Heuristics project

by

Siro Brotón

NIA: 100496683

Email: 100496683@alumnos.uc3m.es

Contents

1 Part 1: excel

Part 1 is the simplest part, since we just have to find the combination of buses and slots that minimizes the total distance traveled. This can be done with a matrix of booleans that "activate" the values of the distance matrix. We have to enforce the constraint of only one boolean is true on each row and column, meaning that only one bus per slot and one slot per bus. After imposing our constraints we can use the solver to get a solution that minimizes the distance traveled.

To make this last computation cleaner we have transformed the matrix into a vector concatenating the columns.

2 Part 2: glpk

2.1 Minimization of the impact of breakdowns

The first part of this problem is understanding what we are doing: we can either put a bus in a slot and pay for the distance, or not put it and pay for the compensation. There are n slots and m buses and we have to see what configuration of buses minimizes the losses. We have buses that we want to let break and buses that we prefer to maintain depending on whether $k^d d_i - k^p p_i$ is bigger or smaller than 0, and from the ones that we prefer to maintain we try to fit the most advantageous.

Now we can go into the modeling. We create $n \times m$ binary variables to signal whether a bus is in a slot. $X_{i,j} \Leftrightarrow$ bus i is assigned to slot j. We have to implement 2 constraints:

- A bus only can be in 1 slot: This can be modeled like $\sum_{j=1}^{n} X_{i,j} \leq 1 \quad \forall i$, meaning "adding all slots of any bus has to be at most 1".
- At most one bus per slot: Modeled like $\sum_{i=1}^{m} X_{i,j} \leq 1 \quad \forall j$ and meaning "adding all buses of a slot has to be at most 1".

For the actual implementation we have opted for parameters corresponding to n and m, as well as for the penalties of distance k^d and passengers k^p. Due to a small undetermined problem to get the real loss we had to factor a constant part of the objective function into a variable c. The actual loss function that we want to minimize is:

$$z = k^d \sum_{i=1}^{m} d_i \left(\sum_{j=1}^{n} X_{i,j} \right) + k^p \sum_{i=1}^{m} d_i \left(1 - \sum_{j=1}^{n} X_{i,j} \right)$$

which can be simplified to:

$$z = \sum_{i=1}^{m} \sum_{j=1}^{n} \left(k^d d_i - k^p p_i \right) X_{i,j} + k^p p_i$$

The generation of the inputs is as simple as reading the filenames, copying the first, separating the parameters with `split()` and the elements of the lists with `split(",")`, and writing in the second with a fixed format. A python script called "problem_generator1.py" was created to generate a random *.in* with a given amount of buses and slots.

No checks are done in the data given since we think the focus of the project should be in the problem solving, not in the parsing. To show which buses are assigned we go through all variables and if any of them is 1 we display the bus and where it is assigned.

When no buses are shown in the output it is understood that no bus was assigned.

2.2 Maximizing passenger satisfaction

Similarly, we have to first understand what we are asked to do. Now we have to forcibly assign every bus. Also, now we have multiple workshops and some of them have "closed" slots. In this case we have to minimize the number of people that lose 2 buses because they are assigned at the same time.

The modeling is a bit more complicated. We use the same idea as in the first part, but now in a 3-tensor instead of a matrix. The binary variable $X_{i,j,k}$ represents whether the bus i is assigned to workshop j at time k. We also introduce a new binary variable $f_{p,q,k}$, which represents whether there exists a conflict between buses p and q at time k. This could clearly be simplified by imposing that $p < q$, but we have decided that the time investment is not worth it. Also, notice that these aren't really variables we choose as much as a result of choosing the values of $X_{i,j,k}$, but they become incredibly useful for the objective function since it is precisely what we want to minimize. Additionally we create an auxiliary constant variable aux that takes into account the conflict from a bus with itself to remove it from the real value of the objective.The restrictions needed are:

- All buses are assigned exactly once: Equivalent to the first constraint of the first part. Modeled as $\sum_{j=1}^{u}\sum_{k=1}^{n} X_{i,j,k} = 1 \quad \forall i$
- At most 1 bus per slot: We also use this constraint to enforce the "closed slot" problem. It is modeled similarly to the second constraint: $\sum_{i=1}^{m} X_{i,j,k} \leq o_{j,k} \quad \forall j,k$
- Finding conflicts: We have to identify which variables $f_{p,q,k}$ are true and which ones have to be false. We know that
  - $f_{p,q,k} \Leftrightarrow \sum_{j=1}^{u} X_{p,j,k} + X_{q,j,k} = 2 \quad \forall p,q,k$
  - $\neg f_{p,q,k} \Leftrightarrow 0 \leq \sum_{j=1}^{u} X_{p,j,k} + X_{q,j,k} \leq 1 \quad \forall p,q,k$

  We can simplify them into: $2f_{p,q,k} \leq \sum_{j=1}^{u} X_{p,j,k} + X_{q,j,k} \leq 1 + f_{p,q,k} \quad \forall p,q,k.$ Notice that the family of f variables are completely dependent on the values of the X variables and (almost) viceversa.

