[LINMA2471] Optimization models and methods
Homework n^o1

Simon Boigelot (72251300)
Quentin Lété (66421300)

October 2016

A. An optimal fast food diet

Daily menu

Our implementation of the model to obtain the cheapest menu that satisfies the guidelines can be found in the file `FastFood.mod` attached. The optimal solution we obtain is the following :

$$\begin{cases} \text{Grande Frite} & = 1 \\ \text{Minute Maid} & = 5 \\ \text{Petites Tomates} & = 14 \\ \text{Royale Deluxe} & = 1 \\ \text{Total cost} & = 42.8EU \end{cases}$$

We can see that the obtained solution is not very realistic. The solution advises us to drink 5 Minute Maid's and to eat 14 PetitesTomates which is not something one would do in practice. With the few contraints we have at that point, the optimal solution tends obviously to select a lot of cheap products that have not much nutriments to easily match the constraints.
To have a better (i.e. more diverse) solution, we will add some constraints. To take variety into account, we will introduce groups of products that are of the same kind : drinks, burgers, fries, chicken, salads and misc (standing for miscellaneous, where we gather products not belonging to any other groups). Now, we introduce two types of constraints :

1. We add a maximum number of products that we can choose for each group.
2. It seems fair to us to force the solution to contain at least one drink.

This is what we did in the files `FastFoodVar.mod` and `FastFoodVar.dat`. The bounds defined in the file `FastFood.dat` were really to tied so we relaxed them a little bit, in order to have feasible solutions. Indeed, we noticed that for the first part, the feasible solutions has to have PetiteTomates = 14 which is really too restrictive. The solution obtained is the following :

$$\begin{cases} \text{Grande Frite} & = 1 \\ \text{Hamburger} & = 1 \\ \text{Petit italien} & = 1 \\ \text{Evian} & = 1 \\ \text{Total cost} & = 8.9EU \end{cases}$$

The solution looks much more realistic, it's even a menu one could order in real life. In particular, each product is chosen at most once. We also notice that the cost decreased dramatically : this is due to the fact that we relaxed the constraints a lot.

Obviously this model is in a certain way the simplest model of this Fast Food menu we can make. We thought of a few ways one could improve the model for use in real life :

— If we collect data about each client, we could analyse them to return a solution specific for each client, taking into account his or her preferences.
— We could use this optimization problem to update the different products available in the Fast Food. For example, if we notice that one of the products, say the McFish, is never chosen, one could think of lower its price or replace it by a new product.
— We could adapt the bounds to the age and gender of the customer. Indeed, the Reference Daily Intake varies a lot with these factors.

Menus for the week

For this new problem, we can keep the constraints we had for the improved daily menu and add constraints to ensure diversity during the week. In order to do that, we introduced a minimum quantity of products in each group that we have to select each week. Also, we forced the products to be all different, i.e. we can't chose twice the same product during the week. Here we obviously had to use the relaxed constraints we used previously to get a feasible solution. The implementation can be found in the files `FastFoodWeek.mod` and `FastFoodWeek.dat`.

To estimate the variety of the solution, we can build the solution matrix with all the products in lines and the seven days of the week in columns and make sure that this matrix contains only 1's and 0's and that each line contains only one nonzero element (which is equivalent to the constraint saying that we can't choose twice the same product during the week). We should also create the six submatrices for each group and compute the number of nonzeros elements of these matrices. Those numbers shouldn't be very different for the seven groups. This is equivalent to the constraint imposing a minimum number of products chosen from each group for the whole week.

Figure 1 shows how our solution looks like for a total cost of €92.6.

```
quantity [*,*]
:                       1   2   3   4   5   6   7    :=
Badoit                  0   0   1   0   0   0   0
BigMac                  0   0   0   0   1   0   0
Cheeseburger            0   0   0   0   0   0   1
ChickenMcNuggets_20     0   0   1   0   0   0   0
ChickenMcNuggets_4      0   0   0   1   0   0   0
ChickenMcNuggets_6      0   0   0   0   0   0   0
ChickenMcNuggets_9      0   0   0   0   0   0   0
CocaCola                0   1   0   0   0   0   0
CocaColaLight           0   0   0   0   0   0   0
CocaColaZero            0   0   0   0   0   0   0
CroqueMcDo              1   0   0   0   0   0   0
DeluxePotatoesGrande    0   0   0   0   0   1   0
DeluxePotatoesMoyenne   0   0   0   1   0   0   0
DoubleCheeseburger      1   0   0   0   0   0   0
Evian                   0   0   0   1   0   0   0
FantaOrange             0   0   0   0   1   0   0
FiletOFish              0   0   0   0   0   0   0
FriteGrande             1   0   0   0   0   0   0
FriteMoyenne            0   0   1   0   0   0   0
FritePetite             0   0   0   0   0   0   1
Hamburger               0   0   0   1   0   0   0
IceTeaPeche             0   0   0   0   0   0   0
McChicken               0   0   0   0   0   1   0
McFish                  0   0   0   0   0   0   0
McWrapPouletBacon       0   0   0   0   0   0   1
McWrapPouletPoivre      0   0   0   1   0   0   0
MinuteMaid              0   0   0   0   0   0   0
PtitItalien             0   0   0   0   1   0   0
PtitWrapRanch           0   1   0   0   0   0   0
PtiteSalade             0   0   0   0   0   0   0
PtitesTomates           0   0   1   0   0   0   0
RoyalBacon              0   0   0   0   0   0   0
RoyalCheese             0   0   0   0   0   1   0
RoyalDeluxe             0   1   0   0   0   0   0
SaladeCaesar            0   0   0   0   0   0   1
SaladePDT               0   1   0   0   0   0   0
SaladePateMozzarella    0   0   0   0   1   0   0
Sprite                  0   0   0   0   0   0   0
;
```

