Report of INF421 PI 2021 : Matching Under Constraints

CHHOUT Laychiva - NORNG Vannvatthana

February 7, 2022

1.1 Introduction

Overview: In this project, we study the many-to-one matching problems, where there are possible non-trivial constraints to be considered in the process of matching. In particular, we consider the terminology used to match Students to Schools.

Suppose that we have 2 non-empty sets, I is the set of all students and S is the set of all Schools, such that each $i \in I$ has a strict preference over the Schools ($\succ_i$) and being unmatched, and each $s \in S$ has a strict preference over the Students ($\succ_s$).

1.2 The differed acceptance algorithm and its variance

1.2.1 Capacity constraints

In this case, we consider that there is only the *capacity constraint*, that for each school s can only admit up to q_s students. Then, $\mathcal{F}_s = q_s$ for all $s \in S$.

Task 1: We know that the matching μ is said to be stable if it is feasible, individually rational, fair and non-wasteful.

1. Feasibility: μ is feasible if $\mu_s \in q_s$ for all $s \in S$.
2. Individually rational: μ is individually rational if $\mu_i \succ_i \emptyset$ for all $i \in I$, which means each student is willing to be matched to a school.
3. Fairness: a student i who has been matched to a school μ_i is happy and does not want to change a school, and school s is happy to have a student $i' \in \mu_s$.
4. Non-wastefulness: there is no pair $(i, s) \in I \times S$ such that $s \succ_i \mu_i$ and $\mu_s \cup \{i\}$ is feasible at s.

By these conditions, we see that it coincides with the standard notion from Gale-Shapley since it is a stable matching. School and Student are willing to be matched rather than being unmatched. A student who is matched to a school can change their school by doing unmatch and than match to a better preferred school. The same, if the school s has room for i, it can and should admit i. If s has fulfilled its capacity, it must check if i is a better choice than one of its already admitted students. If so, s replaces its least preferred admitted student with i. Intuitively, this matching satisfied everyone.

Task 2:

- Algorithm:
  - Begin by selecting a school s with open slots (or has not accepted a total of q_s students yet).
  - If there exist students whom s will match, s proposes to them in order of preference.
  - A student may accept a proposal or reject it.
  - If a student is matched with different school than the one proposing, the student must unmatch from its present school before accepting a proposal from a new school.
- Implementation:
  - The **makeProposals() method** is the **School** class is the most important method in this algorithm, a given school proposes to students one at a time in preference order until its capacity is full or runs out of students to which it can propose.

- The **acceptProposal()** method in the **Student** class handles this logic by checking if the Student want to match with the proposing School, and whether the proposing school is more preferable to its present school. A school that is unmatched may propose again to any remaining Students in its preference list.
  The **complexity** of this method is $0(n^2)$ where n is the number of student. Since, we used Array to store the list of preference and the method **remove()** costs $0(n)$.

1.2.2 Maximum quotas

In addition to the capacity constraint, we consider the maximum quotas q_s^g of each group $g \in \mathcal{G}$ in each school $s \in S$ such that $q_s^g \leqslant q_s$ for all g and $\sum_{g \in \mathcal{G}} q_s^g \geqslant q_s$.

Task 3:

- Algorithm:
  - Begin by selecting a school s with open slots.
  - If there exist students whom s will match, check the group of student and the s's maximum quota if the maximum quota is not filled, s proposes to them in order of preference.
  - A student may accept a proposal or reject it.
  - If a student is matched with different school than the one proposing, the student must unmatch from its present school before accepting a proposal from a new school.
- **Implementation**: We implement it in the same class as **Task2** by adding the constraint of maximum quota of each group before sending a proposal to student.
  The **complexity** of this method is $0(mn^2)$ where n is the number of student and m is the number of group which is small here.

1.2.3 Tests

The method *instance1Matching*, *instance3Matching*, *instance3Matching* allow us to do the matchings from Instance 1, Instance 2 and Instance 3 respectively with algorithm from task 2 and 3.

