School Choice Problem

Jianfei ZHANG Yitao ZHANG

January 2017

1 Introduction

Stable matchings is a very active and successful branch of cooperative game theory commonly used to analyze two-sided markets and their mechanisms. It was originated by Gale and Shapley and further extended by Roth (and others). Their results revealed unknown properties of existing allocation mechanisms and provided a powerful methodology to design new mechanisms with a great impact on our daily lives. Roth and Shapley obtained the Nobel prize in Economics in 2012 "for the theory of stable allocations and the practice of market design".

In this project, we study three types of school allocation mechanisms. Each school has priorities for its candidates while each student either lists all his preferences for schools or partly lists his preferences. We will study some desirable properties, like feasibility, elimination of justified envy, Pareto efficiency or strategy-proof, which are important for players to accept.

2 Mechanisms and solutions of questions

2.1 Boston mechanism

Pseudocode of the implementation

```
input : The number of student $n$, the number of school $m$, students'
        preference matrix $n * m$, schools' preference matrix $m * n$,
        the number of available seats for every school $\{q\}_{i=1,\dots,m}$
output: The final choice of school for every student

Initialization;
for the $k-th$ round do
    for Every remaining student $i$ do
        choose their $k-th$ preferred school $S$ ;
        if $S$ still has available seat then
            add $i$ to the proposer list of $S$
        else
            add $i$ back to the remaining students set
        end
    end
    for Every school $S$ who got proposers do
        if The number of proposers is smaller than his available seats
         then
            accept all the proposers
        else
            only accept the students preferred
        end
    end
    if all the students have been allocated then
        break
    end
    $k = k + 1$;
end
```

In line $11-17$, for certain schools, the number of students who have chose it in this round may exceed their available seats, in this case, school choose their preferred students. In order to facilitate the comparison between these students, we use a data structure $Hashmap < School, PriorityQueue < Student >>$, which use the *PriorityQueue* to store the students who have chose this key *School*. The students who are preferred by the school have higher priority, so we create a new class *Student* which contain the student ID and also his preference order for a certain school.

In line $3-9$, each time when we take a student S from the remaining students, if his $k-th$ choice has no more seat, we add it back to the set for the next round, if not, we add it to the corresponding priority queue.

In terms of complexity of this algorithm, we run m round in maximum, each student take constant time to choose school, and each school take constant time to allocate a seat, we note that $add()$ method for a priority queue of n take $O(\log(n))$ time, so the complexity $< \Omega(m \log(n!))$

2.1.1 Question.1

We suppose that $(i, s\prime)$ and $(i\prime, s)$ are two matches generated by Boston mechanism where for candidate i, $k = rank(s) < rank(s\prime)$(i prefers school s) and for school s, $pr(i) < pr(i\prime)$(i has higher priority than $i\prime$).

In k-th round, s is already full, which means $pr(i)$ ¿ the remain places in s after (k-1)-th round. Whereas we have $(i\prime, s)$ and this means $i\prime$ is allocated to s in k-th round or before kth round. If $i\prime$ is allocated in k-th round, it must have higher priority in s than $i\prime$. But if $i\prime$ is allocated before k-th round, such a situation could happen: $i\prime$ has submitted a higher preference for s. At that corresponding round, s has enough places to accept $i\prime$ even though $i\prime\prime$s priority could be inferior to that of $i\prime$s. In one word, $i\prime$ could place a higher preference for a school he wants to enter but without a good priority, which could lead to unjustified envy.

Hence, Boston mechanism does not eliminate justified envy.

2.1.2 Question.2

The result of the implementation of Boston mechanism is: $s_1 : (i_1, i_5), s_2 : (i_2, i_8), s_3 : (i_3, i_4, i_7), s_4 : i_6$.

2.1.3 Question.3

We use an example to show that Boston Mechanism is not strategy-proof. If candidates submit their preferences honestly in such way and priorities of each school remain the same as the example offered:

$i_1 : s_2 \succ s_1 \succ s_3 \succ s_4$	$i_5 : s_1 \succ s_3 \succ s_4 \succ s_2$
$i_2 : s_1 \succ s_2 \succ s_3 \succ s_4$	$i_6 : s_2 \succ s_1 \succ s_4 \succ s_3$
$i_3 : s_2 \succ s_3 \succ s_1 \succ s_4$	$i_7 : s_2 \succ s_1 \succ s_3 \succ s_4$
$i_4 : s_3 \succ s_4 \succ s_1 \succ s_2$	$i_8 : s_1 \succ s_2 \succ s_4 \succ s_3$

After running Boston mechanism, the result is $s_1 : (i_2, i_5), s_2 : (i_3, i_7), s_3 : (i_1, i_4), s_4 : (i_6, i_8)$. Here i_1 prefers s_2 first and s_1 second, but i_1 has priority at s_1 but not at s_2. So i_1 choose to put s_1 before s_2 in his preference list, which gives: $s_1 : (i_1, i_2), s_2 : (i_3, i_7), s_3 : (i_4, i_5), s_4 : (i_6, i_8)$. i_1 could finally be admitted to his desired school by ”lying”. Hence, Boston mechanism is not strategy-proof.