FIGURE 1 – Solution of the problem A.2

B. Factory Planning

To model this problem, we used two different indexes variables :

— x_{ij} : quantity of product j produced at month i

— s_{ij} : quantity of product j stocked at month i

However, notice that it could have been possible to use a variable y_{ij} representing the amount of product j sold at month i. But as shown below, there is a relation between these three variables so you only have to use two of them since the third can be expressed as follows :

$$y_{ij} = s_{i,j-1} + x_{ij} - s_{ij}$$

Fixed maintenance

Our implementation of this linear problem can be found in the files `factory_planning.mod` and `factory_planning.dat`. We obtained the following results (tables 1 & 2) for the quantity of each

product respectively produced and warehoused each month. The final revenues are of €106016,5. We chose not to take the storage into account in our objective function since it will normally be paid the next month.

Quantity produced	Product 1	Product 2	Product 3	Product 4	Product 5	Product 6	Product 7
January	500	888	383	300	800	200	0
February	700	600	117	0	500	300	250
March	0	0	0	0	0	400	0
April	200	300	400	500	200	0	100
May	0	100	500	100	1000	300	0
June	550	550	150	350	1150	550	110

TABLE 1 – Solution with fixed maintenance : quantity of products produced at each month

Quantity stored	Product 1	Product 2	Product 3	Product 4	Product 5	Product 6	Product 7
January	0	0	83	0	0	0	0
February	100	100	0	0	100	0	100
March	0	0	0	0	0	0	0
April	0	0	0	0	0	0	0
May	0	0	0	0	0	0	0
June	50	50	50	50	50	50	50

TABLE 2 – Solution with fixed maintenance : quantity stored at each month

We can check that this solution makes sense and is feasible. Indeed, two scenarios are possible :

1. $x_{ij} \leq limit_{ij}$. Every product j produced in month i is also sold in month i.
2. $x_{ij} > limit_{ij}$ & $s_{ij} = x_{ij} - limit_{ij}$. The unsold amount of product j is warehoused in month i for another month.

Analysing the tabulars, it is clear that these relations are respected and that the final amount of stock is always 50.

Another argument could be that "only" products 1,2,4,5 and 7 need the borer in their production line. And as observed in the first tabular, the are no units of them produced during March because the only borer the factory possesses goes on maintenance that month. Furthermore, we can also notice that the revenues are not an integer because we store 83 instances of product 3, representing a cost of €41,5.

Flexible maintenance

Solving our linear problem with AMPL, we obtain the following (tables 3 & 4) results for the quantity of each product respectively produced and stored each month. The final revenues are of €109030. Table 5 shows how maintenance is organised during the semester.

Quantity produced	Product 1	Product 2	Product 3	Product 4	Product 5	Product 6	Product 7
January	500	1000	300	300	800	200	100
February	600	500	200	0	400	300	150
March	400	700	100	100	600	400	200
April	0	0	0	0	0	0	0
May	0	100	500	100	1000	300	0
June	550	550	150	350	1150	550	110

TABLE 3 – Solution with variable maintenance : quantity of products produced at each month

Quantity stored	Product 1	Product 2	Product 3	Product 4	Product 5	Product 6	Product 7
January	0	0	0	0	0	0	0
February	0	0	0	0	0	0	0
March	100	100	100	100	100	0	100
April	0	0	0	0	0	0	0
May	0	0	0	0	0	0	0
June	50	50	50	50	50	50	50

TABLE 4 – Solution with variable maintenance : quantity stored at each month

Maintenance	Borer	Grinder	Horizontal drill	Planer	Vertical drill
January	0	0	1	0	0
February	0	0	0	0	0
March	0	0	0	0	0
April	1	2	0	1	1
May	0	0	0	0	0
June	0	0	0	0	0

TABLE 5 – Solution with variable maintenance : number of machines undergoing maintenance at each month

As expected, the revenues are boosted by the flexibility we get from deciding when each machine goes on maintenance, even if the constraints are not exactly the same between the two problems. For example, the planer does not need to be maintained during the fixed maintenance semester but well during the flexible semester. However the boost is not huge. Actually, the freedom we get by deciding the maintenance is restricted by the constraints we have on the number of machines that have to be fixed.

The maintenance tabular is very interesting. Indeed, it seems that it is more profitable to maintain almost all the machines during the same month. In this case, April. April is not randomly selected. It corresponds to the month where the maximum total amount of products sold is the smallest (1700 units). So this is quite a logical way of organising maintenance. At least on the profitable way. In reality, this would mean that the factory is closed in April. This is obviously not very realistic. To make it more plausible, we decided to add a constraint imposing a minimum production each month (here 1000 products) and compare the profits. The new profit is €101905 which corresponds to a decrease of approximately 6% with respect to the previous situation.

Analyzing the maintenance tabular, we were wondering why only the horizontal drill does not go in maintenance in April. We quickly thought that maybe 3 horizontal drills were too much for their usage. We thus tried to run the problem with 2 horizontal drills instead of 3 and guess what... the optimal solution is the same! We can thus learn more than we think from the model if we analyze correctly the data. In this case, one horizontal drill is unused during the semester and is thus useless. The factory could then imagine selling it to buy another machine to boost the production of another product of interest.

The solution proposed is feasible and optimal if we only consider the objective function and the constraints that have to be imposed. However, it is not very realistic . We could add constraints to have a better solution, for example to impose sequencing in the production.