INF421 2021: Matching under constraints

Taha Hammadia and Styve Ngamou

February 2022

1 How to use the material?

This report contains the analysis of our code. The code itself is available in the `src` directory sent with the report. The results are available on the `Results` pdf file. If you want to execute the code, please make sure that the importation of the other `Python` files works properly. Feel free to change the value of the path in the beginning of some files.

The `Python` files are named by the name of the task they answer to.

2 Context:

In this project, we are studying many-to-one matching problems. These problems have many applications such as allocation of daycare slots to babies or allocation of professors to schools and others. For our project, we will use the terminology of *students* and *schools*.

Nowadays, the subject of resource allocation is more and more present in the public scene since social justice advocates ask for the inclusion of affirmative action criteria in the schools selection process. In this project, we will see how to implement some of these criteria such as quotas and the $\frac{4}{5}$-rule. During this phase, we will try to ensure that the matching is *stable*, i.e. feasible, individually rational, fair and non-wasteful.

In the last part of our work, we will see how to optimize matching *without stability constraint*. Indeed, we will consider that a central authority will match students to schools with a budget constraint. The cost of each student will depend on their group, which can correspond to different needs.

Disclaimer:
In this assessment, we may use terms that are not politically correct. These term mean no political statement and we are only discussing programming aspects.

3 Task 1:

Since the definitions of feasibility and fairness are the same for both notions of stability. We only need to show that Gale and Shapley stability implies non-wastefulness and individual rationality.

Non-wastefulness:
Let μ a matching. Suppose there is $(i, s) \in I \times S$ such that $(s >_i \mu_i)$ and $|\mu_s| < q_s$.

We have: $i >_s \emptyset$ and $s >_i \mu_i$. Since $|\mu_s| < q_s$, the student i can choose to let go their school and enroll in s, which will be beneficial for both i and s. μ is unstable.

Individual rationality:
We think that there is a problem in the wording of this condition. We understand that it means that for each $i \in I$, $\mu_i \geq_i \emptyset$.

Suppose by contradiction that there is $i \in I$ such that $\mu_i <_i \emptyset$. i can quit the school μ_i and stay without school. In this case, the quota condition is still satisfied. μ is therefore unstable.

In conclusion, the two notions of stability are equivalent.

4 Task 2:

4.1 Discussing the implementation:

Here, we simply implement the algoithm of Gale and Shapley adapted to many-to-matching problem. Instead of ending when every "male" has a match. We end when the schools have asked all students or filled their quotas.

We would like to point out that we have discussed the possibility to use the Dinitz-Edmonds-Karp algorithm to an ajusted flow network. However, our implementation of Gale-Shapley seems to have a better complexity.

4.2 Choice of data structure:

We have opted to use class structures which helped us combine data over students and schools in an efficient way.

In the `student` class, the list `prefStud` is such as for each $s \in S$, `prefStud[s]` is the ranking of the school s for the student. This choice of data structure makes the access to the ranking easier, reducing time complexity.

In the `school` class, the list `prefer` contains the students ordered according to the preference of the school. This choice is justified by the fact that we only need to go through the list in order to ask the students in order of preference of the school.

In the main function `mate`, the result is presented into the shape of a list of lists. Each list within the great list contains the students who have integrated the school corresponding to the list. The last list corresponds to students who do not have a school.

4.3 Complexity:

4.3.1 Complexity in time:

We count complexity in terms of how many times a school asks a student. This way of counting is justified by the fact that our choice of data structure makes each "asking" in constant time.

In the algorithm, each student is asked by a school at most once, therefore in the worst case, the complexity is $O(Nm)$, where N is the number of students and m the number of schools.

4.3.2 Complexity in memory:

For each one of the N student, a list of $\Theta(m)$ number represent the ranking of schools. For each one of the m schools, a list of N pointers to elements of the class `student`.

At the end, the space complexity is $\Theta(Nm)$.

5 Task 3:

5.1 Discussing the implementation:

As before, we are adapting the Gale and Shapley algorithm to our problem. The ending condition is not changed; whereas a school cannot ask a person only if the number of waiting students of their group is strictly less than the quota of the group.

