present among the stacks of a t truck, a single stack with the SC overlay code can contain items with 2 factory ports:

$$\exists! s^* \in \tilde{TS}_t \;\; \exists (i,j) \in \tilde{SI}_s \;\; TF_t = 1; IG_i \neq IG_j, TGE_{pt}[IG_i] - TGE_{pt}[IG_j] = 1 \tag{8}$$

- If an item i in a stack s has an enforced orientation (IOi), then all items in stack s must share the same orientation:

$$\forall i \in \tilde{SI}_s \;\; so_s = IO_i \tag{9}$$

- For any stack s loaded into truck t, the total weight of the items packed above the bottom item associated with the product must not exceed the maximum allowable weight:

$$\sum_{i \in \tilde{SI}_s and i \neq bottom_item} IM_i \leq TMM_t \gamma \tag{10}$$

- The number of items packed in a stack s must not exceed the smallest "maximum storage capacity ISM_i":

$$|\tilde{SI}_s| \leq minISM_i \tag{11}$$

- The density of a stack s must not exceed the maximum stack density defined for the truck t in which the stack s is loaded:

$$\frac{sm_s}{sl_s * sw_s} \leq TEM_t \tag{12}$$

- The location of a stack s in a truck t must not exceed the dimensions of the truck:

$$\forall t \in \tilde{T} \;\; \forall s \in \tilde{TS}_t, \;\; sx_s^e \leq TL_t \;\; sy_s^e \leq TW_t \;\; sz_s^e \leq TH_t \tag{13}$$

- Stacks should be placed in an increasing manner from the front to the back of the truck, (*) according to the pick-up order of the suppliers, and among the stacks of the same supplier, the stacks should be placed in an increasing manner (**) according to the loading order of the supplier's port, and among stacks with the same supplier and supplier's port, the stacks should be placed in an ascending manner (***) according to the loading order of the port of the 'factory.

$$\forall t \in \widetilde{T}, \forall s_1 \in \widetilde{TS_t}, \forall s_2 \in \widetilde{TS_t}$$

$$(*) \text{ T E}_{u_{s_1}} < TE_{u_{s_2}} \implies sx_{s1}^o \leq sx_{s2}^o$$

$$(**) \left(su_{s_1} = su_{s_2} \, , TKE_{k_{s_1}} < TKE_{k_{s_2}}\right) \implies sx_{s1}^o \leq sx_{s2}^o$$

$$(***) \left(su_{s_1} = su_{s_2} \, , TKE_{k_{s_1}} = TKE_{k_{s_2}}, TGE_{g_{s_1}} < TGE_{g_{s_2}}, \forall g_{s_1} \in \widetilde{SG_{s_1}}, \forall g_{s_2} \in \widetilde{SG_{s_2}}\right)$$
$$\implies sx_{s1}^o \leq sx_{s2}^o \tag{14}$$

- The weights of stacks loaded into a truck t must not exceed the maximum loading weight of the truck:

$$\forall t \in \tilde{T} \;\; tm_t \leq TM_t^m \quad with \;\; tm_t = \sum_{s \in S_t} sm_s \tag{15}$$

5 Decision Variables tree/forest: Systemic Analysis

As outlined in the official document (Alain Nguyen, Mohamed-Amine Khatouf, Christian Serrano; Challenge EURO/ROADEF 2022 Truck Loading). Theses are the chosen decision variables forour truck loading problem:

$\widetilde{T}$: set of used trucks t (planned and extra trucks
$\widetilde{TS_s}$: set of the stacks packed into truck t
$\widetilde{TI_s}$: set of the items of stack s
$\widetilde{TG_s}$: set of the plant docks of stack s

sl_s, sw_s, sh_s, sm_s: : length, width, height, weight of stack s
sx_s^0, sy_s^0, sz_s^0:coordinates of the origin point of stack s
sx_s^e, sy_s^e, sz_s^e:coordinates of the extremity point of stack s
so_s: orientation of the stack s
su_s: supplier of the stack s
sk_s: supplier dock of the stack s
st_s: truck of the stack s
ida_i: arrival time of item i