1. **Results for Instance 1**
   - **Task 2:** The method **instance1Matches** allows us to get the following result:
     - s_1 is matched with $[i_3, i_1]$
     - s_2 is matched with $[i_4, i_2]$
   - **Task 3:** The method **instance1Matches** allows us to get the following result:
     - s_1 is matched with $[i_3, i_1]$
     - s_2 is matched with $[i_4, i_2]$

     Observe that the result is the same because the quota of each group is 2 and the number of students is just 4.
     - We tested with 6 students, 3 of them (i_1, i_2, i_3) are in group A and other 3 (i_4, i_5, i_6) are in group B, we modify the capacity of each schools to 3 and the quota of group A is 1 and group B is 2. We follow the school preference list $\succ_{s_1}$: $i_6, i_5, i_4, i_3, i_2, i_1$ and we suppose that i_6 prefers s_1 and i_5 prefers s_2 Then we get the following result:
     - s_1 is matched with $[i_6, i_3]$
     - s_2 is matched with $[i_5, i_4, i_2]$

2. **Result for Instance 2**

   - **Task 2:** The method **instance2Matches** allows us to get the following result with random instance of $n = 20$:
     - s_1 is matched with $[i_{17}, i_{16}, i_{19}, i_5, i_6]$
     - $s2$ is matched with $[i_{12}, i_{15}, i_1, i_7, i_{13}]$

     Mean value: We do the simulation 200 times and 500 students, we get the following result
     - The average value of student from group A who is matched to his first choice is 158.0
     - The average value of student from group B who is matched to his first choice is 19.0
   - **Task 3:** The method **instance2Matches** allows us to get the following result with random instance of $n = 20$:
     - s_1 is matched with $[i_{13}, i_{19}, i_8, i_3, i_{17}]$
     - s_2 is matched with $[i_5, i_6, i_4, i_{18}, i_{20}]$

     Observe: the number of student in group B is matching list is always at most 2.
     We run a simulation for 100 times of matching and $n = 200$, we calculate the mean value:
     - The average value of student from group A who is matched to his first choice is 63.0
     - The average value of student from group B who is matched to his first choice is 7.0

     Observe: For n small, all the students of each group is more likely to get their first choice, while for n large there is more possibly that they won't get their first choice.

3. **Results for Instance 3**

   - **Task 2:** For the instance 3, it is the same instance 2 of task 2, since the group of student is not taken into account for task 2.
   - **Task 3:** The method **instance3Matches** allows us to get the following result with random instance of $n = 50$:
     - s_1 is matched with $[i_{15}, i_{48}, i_{13}, i_{21}, i_{41}, i_{35}, i_{50}, i_{30}, i_{45}, i_7, i_{46}, i_{43}]$
     - s_2 is matched with $[i_{29}, i_1, i_{34}, i_{33}, i_{19}, i_{23}, i_{10}, i_{11}, i_{49}, i_9, i_{36}, i_{25}]$

     Observe: With 4 different groups A, B, C, D, we notify the matching result is:
     - each schools matches with 12 students. $(int)(50/4)$

     We can run it with n large and we can verify the group's quota constaint.

1.3 Fair rankings

Task 5: Algorithm:

- For each rank k, the algorithm begins by counting the number of students (who are ranked k or better) in each group.
- For each group g that does not satisfy the 4/5-rule, let f_g be the fraction of students of g at rank k, and let p_g be the fraction of student of g in I.

- If $f_g < 0.8p_g$ up to one unit (i.e.$f_g < \lfloor 0.8p_g \rfloor$), the algorithm searches in the school preference list for the most prefer student i of rank $j \geq k+1$ that is in g, then i is moved to rank k and each student from rank k to $j-1$ moves down a rank (new rank = previous rank + 1).
- If $f_g > 1.2p_g$ up to one unit (i.e.$f_g > \lceil 1.2p_g \rceil$), the algorithm searches in the school preference list for the most prefer student i of rank $j \leq k$ that is in g, then i is moved to rank $k+1$ and each student from rank $j+1$ to $k+1$ moves up a rank (new rank = previous rank - 1).

The implementation of this algorithm is in the method **FairRanking** of the class **School**.