2.2 Gale-Shapley Student Optimal Stable Mechanism (SOSM)

Pseudocode of the implementation

```
input  : The number of student $n$, the number of school $m$, students'
         preference matrix $n * m$, schools' preference matrix $m * n$,
         the number of available seats for every school $\{q\}_{i=1,\dots,m}$
output: The final choice of school for every student
1  Initialization;
2  while there are students non-allocated do
3      for every remaining student do
4          Propose to their next-choice $S$;
5          $S$ add it to his candidate list;
6      end
7      for every school who got proposers do
8          It consider all the new proposers with the accepted students
             before;
9          if the number of proposers is smaller than its available seats
             then
10             accept all the proposers
11         else
12             reject the students less preferred
13         end
14     end
15 end
```

For line $7-14$, similarly, we use a data structure $Hashmap < School, Priori tyQueue < Student >>$ to store all the proposers for a certain $School$, normally this mechanism need to compute more than the first mechanism because for each round, a school need not only to consider its new proposers, but also the old ones.

2.2.1 Question.4

We suppose that $(i, s\prime)$ and $(i\prime, s)$ are two matches where student i prefers school s to $s\prime$ and $pr(i) < pr(i\prime)$ for s. This means that i proposed to s but was rejected while $(i\prime, s)$ is a match. First case: s is the first choice of $i\prime$. Then clearly $pr(i\prime) < pr(i)$ for s, which contradicts to our hypothesis. Second case: s is not the first choice. In a certain step, $i\prime$ is chosen while i is not. This means $i\prime$ is more competitive than i in obtaining the remaining seat, which does not leads to a justified envy.

Hence, SOSM mechanism eliminates justified envy.

2.2.2 Question.5

The result of the implementation of SOSM mechanism on the test example is:$s_1 : (i_1, i_2), s_2 : (i_7, i_8), s_3 : (i_3, i_4, i_5), s_4 : i_6$.

2.2.3 Question.6

After implementing the test example, we obtain that: $s_1 : (i_1, i_2), s_2 : (i_7, i_8), s_3 : (i_3, i_4, i_5), s_4 : i_6$. This is a quite satisfying result: all candidates are accepted by their first or second choice. We assume that the preferences submitted by all candidates reveal their private information.

We see that i_1 is admitted by his second-best choice in the case. If i_1 decides to hide his private information and proposes $s_4 \succ s_1 \succ s_3 \succ s_2$. SOSM mechanism gives the result $s_1 : (i_2, i_5), s_2 : (i_7, i_8), s_3 : (i_3, i_4), s_4 : (i_1, i_6)$. He will be accepted "what he proposes". In other words, this example shows that under SOSM mechanism, one would receive a satisfying result if he submits his true private information. If not, he would probably be admitted to a school he does not like. This illustrates that SOSM mechanism is strategy-proof.

2.2.4 Question.7

We use the example of three schools and three student provided in the text. The result by SOSM mechanism is:$s_1 : i_1, s_2 : i_2, s_3 : i_3$. We can find a Pareto improvement such as:$s_1 : i_2, s_2 : i_1, s_3 : i_3$. We show that the stable matching obtained by SOSM is not Pareto efficient

2.3 Top Trading Cycles Mechanism (TTCM)

Pseudocode of the implementation

```
input  : The number of student $n$, the number of school $m$, students'
         preference matrix $n * m$, schools' preference matrix $m * n$,
         the number of available seats for every school $\{q\}_{i=1,\dots,m}$
output: The final choice of school for every student
1 Initialization;
2 while there are still students non-allocated do
3     Find all the circles according to the preference of the remaining
        schools and students;
4     Assign seats to the students in the circles;
5     Remove students who have been assigned from the non-assigned
        student set;
6     Update schools' remaining seats and remove the schools who have
        no more seats;
7 end
```

In this algorithm, the process of assignment and update is not difficult. We can look every school and every student as a node in a graph, in line 3 we need find all the circles in this graph, which is not evident. In order to facilitate the process of choosing school for some student and choosing student for some school, we use two data structure $Map < StudentID, List < SchoolID >>$and $Map < SchoolID, List < StudentID >>$, that's for some Student/School, the remaining list of School/Student that he can choose and which is ordered in decreasing priority. And we can see that these two Map include the graph information. We use a algorithm similar to Tarjan's strongly connected components algorithm here:

Findallcircles()

```
input  : Graph $g$
output: The students who are found in a circle and the school to
        which he points

1 for every school node $S$ do
2     if $S$ hasn't been read then
3         Find the circle containing $S$ if exist
4     end
5 end
```

The loop ensure that all the circles will be found. For every school node S, we search the strongly connected component (circle here) that contains S if S hasn't been read.

For finding the circle that contains a certain School, we use the following algorithm:

Search a circle containing School S if it exist

```
input  : School $S$, graph $g$, a table indicating if a node has been
         readed, Stack for storing the school node which can be
         reached from $S$
output: The pair of node [Student, School] in the circle containing $S$

1 if Stack doesn't contain School $S$ then
2     Push $S$ to the Stack;
3     Stunext = S.Nextnode;
4     Schnext = Stunext.Nextnode;
5     Search a circle containing School Schnext if it exist;
6 else  //this node is already in the Stack, so we can find a circle
7     Pop the elements in the Stack one by one until we get $S$;
8     Mark all the elements in the Stack that they have been read;
9 end
```

So we can see that we use recursion to store all the nodes which can be reached from S in a Stack, when we reach an element which is already contained in the Stack, we get a circle.

For the complexity of **Findallcircles()**, since we read all the school nodes only one time, we have the complexity $O(m)$ for this method.

2.3.1 Question.8

The result of the implementation of TTCM is:$s_1 : (i_2, i_5), s_2 : (i_1, i_7), s_3 : (i_3, i_4), s_4(i_6, i_8)$.