HW2 – Matching Markets

Deferred Acceptance and Market Clearing

<u>TA in charge – Oleg Zendel</u>

In this exercise you will construct some of the matching mechanisms you've learned in class.

The data sets you will use:

1. *preferences_{1..4}.csv* – columns are project numbers, rows are student ids'; the values in the table are the utilities. i.e. in the file *preferences_test.csv* the utility of student-1 for project-2 is 20.
2. *grades_{1..4}.csv* – average grades of students in Math and in CS.
3. *pairs_{1..4}.csv* – partition of students to pairs.

You should write your solution code only in the files hw2_part1.py, hw2_part2.py and hw2_comp.py. You can add or modify any code you like, as long as the functions that are called in the main.py file remain with the same parameters and return the objects in the format described in the function.

You can run the attached code files; the result will be correct for the '*test_mc*' datasets.

To test your code, run the main.py file on the Azure VM. Code files that will fail to run with the attached main.py file on the Azure VM, will result in failure in the submission.

Don't write solution code in main.py it is only for your tests; you shouldn't submit it.

Part 1 – 30 points

Implement and run Deferred Acceptance algorithm, where each project can be assigned to (at most) one student. The priority over students is according to their grade: projects with even index are math projects (use math grade as priority); project with odd index are CS projects (use CS grade).
Then compute the social welfare and the number of blocking pairs[1] for the resulting matching. When counting the blocking pairs, you should count all the possible blockings. i.e. if student 4 constitutes a blocking pair with student 7, and with student 10, it will be counted as 2 pairs {4,7} and {4,10}.

For this part you will have to implement the following functions:

run_deferred_acceptance(n): n – is the id of the dataset {1,2,3,4, *test*}. The function should implement the DA algorithm and return a dictionary with the matching, the format of the dictionary should be: {sid:pid} where sid-student id, pid-project id. We will refer to this matching as *'single'* matching.

count_blocking_pairs(matching_file, n): this function receives a matching file, in the format that was written by the **write_matching** function and *n*-id of the dataset. The function should reconstruct the matching based on the input file and return the number of blocking pairs (an integer).
You can see the *test* dataset files for an example. The given matching in the example output files that you received should result in 1 blocking pair (sid-2 with sid-4) for the *'coupled'* matching.

calc_total_welfare(matching_file, n): this function receives a matching file, in the format that was written by the **write_matching** function and n-id of the dataset. The function should reconstruct the matching based on the input file and return the value of total welfare (an integer).
You can see the *test* dataset files for an example. The given matching in the example output files that you received should result with welfare=73 for the

[1] Recall that a blocking pair is 2 matches (student1, project1) and (student2, project2) such that each student prefers the other project, and each project prefers the other student.

'*single'* matching and welfare=63 for the '*coupled*' matching.

Part 2 – 30 points

Merge each pair (according to the pairs_n.csv file) and run the algorithm again using the preferences and the priority of the student with the higher average grade (math + cs). Compute the welfare and the number of blocking pairs. Then compute the social welfare and the number of blocking pairs for the resulting matching.

For this part you will have to implement the following functions:

run_deferred_acceptance_for_pairs(n): The function should implement the DA algorithm, only this time for pairs. The function should return a dictionary with the matching, the format of the dictionary should be: {sid:pid} where sid-student id, pid-project id. This is the same format as in part 1, only that here each two students will have the same pid. We will refer to this matching as '*coupled'* matching.

count_blocking_pairs(matching_file, n): This is the same function as in part 1. Make sure you implement it to be compatible for both parts, you can use the names of the matching file to distinguish between '*single*' or *'coupled'* matchings.

calc_total_welfare(matching_file, n): This is the same function as in part 1. Make sure you implement it to be compatible for both parts, you can use the names of the matching file to distinguish between '*single*' or *'coupled'* matchings.