Complexity: $O(mn^2)$ where m is the number of groups and n is the number of students

Task 6:

1. **Instance 1:** With the method **instance1Matches**, we get the following result
   - After being modified by algorithm from Task 5, lists of preference of s_1 is [i_4, i_3, i_2, i_1] and s_2 is [i_4, i_3, i_2, i_1].
   - s_1 is matched with [i_3, i_1]
   - s_2 is matched with [i_4, i_2]

   From the observation, the lists of preference of s_1 and s_2 stay the same as the original lists since they already satisfied the 4/5-rule. In addition, the final matching satisfy the 4/5-rule from the modified lists.
2. **Instance 2:** The method **instance2Matches** gives random matching for each school each time the program executes. Given n = 20 students, we get one of the result as follow:
   - Students in each group: $A : \{i_1, \ldots, i_{18}\}$ and $B = \{i_{19}, i_{20}\}$
   - s_1 is matched with $[i_{18}, i_{11}, i_{16}, i_5, i_7]$
   - s_2 is matched with $[i_1, i_{10}, i_8, i_4, i_14]$

   From observation, the final matching for each School satisfies the 4/5-rule after the modification to the list of preference of each school using algorithm from Task 5. However, from another results as shown below, the final matching does not satisfy the 4/5-rule.
   - s_1 is matched with $[i_1, i_{16}, i_2, i_8, i_{14}]$
   - s_2 is matched with $[i_{11}, i_{19}, i_{20}, i_4, i_{12}]$

   We see that 2 students of group B are matched to s_2, while the lower bound and upper bound up to one unit are 0 and 1, respectively. Therefore, s_2 does not satisfy the 4/5-rule.
3. **Instance 3:** The method **instance3Matches** gives random matching for each school each time the program executes. Given n = 50 students, we get one of the results as follow:
   - Students in each group: $A = \{i_1, \ldots, i_{25}\}$, $B = \{i_{26}, \ldots i_{40}\}$, $C = \{i_{41}, \ldots, i_{47}\}$ and $D = \{i_{48}, i_{49}, i_{50}\}$
   - s_1 is matched with $[i_{50}, i_{30}, i_{10}, i_2, i_{27}, i_{19}, i_{44}, i_{22}, i_{43}, i_{49}, i_{16}, i_{39}]$
   - s_2 is matched with $[i_8, i_3, i_{38}, i_7, i_{35}, i_{33}, i_{12}, i_{32}, i_4, i_{24}, i_{45}, i_{26}]$

From the observation, there are 2 students in group D in the final matching for s_1, and since the lower bound up to one unit of group D is $12 \times 0.8 \times \frac{3}{50} = 0.576$ which is 0 and the upper bound up to one unit is $12 \times 1.2 \times \frac{3}{50} = 0.864$ which is 1. Therefore, the final matching does not satisfy the 4/5-rule.

In conclusion, after we modify the lists of preference of all schools to satisfy the 4/5-rule, the final matching will not guarantee to satisfy the same rule. But from our point of view, the modification of the preferences list gives us the better matching which is fair.

1.4 A fixed-point algorithm for arbitrary constraints

Task 7: The implementation of this algorithm is in class **FixedPoint.java**.

Task 8: Results of the test:

- The method **instance1Matches** in class **Task8** allows us to get the same result for the Instance 1 with only Capacity Constraint which is:
  - s_1 is matched with $[i_3, i_1]$
  - s_2 is matched with $[i_4, i_2]$
- The methods **instance2Matches** and **instance3Matches** (without maximum quota constraint) in class **Task8** gives us the randomly matching result with at most n students, which is the capacity of the school. Testing with $n = 50$, the results are:
  - Instance 2:
    - s_1 is matched with $[i_{12}, i_2, i_{40}, i_{18}, i_{24}, i_{25}, i_{20}, i_{39}, i_{49}, i_{30}, i_5]$
    - s_2 is matched with $[i_{50}, i_{34}, i_{15}, i_{43}, i_7, i_3, i_{32}, i_{44}, i_{22}, i_{42}, i_{17}, i_{11}]$
  - Instance 3:
    - s_1 is matched with $[i_{16}, i_{37}, i_7, i_1, i_{33}, i_{46}, i_{14}, i_{11}, i_{24}, i_{27}, i_{32}, i_{42}]$
    - s_2 is matched with $[i_3, i_{34}, i_{41}, i_{22}, i_{40}, i_{23}, i_{49}, i_{20}, i_2, i_{43}, i_9]$

  Observe: The matching list of s_1 from Instance 2 and s_2 from Instance 3 has dimension of 11, which is inferior of the capacity constraint of 12 because this algorithm drop the non-wastefulness properties. We can see that Differed Acceptance Algorithm in Task2 always guaranteed the maximum student in a school regarding to school's capacity.

