Axel Fridman application to the Shipping Optimization Team at Mercado Libre

Q1

I have formulated and implemented a mixed integer linear programming model to determine the optimal operations to maximize profit on a 4 week time period. The github to the code repository can be found on https://github.com/AxelFridman/MELIapplication with comments and explanations that reflect my understanding of the problem and the mathematical model behind it.

a) Each day EcoMole had, for each product, to take the decision of how many will they inbound ship from a seller of that product, how much will they transfer in between fulfillment centers, and how much will they deliver to a state with remaining demand.

An assumption taken was given that demand was specified as weekly and dependent on state, and safety stock was specified as days of "the forecast demand of d days of sale", that left room for much interpretation. For example, was the forecast based on the demand of that particular week or the upcoming week or an average of all weeks? What about states? And should we consider that a fulfillment center rarely distributes the last mile to all the states and therefore the forecast should consider the distribution between fulfillment centers? I eventually decided to assume that the amount of safety stock of a given product P and fulfillment center FC would be equal to

$$DaysSafety_{FC,\,P} * \left(\sum_{st:\,States}\ \sum_{w:\,Weeks} Demand_{P,\,st,\,w}\right) / \left(\#States * \#Weeks * \#Fulfilment\ centers\right)$$

Where # Indicates the amount of each entity.

Another assumption taken was that since sometimes transfers costs between FC were cheaper than the storage cost of a FC, it was sometimes convenient to keep products moving rather than static. Therefore in my implementation I assumed that while on transit there is no storage cost. Storage cost is only paid while static on a fulfillment center.

I also assumed that if a truck leaves towards the last mile on a week (for example day 6) but arrives at a state on another week (say day 8) then the demand that was being satisfied was the week 1 demand, and not the second.

Another necessary assumption was that since initial stock was sometimes below the safety stock levels, the restriction would only apply when there was already an opportunity to have an inbound shipment. That meant that if it took 2 days to inbound ship the restriction would only apply from day 3 onwards.

The results were consistent and reasonable, the product transfers between FCs that maximize the profit of Ecomole can de found on file **answers1a.txt**
Profit (Revenue - Costs) = $98.913.072

b) The way it was implemented part b was to simply modify the cost of delivering the last mile from a FC to a State for each possible routing which took more than 2 days, and changing the cost of those routes to “M " a big number. Therefore we were left with the same feasible solutions but now the preferred ones by the objective functions were those that involved fast travel to the client from the FC. The optimal solution for this new variation was extremely costly and unprofitable, it seems like there was no solution found where all the products are delivered in less than 2 days. From the 6 million products delivered there were millions that didn't make the 2 day timespan. The company can mitigate the impact by providing approximately 2 days delivery, where the cost of the last mile is P% more expensive if we don't deliver quickly, but not an infeasible solution. When P = 15% meaning if we take more than 2 days the cost is 115% the original cost I found that profit went down to $50.788.873 which is a 48% decrease in profit. While last mile costs went from $725.746.875 originally up to $773.871.809 approximately a 6% increase but it had a huge impact on profit. There were also more deliveries that satisfied being in the 2 day time-span, but not a significant reduction on late deliveries.

c) Safety stock was found to have a considerable effect on the profit of the firm. I experimented by variating the percentage of safety stock that must be present at all times on a given FC. On the original problem there must be at least 100% of the safety stock at all times, and when I changed the percentage to 50% on each product and FC profit improved to $134.838.976 which is a 36% increase in profit and a 41% decrease in storage costs. When there was no safety stock (0%), profit was $165.253.516, which meant a 67% increase in profit compared to the baseline of 100% safety stock. Overall profit can be highly dependent on costs, and cutting costs by reducing the minimum amount at storage at a FC can be an easy and applicable action towards the goal of maximizing profit. Especially when ignoring the risk of volatile demand on the system.

d) A way to implement this new approach to the problem would be to run the model for each possible case. That would take a lot of different runs of the model and can be time-consuming. Another approach can be a greedy type of algorithm that at each step chooses the best time of delivery for a given category. After finding the best for all categories, we obtain that the best possible time-delivery for any of the products will be the best in its respective category.

e) Linear programming is great, but not good enough for a problem of this size. One thing to point out is that products from the same seller might have the same behavior in real life, and might have similar categories, and that could save time in computing power when devising a solution if we could group similar products easily.
Another idea that has come to my mind is that there is no inter-relation between products on this particular problem, and there is no initial cost to an operation, all operations are proportional to the amount related to the specific operation. With this in mind it is easily seen that if we divide the set of all products in 2 parts P, P’ and solve each problem optimally individually and then merge solutions, then the combined solution would still be optimal! This represents an opportunity to develop a quick heuristic for each individual product and then run that heuristic on all products simultaneously. In this sense we could arrive at a quick solution by combining good solutions of each product. Which heuristic suits the problem best

is hard. I'd try with tabu-search, simulated annealing and genetic algorithms, and continue refining on the most promising ones.