However, it's noteworthy to mention one of the most important theorems about mathematical modeling; No Free Lunch theorem: while there is no universally best model, there are only models that are better or worse than other models depending on the scope and the context. That said, a model is never unique but always invariant (and it must be!), that includes the decision variables as well; they are never unique but whatever set of decision variables one chooses, the object being modeled is literally the same. Taking NFL theorem into account, there are certainly decision variables that are better than others (in terms of combinatorial explosion, time-complexity, space-complexity, compatibility, validity...etc).
In that telling, it's best to try to condense all the aforementioned decision variables into an overarching, all-encompassing smaller set of decision variables that encode all the information about the former ones. In our case, Boolean variables are the ones to go to. First, the Boolean decision variables are presented. Second, a decision tree/forest analysis is presented that shows how the former decision variables can be efficiently encoded and/or abstracted in the new ones without toppling over the abstraction-coupling trade-off.
- The Boolean decision variables are as follow:
y_t: 1 (use the truck t), 0 (otherwise)
x_{is}: 1 (insert item i to stack s), 0 (otherwise)
x_{st}: 1 (insert stack s to truck t), 0 (otherwise)
The last two variables can be combined into one: x_{it} such that $x_{it} = x_{is}x_{st}$
- The decision variables tree:

$\widetilde{T} = \sum_{t=1}^{m} \mathrm{y}_t$
$\widetilde{TS_s} = \sum_{s=1}^{d} \mathrm{x}_{st}$
$\widetilde{TI_s} = \sum_{i=1}^{n} \mathrm{x}_{is}$
$st_s = y_t x_{st}$
$\mathrm{Prob}(ida_i = TDA_t) = \mathrm{Prob}(x_{it}y_t) = \frac{TDA_t - IDE_i}{IDL_i - IDE_i}$
$Prob(\mathrm{sl}_s \leq \mathrm{TL}_t, sw_s \leq \mathrm{TW}_t, sh_s \leq \mathrm{TH}_t, sm_s \leq \mathrm{TM}_t^m, su_s \in \widetilde{TU_t}, sk_s \in \widetilde{TK_{ut}}) = \mathrm{Prob}(y_t = 1)$

such that $sl_s = sx_s^e - sx_s^0$, $sw_s = sy_s^e - sy_s^0$, $sh_s = sz_s^e - sz_s^0$
$\widetilde{TG_s} = x_{st}y_t$ $(\mathrm{TF}_t = 0)$,$\widetilde{TG_s} \leq 2\mathrm{x}_{st}y_t$ $(\mathrm{TF}_t = 1)$

6 Problem solving methodology: K-Complex Meta-Opt

K-Complex MetaOpt:Our problems is divided into 3 fundamental subproblems that are interconnected between them because the problem proposed is complex in the sense of emergence and non-linearity between the subproblems (min function is non-linear) that exhibit feedback loops. Our first subproblem is the truck usage (K=1), second subproblem is item/stack/truck insertion (K=2) and third subproblem is minimization of Gibbs entropy for an optimal arrangement of items in the truck (its dual is the maximization of compacity/packing density of the items; eliminating the unnecessary void between items).
Complexity arises from the fact that subproblem (K=1) is the dual optimization problem of the subprobem (K=2).

The subproblems K = 1,3 are spatial in nature because they are instances of packing and recovering problems.
The subproblem K = 2 is temporal in nature based on efficient matching (in set-theoretic terms, a surjective map) between two sets of objects (stacks or items and trucks)

7 Divide-And-Conquer strategy

Given the ginormous amount of parameters, variables and constraints within the problem. It's of a good practice to follow a divide-and-conquer strategy by solving simpler, smaller and less complex (taking the best case scenario) subproblem versions of the big problem just to have an insight on the nature of the problem. This approach assumes that the subproblems are independent and of a non-complex nature (matroids).
This approach is bottom-up; it builds from simpler subproblems (Knapsack problem, bin packing problem, VM packing problem, Cutting stock problem, Assignment problem...etc) in order to recover the truck loading problem (in the case where there is no emergence arising out of scaling up the subproblems) or at least give an insight about the nature of the truck loading problem and proceed afterwards to solve the problem top-down.