Since most of the complexity is encoded in the constraints we have a very simple objective function:

$$z = \frac{1}{2}\sum_{p=1}^{m}\left(-c_{p,p} + \sum_{q=1}^{m}\sum_{k=1}^{n} c_{p,q} f_{p,q,k}\right)$$

We subtract all conflicts of buses with themselves and divide everything by 2 to account for double counting.

The parsing of inputs is almost as simple as the other one. The only difference is that we have 2 matrices instead of 2 vectors, so we have to make a few extra loops. Once again, we naively ignore the possibility of having an error in the input.

On the other hand, for generation we have created a program called "problem_generator2.py" that generates random problems given some parameters for buses, slots and workshops (when not enough spaces are available, the number of workshops is chosen such that it minimizes the leftover slots while leaving enough for all buses). The biggest solved problem was with 10 buses, 4 slots and 4 workshops, and it took almost 20 min.

3 Part 3: analysis

3.1 Model 1

The results coincide with what we can manually get for small models. The tests done for the first part were as shown in the first table

A surprising realization is that the first model can be simplified to a sort of a list of tuples of the form $(k^d d_i - k^p p_i, i)$. We would sort by the first value and get the n lowest negative ones (the second value tells us which buses we want to send to the workshop). To get the general trend we have developed a few python scripts that try a battery of random tests and records the performance. We have "problem_generator1.py", which generates a random problem with some given parameters, and "create_data1.py", which runs the other program multiple times with some default values for buses and slots and records the performance into "data1.csv".

The number of variables grows as $O(m*n)$, and the (defined) constraints as $O(m+n)$. We expect a very slow increase in the computational cost with respect to the number of buses. This is corroborated by the execution of "create_data1.py". We see everything is instant up to more than 500 buses and 100 slots. The data is already computed in data1.csv for some arbitrary increments in parameters.

Test	Objective	Result	Correct?
4 3 1 10 100,0,5 1,4,1	basic functioning with clear solution	loss = 15 buses 2 and 3 assigned	correct
4 3 1 10 100,0,5 11,1,1	basic functioning	loss = 105 all buses assigned	correct
2 5 1 2 10,0,0,1,5 100,1,1,4,20	not enough slots for all buses	loss = 27 buses 1 and 5 assigned	correct
5 5 2 5 206,66,263,104,154 50,60,40,30,70	more realistic example	loss = 1040 buses 2 and 5 assigned	correct

Table 1: Tests part 2-1

3.2 Model 2

Testing for the second part was much more tedious, since more parameters have to be given. For these tests we can draw the schedule and manually check the interactions. It's impossible to manually search for the optimal solution on a problem not explicitly designed with a given result in mind, so we have designed small tests to check for intended and unintended behaviors (see Table 2), and then automatically generated others to check performance. In the table we show designed problems as the final assigned timetable with the collisions at each slot added up on the side as $c =$ # collisions and x meaning that the slot is unavailable.

Regarding the number of variables: the first part grows as $O(m*n)$, while the second one grows much faster like $O(m*n*(u+m))$. The number of constraints is much smaller: the first part grows linearly like $O(m+n)$ and the second one like $O(m+u*n+m^2*n)$.

The efficiency is terrible. When we executed our battery of random tests we had to wait a few minutes for relatively small inputs(8 buses and up to 8 slots). It is important to note that the efficiency gets worse as the number of empty spaces increases (fitting 8 buses in 7 slots and 2 workshops is way worse than doing so in 3 slots and 3 workshops). All colected data is available in data2.csv

Test	Objective	Result	Correct?
2 3 2 1,0,1 0,1,1 1,1,1 0,1 1,1	basic functioning avoiding conflicts	passengers lost = 0 x | 3 c = 0 1 | 2 c = 0	correct
4 5 2 1,2,2,0,1 2,1,4,2,1 2,4,1,2,1 0,2,2,1,3 1,1,1,3,1 0,1 1,1 1,0 1,1	choose which ones should collide	passengers lost = 0 x | 5 c = 0 | 2 c = 0 3 | x c = 0 1 | 4 c = 0	correct
3 6 4 0,0,0,2,15,10 0,0,5,0,10,15 0,5,0,0,0,10 2,0,0,0,4,5 15,10,0,4,1,5 10,15,10,5,5,10 1,0,1,1 1,1,0,1 0,1,0,1	more complexity	passengers lost = 2 2 | x | 4 | 1 c = 2 5 | 3 | x | c = 0 x | | x | 6 c = 0	correct
4 10 4 C 0,1,1,1 1,1,1,1 1,1,1,1 1,1,1,1	Big (big.in)	passengers lost = 203 x | 1 | 9 | 7 c = 52 | | 5 | 4 c = 30 | 8 | 6 | 3 c = 95 10 | | 2 | c = 26	

Table 2: Tests part 2-2