Part 3 – 30 + 5 points

Recall the market clearing algorithm:

```
1. Set all prices to 0
2. Construct $G_p$ from current prices $\boldsymbol{p}$
3. Check if $G_p$ has a perfect matching (if so, we are done)
4. Otherwise:
    a. find a **minimal** constricted set of buyers $B$
    b. Let $S = \Gamma(B)$
    c. All sellers $j$ in $S$ increase $p_j$ by 1
    d. If all prices (of all sellers) are $> 0$, decrease all prices by 1
    e. Go to (2)
```

Note that in step 4a we look for a *minimal* constricted set (this was not emphasized in class).

Implement and run Market Clearing Prices algorithm, where each project can be assigned to (at most) one student.
You may use existing python packages (make sure it exists on the Azure VM) for computing a perfect matching.
You may use brute-force search to find a minimal restricted set.

Compute the social welfare for the resulting matching.
We will refer to this task as *'mcp'* task.

For this part you will have to implement the following functions:

run_market_clearing(n): n – is the id of the dataset {1,2,3,4, *test*}. The function should implement the Market Clearing algorithm and return 2 dictionaries:

1. matching_dict – {sid:pid} a dictionary with the matching
2. prices_dict – {pid:price} a dictionary with the prices of the projects

where sid-student id, pid-project id.

calc_total_welfare(matching_file, n): this function receives a matching file, in the format that was written by the **write_matching** function and n-id of the dataset. The function should reconstruct the matching based on the input file and return the value of total welfare (an integer).
This should be the same function for all 3 parts, it computes the welfare based on the matching alone (no need to consider the prices)

You can see the *test_mc* dataset files for an example. The given matching in the example output files that you received should result with welfare=45 for the '*single'* matching and welfare=55 for the '*coupled*' matching and welfare=59 for the *'mcp'* matching.

You can find an example by looking at the files attached for *test_mc.*
Your code should be able to run and finish within a minute on 1-3 datasets.
5 bonus points will be given if your code will finish (every time – not in expectation) within a minute on dataset 4.
A description to a possible solution can be found in the course book:
https://www.cs.cornell.edu/home/kleinber/networks-book/networks-book-ch10.pdf
Pages 295-302

Competition – 15 points

Write a matching algorithm that matches pairs of students to projects. The algorithm should minimize the number of blocking pairs and maximize social welfare.

You should create one matching file for each dataset (1,2,3,4). You may use different methods for each file. As long as you can reproduce your solution.

The results will be calculated on all the given datasets, the scoring will be as follows:

- $0<x_1<5$ pts (points) for maximizing the welfare, summing on all 4 datasets.
- $0<x_2<5$ pts for minimizing the number of blocking pairs, summing on all 4 datasets.
- 2 - multiplier for the highest score among the previous two. E.g. $Score_{comp} = \min(x_1, x_2) + 2 \times \max(x_1, x_2)$

The points in this section will be calculated with respect to the rest of the submissions. Where the first place in each task (maximizing the welfare and minimizing the number of blocking pairs) will receive 5 points for the mentioned task, all the submissions ranked below the top result will receive less than 5 points according to the overall ranking per task, as stated in the heading – this part is a competition.

The submitted files should be written with the **write_matching(matching_dict, type, n)**: function; where type=*'competition'* and n will be the id of the dataset according to the dataset used for the matching. The format of the submitted file should be identical to the example file *matching_task_coupled_data_test.csv*

Implemented functions

This function will be used by the *Automated testing,* you should avoid committing any changes to it, nor using a different function for writing the matching file. Make sure that the output file has identical formatting as the example files. (*matching_task_single_data_test.csv, matching_task_coupled_data_test.csv, matching_task_mcp_data_test_mc.csv, mc_task_data_test_mc.csv*)

write_matching(matching_dict, type, n): this function is already implemented, it receives a matching dictionary, type {*'single', 'coupled', 'mcp'*} and n the id of the dataset. You should make sure that your files are in similar formatting as the files attached.

write_projects_prices(prices_dict, n): this function is already implemented, it receives a prices dictionary and n the id of the dataset. You should make sure that your files are in similar formatting as the files attached (*mc_task_data_test_mc.csv*).