Raphaël Lescot / Victor Catteau

Programming project report

School Choice Problem

I. Boston mechanism

1) Lets consider the following example: we have three students i1, i2, i3 and i4 and three schools s1, s2 and s3. Their preference and priorities are:

i1: s1 > s2 > s3	s1 : i3 > i1 > i2
i2: s2 > s1 > s3	s2 : i2 > i3 > i1
i3: s2 > s1 > s3	s3 : i1 > i2 > i3

If we run the Boston mechanism we have:
Round 1: (i1, s1) matched, (i2, s2) matched, i3 rejected
Round 2: i3 rejected
Round 3: (i3,s3) matched

Therefore the pair (i3,s1) is unmatched, but the student i3 prefers s1 to his current assignment s3 and has a higher priority than i1 at school s1.

We can conclude that the **Boston mechanism doesn't eliminate justified envy**.

2) Pseudo-code of the algorithm:

Input : number *n* of students; number *m* of schools; array of size m *capacity* containing the capacity of each school; double array *priority* of length m*n such that priority[i] is a list of n students sorted in decreasing order of priority for school i; double array *preference* of length n*m such that preference[i] is a list of m schools sorted in decreasing order of preference for student i

Output: a double array *matching* representing the matching

```
k = 0;
while ( (there are free students) and (there are free seats in some schools) and (k < m) )
        Let school be a school that still has empty seats
                Let j = 1.
                while ( (there are free seats in school) and (j < n) )
                        student = priority[school][j] ;
                        if ((student is free) and (preference[student][k] == school)
                                assign a seat to student in school ;
                        j++ ;
k++ ;
```

Complexity= at most O(m*m*n)

(cf. source code for the detailed algorithm)

Result on the test-example:
(s1: i2, i5) ; (s2: i1, i8) ; (s3: i3, i4, i7) ; (s4: i6)

3) When we run the Boston mechanism algorithm on the test-exemple it gives us the following results:
Round 1: (i1,s2) ; (i2,s1) ; (i3,s3) ; (i4,s3) ; (i5,s1) ; (i6,s4) are matched
Round 2: (i8, s2) matched
Round 3: (i7, s3) matched

If we keep the same example but only change the preferences of i7 to: s2 > s1 > s3 > s4 : then we have the same results instead that in the final matching: (i7, s2) in round 1 and (i8, s3) in round 3.

Hence, if student i7 misrepresents his real preferences (given by the initial example) by putting them this way, he benefits (because in his real preferences, s2 > s3) from this strategy.

Therefore the **Boston mechanism isn't strategy proof.**

<u>II. Gale-Shapley Student Optimal Stable Mechanism</u>

4) *Lemma* : the worst priority of a student assigned to a school will decrease during the algorithm.

> *Proof* : a school accepts a student only if he has a better priority than a student which a which was already assigned.
>
> Let's assume that there is an unmatched pair (i,s) given by the algorithm such that i prefers s to his current assignment s' and he has a higher priority than a student j who was assigned to school s.
>
> <u>1st case</u>: i was never assigned to school s. Because i was finally assigned to s' and i prefers s to s', at a round i asked school s but was rejected. Hence, at this round, the student assigned to s with the worst priority (lets call him a) had a better priority than i. j was assigned to s after this round because if not he would have been replaced by i. Therefore, this contradicts the lemma because j has a worse priority than a.
>
> <u>2st case</u> : i was at a given round assigned to s. Hence j cannot have been assigned later to s because of the lemma. If j was assigned to s before i, he would have been rejected before i.
>
> We conclude that **SOSM eliminates justified envy**.

5) Pseudo-code of the algorithm:

Input : number *n* of students; number *m* of schools; array of size m *capacity* containing the capacity of each school; double array *priority* of length m*n such that priority[i] is a list of n students sorted in decreasing order of priority for school i; double array *preference* of length n*m such that preference[i] is a list of m schools sorted in decreasing order of preference for student i

Output: a double array *matching* representing the matching

```
initially each student's rank of school to ask is 0;
while (there are free students)
    Let student be a free student.
    school  = preference[student][student's rank of school to ask]
    if (school has available seats)
        Assign student to school
    else
        if (there is a student assigned to school who has a lowest priority for school
        than student)
            then he is excluded from school and student takes his place
        else
            student's rank of school to ask ++ ;
```

6) Let's consider the following example, where in the final matching, two students are happy with their schools and one is not.

s1: i1 > i3 > i2	i1: s1 > s2 > s3
s2: i2 > i1 > i3	i2: s2 > s1 > s3
s3: i2 > i1 > i3	i3: s2 > s1 > s3

The final matching is: (i1,s1), (i2,s2), (i3,s3)

There is no strategy for i3 which could improve his final school.

The real reason why **the SOSM algorithm is strategy proof** is that it gives the final choice to the schools and not the students. Whereas the « power » was given to the students in the Boston mechanism, in the SOSM algorithm, schools end up by assigning to their top priority students. The preferences of the students matter when for instance several schools are in competition for the same student : then, it is the school the most preferred by the student that finally assigns him a seat.

7) The final matching is : (i1, s1), (i2, s2), (i3,s3)
Let's consider another matching: (i1,s2), (i2,s1), (i3,s3)
i1 prefers s2 to s1 and i2 prefers s1 to s2, therefore i1 and i2 would rather switch their assignments.

Hence, **the SOSM algorithm is not Pareto efficient**.

8) Pseudo-code of the algorithm:

Input : number *n* of students; number *m* of schools; array of size m *capacity* containing the capacity of each school; double array *priority* of length m*n such that priority[i] is a list of n students sorted in decreasing order of priority for school i; double array *preference* of length n*m such that preference[i] is a list of m schools sorted in decreasing order of preference for student i

Output: a double array *matching* representing the matching.

```
while (there are free students)
    Let element be a free student and pointingList be an empty list.
    while (pointingList has no cycle)
        if (element is a student)
            Let school be element’s favorite school among the ones that still have
            empty seats
            element points to school in pointingList
            element = school
        else element is a school
            Let student be the free student that have the highest priority for
            element
            element points to student in pointingList
    while (pointingList is not empty)
        first = pointingList.remove()
        if (first is part of the cycle and is a student)
            Assign the student first to the school that it pointed to in the cycle
```