We can conclude that the fixed-point algorithm for capacity constraints does not guarantee that the number of students matched to each school can reach the maximum of school's capacity unlike the Algorithm from Task2.

Task 9: Results of the test:

- Instance 1 : from the method **instance1Matches** in class **Task9**, we get the same result with the algorithm from Task 3 which is:
  - s_1 is matched with $[i_3, i_1]$
  - s_2 is matched with $[i_4, i_2]$
- The methods **instance2Matches** and **instance3Matches** (with maximum quota constraint) in class **Task9** give us the randomly matching results with at most $\lfloor n/4 \rfloor$ students for each school, which is the capacity of the school. The result is following which uses $n = 50$

- Instance 2:
  - s_1 is matched with $[i_{36}, i_{45}, i_{47}, i_{37}, i_{24}, i_{32}, i_{23}, i_{46}, i_{42}, i_{44}, i_{41}, i_{35}]$
  - s_2 is matched with $[i_{21}, i_{43}, i_8, i_{14}, i_{18}, i_2, i_{38}, i_{13}, i_6, i_{29}, i_{20}]$
- Instance 3:
  - s_1 is matched with $[i_{18}, i_7, i_3, i_{41}, i_{14}, i_{30}, i_{43}, i_{10}, i_{33}, i_{40}, i_{31}]$
  - s_2 is matched with $[i_{27}, i_4, i_{38}, i_{44}, i_{22}, i_6, i_{28}, i_{11}, i_{21}, i_{47}, i_1, i_{29}]$

Observe: The matching list of s_2 from Instance 2 and s_1 from Instance 3 has dimension of 11, which is inferior of the capacity constraint of 12 because this algorithm drop the non-wastefulness properties. We can see that Differed Acceptance Algorithm in Task3 always guarantees the maximum student in a school regarding to school's capacity.

We can conclude that the fixed-point algorithm for capacity and maximum quota constraints does not guarantee that the number of students matched to each school can reach the maximum of school's capacity.

Task 10: If we try to run the algorithm with the 4/5-rule constraint, the number of students matched to each school also does not guarantee the maximum of school's capacity, which is the same as constraints from Task 8 and 9 and the final matching is not satisfy the 4/5th-rule. However, from our point of view, it is better to run an algorithm with 4/5th constraint because it will give us the fair matching.

The class **Task10** shows the test result of each instance in Task 4 using the fixed-point algorithm with 4/5-rule constraint.

- Instance 1 : We always get the same results for instance 1
  - s_1 is matched with $[i_3, i_1]$
  - s_2 is matched with $[i_4, i_2]$
- Instance 2 and 3 : By using the method instance2Matches and instance3Matches, we get random matches for both schools for each instance, and given n = 50 students, one of the matches has the following result:
  - Instance 2:
    - s_1 is matched with $[i_{41}, i_{46}, i_1, i_{42}, i_{15}, i_{16}, i_{37}, i_{11}, i_{33}, i_{30}, i_{36}]$
    - s_2 is matched with $[i_6, i_{27}, i_{45}, i_{35}, i_{47}, i_{39}, i_7, i_9, i_{22}, i_4, i_{21}, i_{31}]$
  - Instance 3:
    - s_1 is matched with $[i_{41}, i_6, i_{15}, i_{20}, i_3, i_{31}, i_{12}, i_{16}, i_{10}, i_{46}, i_{17}, i_{38}]$
    - s_2 is matched with $[i_{27}, i_{50}, i_{19}, i_{22}, i_{29}, i_{42}, i_{18}, i_{28}, i_{14}, i_5, i_{49}, i_{39}]$