7.1 Divide-And-Conquer the objective function

In our case, the problem is divided into two main subproblems; the first is minimizing the total cost TCt by minimizing the number of trucks used (Bin packing problem) and the second is minimizing the variable inventory cost tied to the arrival time of item deliveries (Selective matching/Assignment problem). Divide-and-Conquer technique will relax one problem at a time and solve the other.

Concerning the first side of the objective function, the subproblem in question is the famous bin packing problem (offline vs online)l, VM packing problem (VM is the stack structure in our case; nuclear mass analogy via space-sharing)..etc. These problems are usually solved by Fit algorithms like: First-Fit, First-Fit decreasing algorithm, Best-Fit algorithm, Next-Fit algorithm, Next-k-Fit algorithm, Worst-Fit algorithm....etc

With the exception of next-fit algorithm, all other algorithms yield a solution in O(nlogn) complexity where n is the number of items to be packed; the logarithm comes from the operation of pruning resulting from geometric and/or Pascal probability distributions of the algorithm.

As for the second side of the objective function, the subproblem in question is the matching/assigment problem (thus the use of bipartite graphs and independent edge sets/matching) where latest times are favored over earlier times as to mitigate inventory cost of the items. These problems can be solved via greedy algorithms that take advantage of the matching at each step; thus the utility of greedy algorithms applied on complete bipartite graphs where the two stable sets A and B are respectively the set of items/stacks and the set of trucks.

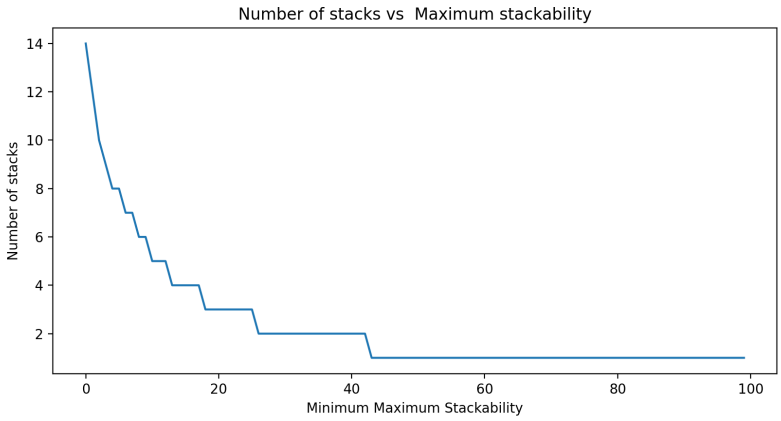

Figure 2: Number of stacks vs Maximum stackability

7.2 Divide-And-Conquer the constraints/parameters: Scalability problem

This approach is similar to the one adopted with respect to the objective function given that some constraints are independent from other constraints (and thus logically separable). Just like the objective function, the motif of such methodology used in dealing with constraints and/or parameters is to gain insights into the nature of problem and how the constraints interact in microscales and macroscales of the problem.
Some constraints are independent from each others (matroids) and thus could be dealt with separately guaranteeing the superposition principle (linear independence, spanning vector spaces of possibilities).

The very first subproblem solved is one with the minimum amount of constraints possible; basically a generic bin packing problem where the only constraints are to deliver all the items such that their sizes fit the trucks (bins). This problem is solved using the famous Fit algorithms and Greedy algorithms. For Fit algorithms, the First-Fit algorithm, First-Fit decreasing algorithm are used. In the case of greedy algorithm (in this case being the Best-Fit algorithm), complete bipartite graph-based algorithms are used . It's worthy of mention that there are two flavors to solving this problem; item-based loading or truck-based loading with the difference being which object is fixed in the process (ceteris paribus), as obvious as it sounds, item-based loading is performed by selecting an item (hold fixed) and searching for the first truck in which it fits in terms of weight constrained (First-Fit algorithm, First-Fit decreasing algorithm, First-Fit increasing-decreasing algorithm via bipartite graphs...etc). Truck-based approach is performed by selecting a truck (one with the biggest size in the case of Best-Fit algorithm) and filling as many items as possible into it via greedy algorithm until it runs out of space (recovering problems vs packing problems) and repeating the same thing for other "necessarily used" trucks.