Since the algorithm goes through the preferences of the schools in decreasing order of preference, we have a stable matching when all quotas are met. Since a school cannot prefer a student it does not have only if they are in a school they prefer. Moreover, at each iteration, we have a stable matching formed by the schools and the students who have matched. Therefore, the algorithm is correct.

5.2 Choice of data structure:

We use the same structures as before. Moreover, in order to account for groups, we add an attribute group in the `student` class and a list in the `school` class that corresponds to the number of people of each group.

5.3 Complexity:

5.3.1 Complexity in time:

As earlier, we see that they are at most $O(Nm)$ "demands". Furthermore, thanks to our data structure, each "asking" takes a constant time. Therefore, the time complexity is $O(Nm)$.

5.3.2 Complexity in memory:

We add to the previous data structure a list of group quotas for each school. Since the number of groups is less than the number of students, the space complexity is always $\Theta(Nm)$.

We also see here that both algorithms seem to guarantee that roughly three fourths of students get their first choice.

6 Task 5:

6.1 Discussing the implementation:

The basic idea is to keep sure that we do not add a person from a group in excess. We must also get sure that we add a person from an underrepresented group.

For this, we use two lists `lim_inf` and `lim_sup` that keep track of groups that have reached the minimum proportion limit or the maximum proportion limit respectively. When this is the case, we take the first adequate person in the ranking list give them the position and then move all the intermediary people by one downward. In the case where no more elements of a lacking group are to be found, we basically do not act.

We can show by recursion that for each k, the subset of students ranked k or better satisfies the $\frac{4}{5}$-rule up to one unit. Moreover, the change in the preference list is minimal due to the insertion of the needed person while keeping the other relative rankings. Since no precise definition of a "distance between rankings" is proposed, we cannot carry the analysis further.

6.2 Choice of data structure:

We have chosen, in respect with the structure of the class `school`, that the list of preferences is the ranking of students from the most preferred to the least. Proportions is represented by a list. Moreover,

the list `cpt` keeps track of the number of classified people in order to optimize the counting and save computation time.

6.3 Complexity estimation:

6.3.1 Time complexity:

At first glance, we might say that the time complexity is $(O(N(n+N))$, where n is the number of groups and N the number of students. However, $n \leq N$, therefore the complexity is $O(N^2)$.

In the worst case scenario, we will need to change the ranking at each iteration. The time complexity is $\Omega(N^2)$.

In conclusion, the time complexity is $\Theta(N^2)$.

6.3.2 Space complexity:

We have used 5 lists of length n or N. Therefore, the space complexity is $\Theta(N)$.

7 Task 7:

7.1 Discussing the implementation:

For memory efficiency and for avoiding useless search, we suppose that `feasibility` is a function that returns whether a demand set is feasible for a school or not.

In order to make the definition of the demand function faster, we will compute a table that yields for each pair (school s, student i) the value of p_s such that: $i = i^{(s,p_s)}$.

Moreover, we can observe that once we find a feasible demand for a school, it will be the demand returned by the algorithm.

7.2 Complexity estimation:

7.2.1 Time complexity:

Let us note $\mathcal{F}$ the worst case time complexity of the feasibility function when applied to our data.

Building the table is of time complexity $\Theta(|I||S|)$.

Each execution of the function `demand` is of time complexity $O(|I||S|)$.

Therefore, the time complexity of one call of `T` is $O(|I||S||S|\mathcal{F}) = O(|I||S|^2\mathcal{F})$, depending on whether or not we call `demand`.

Now, we must determine the number of times we must run `T` with a `demand` call. Since we are working with a general `feasibility` function, we can at least say that it is $O(|I||S|)$ times.

In conclusion, we can say that in worst cases, the complexity is $O(|I|^2|S|^3\mathcal{F})$.

7.2.2 Space complexity:

A `student` has space complexity of $\Theta(|\mathcal{G}| + |S|)$. A `school` has space complexity of $\Theta(|\mathcal{G}| + |I|)$. The arguments of the code of complexity $\Theta(|S||I| + |\mathcal{G}||S| + |I|)$, i.e. $O(|I|^2 + |I||S|)$. Since the remaining elements are only $O(|I||S|)$ pointers to the arguments, the total space complexity is $O(|I|^2 + |I||S|)$.