Q2

a) and b)

I will be giving the following representation to the labeling problem. Let's consider the following representation of the problem. Each label l shall be a vertex V_l of graph G, and we will have an edge E_l, l' connecting V_l to V_l' if and only if they don't share any elements in common. That is, there is a feasible solution that contains both of those labels such as the set L' = {l, l'}.

It is now clear that our problem consists of finding the max-clique of G, as that would imply that none of the boxes share an element and the solution is maximal. Max-clique is NP-complete therefore there is no known polynomial time algorithm to solve it.

Note that by using this representation we are not "using" the information given by each label being non disjoint. This representation would allow a label that has 2 separate components which is not our case, nonetheless we can consider this to be a practical generalization.

Quick disclaimer: I will be providing a computer program that finds the max-clique of a graph designed in group for a course I took at college a few years ago. It is certainly not the best implementation, it has not been maintained, and it is not elegant, but it is sufficient.

Note that in this case an upper bound K to the size of the max clique is the number of vertices with degree K or more. So if there are less than K vertices with degree K or more, then K is an upper bound to the size of the max-clique. For example if the degree of the vertices is [2,4,6,1,3,3,2,7] where the i-th position represents the degree of vertex i, then 3 is an upper bound, as there are no 4 vertices with degree 4 or more.

Another way of going about this problem is Heuristics. Let's consider the following representation of a solution: Consider the tuple {0, 1}^#(L) where #(L) is the size of the set L, and 0 represents that the box is not a part of the solution and 1 that it is.

Now comes the problem of the objective function, we would like to have the optimal value of the objective function when the solution is feasible and the sum of 1's on the solution is maximal. So we can preliminarily define it as: "The sum of 1's"

Now we are left with the problem of how we should find a starting feasible solution and furthermore maintain it feasible as we progress with our heuristic.

With that in mind I shall define a neighborhood and variation operators that fulfill such a task. Our neighborhood of a given solution S will be those with hamming distance 2 of our current solution, this is a good tradeoff between having large neighborhoods which are slow to explore and small neighborhoods which are fast to explore but risk our solution to be stuck at local optima. Also we shall impose our heuristic to never choose an infeasible solution, and as we shall start with a feasible solution it will always have one alternative solution to choose from (sometimes for a worst objective value).

The heuristic chosen will be the modest and classic hill climbing, partly because of its simplicity, partly because of its effectiveness on similar problems, and partly because I have already previously implemented it. It is obvious that it is possible to choose (0,0,0…,0) as a starting feasible solution, which provides the worst objective value but is always feasible regardless of the size of L. The results were satisfactory and I was able to achieve very fast

and good results with around 100 labels. The python code can also be found on the github repository by the name 2hillclimbing.

Q3.

a)

Let's consider a representation of the problem as tuples of non-negative integers. Where (A, B) means there are A number of matches in stack 1 and B is the number of matches in stack B.
We will prove this using induction, let's consider as a base case the instance (1,1) and it's the first player turn. As there are the same number of matches in both stacks and unlike (0,0) the game is not yet over. Each player must remove at least 1 match from exactly 1 stack, therefore as there is only 1 match in both stacks the player will remove a single match from either of the 2 stacks, let’s assume without loss of generality that the first player removed the match from stack 1. Therefore it's now the second player turn and he/she is left with (0, 1) which is indeed a “forced win”, as there is no other option than to remove the last match and consequently win.
Indicative step: We will have as our inductive hypothesis that if both stacks have the same number of matches, and the amount of matches in each stack is less than N, and it's the first player turn, then the second player can always win.
We are assuming that for every (K,K) situation then the second player can always win for every K natural number such that K<N.
We will now prove that it will still hold for the instance (N,N).
Suppose without loss of generality that the first player took some J number of matches from stack 1. Therefore the second player is left with the instance (N-J, N). If J=N then obviously there is an easy win for the second player as he can remove J matches from the second stack, end the game and win. If J is not N, then the second player could mimic the first player but remove it from the second stack. Leaving now the first player with (N-J, N-J). As J is not N, then N-J is a natural number that we could rename K, and by our inductive hypothesis (K,K) is a win for the second player for every K<N.
Now we have proven that it holds as well for (N, N) and by following these inductive steps it holds for any natural number.

b)

It's a tricky proof but it is incorrect. Notice how it is indeed true when n=1, but let's examine p(2). The “proof” sorts the dog and then selects from the first to the one previous to the last, and then selects from the second up to the last, and then subtly claims that the intersection of these sets always contains a dog, but that is not correct.

When N=2 the first set is just the first dog, and the second set is just the second dog, there are no dogs in both sets (because the first one is the same as the previous from the last, and the second one is the same as the last one).
Therefore there is no proof. Fun fact: If it were true for n=2, then the argument would be correct for all natural numbers. Obviously it's not, and that's why we don't all have the same dogs.

Feel free to contact me if any of the explanations provided were hazy, I'll be happy to clarify.
Thank you for your consideration, best regards.