The second type of subproblems is know as VM packing problem. VM refers to virtual machine which is an allusion to the almost magical property of space-sharing in packing problems (similar to nuclear mass difference between nucleus and nucleons in the atom). Virtual machines tend to occupy less memory when packed into a server because they share the same pages that need to be stored once (basically memoization). Similarly, VMs can be replaced by stacks in this case such that items occupy less space due to sharing space when packed into a hierarchy which is the stack structure. Basically, the same set of techniques and algorithms that solved the bin packing problem are used, except that they are naturally scaled to fit the stack structure. However, prior to solving the VM packing problem, the notions of stackability code and maximum stackability are examined. First, a comparison is drawn between the case of items having stackabiltiy codes versus items without stackability codes. Second, the relationship between the number of stacks (dictated by the number of different stackability codes) and maximum stackability is examined (inversely proportional). The following figure clearly shows that if the average maximum stackability of items is too high, one would end up with the minimum amount of stacks possible (and it makes sense because that's a recovering problem too!):
This relationship is of crucial importance to our problem, that's why I chose to visualize it. In fact,

the solution to the problem lies in this very graph where the following observation is made; the number of trucks used is lower in the case where the average maximum stackability of items is higher than the otherwise case. As mentioned before, in the case where there is no stackability code, the stacks in the graph shown can be replaced by trucks because they come in arbitrary cycles. The whole point of the challenge is to keep the function in the graph low.
VM packing problem is solved using filtration (stochastic vs deterministic). In the stochastic case, the probability of selection is determined by the arrival time to the late arrival time, concomitant to sorting the trucks sizes in a decreasing order. The deterministic case is naturally where this is not the case such that the filtration is performed using pre-determined values.

8 Heuristics-Metaheuristics:

8.1 Heuristics description and justification:

8.1.1 Graph-based greedy algorithm

the heuristic adopted is a combinatorial optimization based on a pseudo-complete bipartite graph between trucks nodes and item/stacks nodes such that a greedy algorithm is performed to maximize the number of items within a truck which is the dual problem of the minimization of the number of trucks needed/used. This method starts with the truck with the largest size and the largest arrival time (the closes to late arrival time), it then performs a local greedy optimization by filling most of the items within that truck. It then moves to the next truck (node) and solve the problem recursively This method assumes two fundamental assumptions: 1-Optimal substructure: the optima solution of the main problem contains optimal solutions to the subproblems. 2- Overlapping subproblems: the same problems are solved recursively, the only difference is the size of the problem (in our case; the number of items and the number of trucks available) thus a memoization technique (often used in dynamic programming) is used to store the solutions for the last subproblem solved (to optimize computational efficiency) that solves the problem recursively.

8.1.2 Genetic algorithms

Genetic algorithms are computational methods used to solve optimization problems by simulating the process of natural selection. The algorithm starts with a random population of candidate solutions to the problem and applies genetic operators such as crossover, mutation, and selection to generate new populations of solutions. The fitness function, which evaluates the quality of each solution, guides the selection of the best solutions for the next generation. This iterative process continues until the algorithm finds the optimal or near-optimal solution to the problem. Genetic algorithms are useful when the problem is complex, and the search space is too large to enumerate all the possibilities (explicit/exhaustive brute force search) or difficult to define explicitly. They can efficiently explore the solution space and converge towards a global optimum or a good solution. Genetic algorithms are also flexible and can handle various types of problems, including optimization, machine learning, and data analysis.
The goal of this problem is to minimize the number of trucks used for item packing while also selecting the trucks with the maximum ratio of arrival time to the latest arrival time. To avoid computational redundancy, it is assumed that each truck has a pre-defined number of items K=n/m in it. This "methylation" hypothesis considers the item-truck insertion operation as a black box a posteriori, meaning it has already happened without knowing how. That is to say that the subproblem K = 2 is "methylated" or skipped. I preferred to call it that way because it's similar to the phenomenon of methylation in gene expression where genes from one parent methylate the other genes from the other parent (here taken to be y and x such that y methylates x). Therefore as a corollary to the methylation hypothesis, the genetic algorithm will be applied only on the y[j]s. The population generated is divided into two types of chromosomes, namely K-lists and p-lists, which are almost two different populations but linked in an order-specific way. The K-lists represent the first term of the objective function (a*sum(TCty)), and each list in the population has K ones. On the other hand, p-lists represent the truncated m! population of permuted lists for each K-list. Hence, the total number of solutions in the population is Npop= e*p, where e is the number of K-lists, and p is the number of permuted lists for