8 Task 8:

Here, the `feasibility` function tests simply if the number of students within the school is less than the school's quota. This operation can be done in constant time. The time complexity is $O(|I|^2|S|^3)$.

9 Task 9:

The `feasibility` function passes through the list of proposed students and counts their group. For the sake of optimization, since we return that the set of students is not feasible for the school when we want to count the student that exceeds their group quota. In the worst case scenario, `feasibility` takes at $O(|I|)$ time complexity. Hence, the overall time complexity is in the worst case $O(|I|^3|S|^3)$.
As we can see in the Part Results of this task, we observe a sharp decrease in the number of people getting their first choice since it went from nearly 75% to roughly 25%.

10 Task 10:

We will try to run the codes we previously run for testing the capacity constraints and the maximum quota constraint feasibility condition.

Here, the `feasibility` function counts the number of students from each group and tests whether we satisfy the $\frac{4}{5}$-rule. So the time complexity is $O(|I|^3|S|^3)$.

We suppose that the result will not be `individually rational`. It can be understood since many schools will end up without a student.

10.1 Test Instance 1:

When we apply the algorithm for the first instance, we see that only the 3^{rd} student gets a school (s_2). The condition of individual rationality is not satisfied.

10.2 Test Instance 2 and Instance 3:

For this instance, we observe a strange behavior. It seems that when we apply the algorithm of Task 7, at most one student gets a school. Of course, this is an "outrageous" violation of the individual rationality condition.

This property has been tested for a 1000 independent random situations as shown in the code.

11 Task 11:

Here we use an Integer Linear Programming algorithm. Since no algorithm has been proposed in the course and the implementation of the usual algorithms is complicated, we have chosen to use the library `pulp` of Python that implements the ILP.

The problem corresponds to the following model :

Maximize $c^T x$ subject to:

$$\begin{aligned} w^T x &\leq W_s \\ A^{(1)} x &\geq 0 \\ A^{(2)} x &\geq 0 \\ x &\geq 0 \\ x &\leq n \\ x &\in Z^{|\mathcal{G}|} \end{aligned}$$

where : x is the unknown column of the number of students from each group, w is the column of the costs of each group, $c = (1, ..., 1)^T$. The two last conditions assure positivity and integrality of the solution. n is the column of the numbers of students from each group.

The two matrices $A^{(1)}$ and $A^{(2)}$ assure the $\frac{4}{5}$-rule. Indeed, if we set $A^{(1)} = (\frac{6}{5}p_i - \delta_{i,j})_{1 \leq i,j \leq |\mathcal{G}|}$, we get the upper-bound condition. Similarly, setting $A^{(2)} = (\delta_{i,j} - \frac{4}{5}p_i)_{1 \leq i,j \leq |\mathcal{G}|}$ gives us the lower-bound condition.

Since we do not have access to the way the IPL is implemented, we cannot discuss its complexity or data structure.

12 Task 13:

Here, we see that we cannot use linear programming since the loss function $\sum_{i=1}^{|I|} l_i^2$ is quadratic. In the code, you can see that we used a greedy algorithm in order to find the optimum solution. The idea is to try to use the previous integer linear optimization for each ranking. In other words, we use the list `studs` which is the list of lists of students. Each list of `studs` contains the students who have ranked the school at the i^{th} position. In the same list, the students who have classed less schools are prioritized in order to avoid the $(|S| + 1)^2$ penalty. This prioritization is done when we *call* the function. Of course, this heuristic is far from being perfect. In particular, it does not take into account the value of the penalty.

As in the task 11, the analysis of data structure and time complexity cannot be done because we do not know the details of the ILP.

We have also thought about using an algorithm inspired by the Ford-Fulkerson algorithm. This algorithm would use a network flow with three layers: student, (group, school) and school. We can then give the edges going from the source to the students and from the students to the group-school layer as capacity the weight of their costs. We give to the edges going from the schools to the target a capacity of their budget constraint. However, crafting the right capacities between the group-school layer and the schools seems to be tricky. We know that we would need to use $\leq$ and $\geq$ conditions. It is not a problem in itself since the Ford-Fulkerson algorithm includes in itself the ≥ 0 condition.