Stable Marriage Problem Implementation Report

Titouan Pastor and Margot Bonafos

Supervised by
Jaklari Gentian

June 14, 2024

Contents

1 Introduction

1.1 Problem Statement

The stable marriage problem is a classic problem in graph theory and game theory. It involves finding a stable matching between two equally sized sets of elements, where stability means that there are no two elements that would prefer each other over their current partners. This project models this problem to assign students to schools, inspired by the French Parcoursup system, where schools can accept multiple students.

1.2 Team

This project was implemented by Titouan Pastor and Margot Bonafos under the guidance of Jaklari Gentian for a course in graph theory. Our goal was to explore and solve a real-world problem using theoretical concepts learned during the course.

2 Algorithm Description

We used the Gale-Shapley algorithm, also known as the deferred acceptance algorithm, to solve the stable marriage problem. The algorithm operates as follows:

- Each student proposes to their top-choice school.
- Each school tentatively accepts proposals up to its capacity and rejects the rest.
- Rejected students propose to their next choice, and schools reconsider the new proposals along with those tentatively accepted, again retaining up to their capacity.
- This process repeats until no more rejections occur.

In our adaptation, we modified the algorithm to handle the multiple capacities of schools. This means schools can tentatively accept up to n students where n is the school's capacity.

3 Implementation

The implementation was done in Python, utilizing the following key components:

- **Data Input:** Preferences for students and schools are read from an Excel (.xlsx) file, with each sheet representing a different scenario.
- **Data Structures:** Lists and dictionaries store preferences and current matchings.
- **Algorithm:** The modified Gale-Shapley algorithm is implemented to handle multiple capacities for schools.
- **Integrity Check:** A script checks the input data for consistency and correctness before running the algorithm.
- **Visualization:** Results are visualized using a graph where nodes represent students and schools, and edges represent matchings.

4 User Manual

To run the algorithm, follow these steps:

1. Ensure you have Python and the necessary libraries installed. Use the ‘requirements.txt‘ file to install dependencies:

```
pip install -r requirements.txt
```

2. Prepare the input Excel file with the correct format: each sheet should represent a different scenario with students' and schools' preferences.
3. Run the script by executing:

```
python parcoursup.py
```

4. The script will check the integrity of the Excel file. If errors are found, you will be prompted to correct them.
5. After successful validation, the algorithm will run and produce a stable matching. Results will be displayed graphically and saved to a file if specified.

5 Algorithm Results

We tested the algorithm on several scenarios, including:

- **Handmade Cases:** Manually created scenarios to test specific conditions and edge cases.
- **Random Cases:** Automatically generated scenarios to test the algorithm's robustness and performance.

The results demonstrated that the algorithm quickly finds stable solutions. For example, in scenarios with varying school capacities and student preferences, the algorithm consistently produced matchings where no student and school pair would prefer each other over their current assignments.

5.1 Case Study: Handmade Scenario

In one handmade scenario, we tested a case with three schools and six students, where:

- School A had a capacity of 2
- School B had a capacity of 3
- School C had a capacity of 1

The preferences were arranged to create potential conflicts and test the algorithm's ability to find a stable solution. The results showed that all schools filled their capacities without any unstable pairs.

5.2 Case Study: Random Scenario

In a random scenario with 10 schools and 50 students, each school's capacity was randomly set between 1 and 10. Preferences were randomly generated. The algorithm efficiently processed this larger dataset, resulting in a stable matching in under a second.

6 Graphs

We included graphical visualizations of the results. In these graphs:

- Students and schools are represented as nodes.
- Edges between nodes represent matchings.
- The graph allows for a clear visual verification of stability and completeness.

Figure 1: Example of stable matching graph

7 File Integrity

We developed an integrity check script to ensure the input Excel file is correctly formatted. The script checks:

- Non-zero capacities for schools.
- No duplicate preferences.
- Consistency in the number of students and schools.

If an error is detected, the user is informed to correct the data. This ensures that the algorithm runs smoothly and produces valid results.

8 Conclusion

Our implementation of the stable marriage problem, adapted for assigning students to schools with multiple capacities, demonstrates that the Gale-Shapley algorithm can be effectively extended for more complex scenarios. The consistent results and useful visualizations validate the effectiveness of our approach. This project highlights the practical application of theoretical concepts in solving real-world problems.