each K-list (e*m! without truncation).
While the methylation hypothesis works just fine judging by convergence/stability of results, the fly in the ointment is that such a hypothesis falls short of results accuracy as this is the limitation of such a hypothesis that results in disrupting the semantics of the problem. As mentioned before, methylation is equivalent to skipping the subproblem K = 2 (item/stack-truck insertion) as a black box process but then this part of the optimization problem is cut short and left randomized which naturally makes the solution less optimal exactly due to the principle of indifference arising from such a randomization (ie; the methylation is "indifferent" to the optimality of the solution to the subproblem K = 2).
For convenience and algorithmic ease, the methylation hypothesis is kept in the case of genetic algorithms due to the number of lines of code involved. However, that's no longer the case in simulated annealing and tabu search as both cases are examined.

8.1.3 Simulated Annealing

Simulated annealing is a global optimization algorithm that is memoryless and takes inspiration from the process of annealing solids in solid mechanics. This process involves heating solids up to a certain point to increase their ductility and obtain regular crystals. The analogy of lowering activation energy in Enzyme-Catalyzer activation is also valid. Ductility in this context refers to the degree of tolerance/acceptance of bad solutions for search space exploration. In the SA algorithm, heating is equivalent to being generous while cooling (which must be progressive to avoid premature convergence) is equivalent to being greedy/exploitative. This helps in balancing the exploration-exploitation tradeoff. In fact, "Temperature" in the context of SA is the same concept encountered in machine learning hyperparameters where it's used to control the randomness of predictions and introduce stochastic variation in the results (for exploring large search spaces) widely used in Natural Language Processing, GPTs (it's, for instance, the reason why chatGPT throws different responses to the same prompt after "regenerate response") and LSTMs..etc; It's all the same thing.
There is a possible enhancement of the simulated annealing algorithm using tabu lists in order to minimize the redundancy of randomized solutions and avoid computational waste (usually an intermediate tabu length is most favored). The problem is solved in both cases.

Simulated Annealing works relatively perfectly on functions with "Weierstrassian" topology (ie; functions with stochastically many local optima). Examples: Ackley function, Eggholder function, Weierstrass function (obviously)...etc

The Methylation Hypothesis assumes that to avoid computational redundancy, each truck has a pre-defined number of items (K=n/m) in it. This hypothesis treats the item-truck insertion operation as a black box aposteriori, which means it has already happened without knowing how. As a corollary of this assumption, Simulated annealing will be applied only on the y[j]s. However, unlike genetic algorithms, this hypothesis can be downplayed as it's not much necessary in terms of algorithmic ease. The problem is solved in both cases.

8.1.4 Maximum Independent Set search via Intersection Graphs (Vertex Packing)

This heuristics makes the use of the dual recovering problem of our packing problem
the intersection graph used have non-identical rectangles as its vertex sets A lot of notions are involved: complement graph clique problem, percolation theory (percolation threshold), ramsey theory (relationship between graph size and probability of finding an independent set/clique)

The methodology being described here involves using a correlation-based model to create a Platonic representation of trucks and their optimal arrangements, achieved through independent sets. The graph-based method used involves finding maximum independent set (not to be confused with maximal independent sets), which is equivalent to maximizing packing density and minimizing redundant void, both of which are important factors when there is no explicit formula for geometry-specific Gibbs entropy. The independence number of each truck is positively correlated with its size and optimal arrangement. Correlation-based modeling is used to compare trucks to graphs and determine the sparsity ratio, or the number of zeros in the AdjMat resulting from the topological shape of the graph.

A high sparsity ratio indicates that the occupation probability is far from the percolation threshold, meaning that the probability of finding a maximum independent set with high independence number is high. The maximum graph degree is also a measure of its independence number. However, there is a trade-off between the difficulty of finding an independent set and the quality of the independent set (rock and gold analogy). The truck size is related to the efficiency ceiling, while the arrangement is achieved by reducing the number of edges in the isomorphic graph to reach the maximum sparsity ratio guaranteed by optimal arrangement. There are two cases to consider: when the number of stacks is concrete and when it is abstract. In the concrete case, there is a dependence problem between different truck loadings, which can be solved by prioritizing trucks with the highest size. In the abstract case, there is no dependence problem, but the isomorphism between trucks and graphs can be disrupted. Ultimately, there is a trade-off between truck loading independence and the independent set problem, which can be resolved through the intersection of sets in the graph representation.

8.1.5 Tabu Search

Tabu Search is a memory-based local optimization algorithm that makes the use of a tabu list of temporarily (depends on the tabu tenure/expiration rate) prohibited solutions either because they were visited previously or violate a certain user-defined rule, in order to avoid premature convergence to local optima and explore the search space. In Tabu Search, there is a trade-off between memory requirement and exploration process; it is favorable to maintain a balance between the two through an intermediate-length tabu list. The tabu list is basically a queue structure based on FIFO discipline dictated by the threshold which is the tabu tenure in this case (in hybrid TS, this threshold changes depending on the value of the fitness function).
At times, the aspiration criterion is used in order to give a chance to previously visited solution that eventually, after the exploration phase, turned out to be a better solution, or at least not as bad, as the current one

8.2 Comparison table

Algorithm	**Speed/Convergence Rate**	**Stability**	**Precision**	**Big O Notation Complexity**
Genetic Algorithm	Slow/High	High	Low	$O(N^2 * L)$ to $O(N * L^2)$
Greedy Algorithm	Fast/Low	Low	Low	$O(N^2)$
Simulated Annealing	Moderate/Moderate	Moderate	Moderate	$O(N^2)$
Independent Set Search	Fast/High	High	Low	$O(2^N)$
Tabu Search	Moderate/Moderate	High	High	$O(N^2)$

Table 1: Complexity comparison table between algorithms for packing problems.

In combinatorial problems, different algorithms have their own strengths and weaknesses. Here is a brief explanation of the differences between the algorithms mentioned above in terms of combinatorial problems.

Genetic Algorithm: Genetic algorithms are a type of optimization algorithm that uses principles from natural selection and genetics to search for solutions. Genetic algorithms are good for problems with a large search space and many possible solutions, but they can be slow and imprecise compared to other algorithms.

Greedy Algorithm: A greedy algorithm always chooses the locally optimal solution at each step, without considering the overall optimal solution. Greedy algorithms are fast, but they can produce suboptimal results.

Simulated Annealing: Simulated annealing is a probabilistic technique for finding the global minimum of a function. It is a good choice for problems with a large search space and many possible solutions. Simulated annealing is slower than greedy algorithms, but it can produce more accurate results.

Independent Set Search: An independent set is a set of vertices in a graph that are not adjacent to each other. Independent set search is a type of optimization algorithm that searches for the largest independent set in a graph. This algorithm is fast and precise, but it only works for problems that can be represented as graphs.

Tabu Search: Tabu search is a type of optimization algorithm that keeps track of recently visited solutions to avoid getting stuck in local optima. Tabu search is a good choice for problems with many local optima. It is slower than greedy algorithms, but it can produce more accurate results.

The Big O Notation complexity is an estimation of the worst-case time complexity of an algorithm. Here I added a column to the table to show the Big O Notation complexity of each algorithm in terms of packing problems.

The genetic algorithm's complexity can range from $O(N^2L)$ to $O(NL^2)$, depending on the population size and the length of the chromosome. The greedy algorithm's complexity is $O(N^2)$, where N is the number of items to be packed. Simulated annealing has a complexity of $O(N^2)$, where N is the number of iterations. Independent set search has a complexity of $O(2^N)$, where N is the number of vertices in the graph. Finally, tabu search has a complexity of $O(N^2)$, where N is the number of items to be packed.

It is important to note that the Big O Notation complexity is an estimation, and the actual performance of an algorithm can be affected by many factors, such as the implementation, the input size, and the specific problem being solved. Therefore, this complexity comparison is only a rough estimation of the performance of each algorithm in packing problems.