Project Allocation Portal

Subhash S, Sai Vamsi Alisetti, Albert Sunny

Indian Institute of Technology Palakkad

1 Introduction

Web Apps have been with us since early 2000's. However, with today's technology and opens source libraries, they have become an irreplaceable part of our day-to-day lives. Our initial motivations of making this portal was to reduce the drudgery people have to go through due to manual project allocation in this modern era where everything is code driven. We have seen that most of the project allocations are done using Google Docs or Excel — a tiresome and time-consuming process. So, we came up with a web-based solution that uses an algorithm to allocate projects after collecting preferences from students and mentors.

2 Overview

We have four positions namely Students, Faculty, Co-ordinator, Super Admin. Every individual falls into one of these four positions. Each *program* has a co-ordinator and is supported by a set of faculty members. For e.g., a program could be a UG program of CSE, or the PG program of EE.

- **Registration & Login**: On the first sign-in with institute email, we parse it and decide whether the user is a student or faculty, and redirect the user accordingly.
- **Super Admin**: They are typically faculty/staff from the academic section. They can view and delete projects, faculties and students. They are in-charge of adding information about *programs* into the portal. When creating programs they also have to map student roll numbers to programs using a regular expression. They can also assign the program co-ordinators.
- **Co-ordinator**: Each program will have a faculty as its project co-ordinator. They are responsible for setting the deadlines for various stages of the project allocation, and dissemination of information using the portal. They can also configure parameters such as number of projects per faculty, maximum intake per project, student cap per faculty, total number of students in the program. They can also view/delete students and faculties of the program they are coordinating.
- **Faculty**: A faculty can choose to support a program by adding it in their profile. Each faculty can add, edit, delete projects. They are also given the option to record their preference among students who have opted for their projects.
- **Students**: Student's page has a very minimalistic UI i.e., it is very intuitive to use. All the students have to do is when the appropriate stage comes to pass, they have to record their preference among the projects offered by the faculties supporting their program.

3 Allocation Algorithm

Let $\mathcal{S} = \{1, 2, \ldots, m\}$ and $\mathcal{P} = \{1, 2, \ldots, n\}$ represent the set of students and projects in a program. We note that it may be possible to accommodate multiple student in a project. Let $m_j \leq m$ denote that maximum number of students that can simultaneously work on project j. Each student $i \in \mathcal{S}$ has a preference for the projects in set $\mathcal{P}$. We denote this preference as a function $\sigma_i^S : \mathcal{P} \to \mathcal{P}$. Note that σ_i^S is a permutation of set $\mathcal{P}$, and $\arg\max_{j \in \mathcal{P}} \sigma_i^S(j)$ and $\arg\min_{j \in \mathcal{P}} \sigma_i^S(j)$ denote the most and least preferred projects by student i. Similarly, each project (its mentor) also has a preference of over the set of students $\mathcal{S}$. Preference of project $j \in \mathcal{P}$ is denoted by the function $\sigma_j^P : \mathcal{S} \to \mathcal{S}$. Note that σ_i^P is a permutation of set $\mathcal{S}$, and $\arg\max_{i \in \mathcal{S}} \sigma_j^P(i)$ and $\arg\min_{i \in \mathcal{P}} \sigma_j^P(i)$ denote the most and least preferred student by the mentor of project j. The permutations are obtained using the following approach:

- σ_i^S (*Students' preference*): In the second phase of project allocation, students are allowed to choose and order them in the decreasing order of preference. At the end of this phase, projects not chosen by a student are appended to his/her preference list. These newly added projects will be ordered in the increasing order of number students who have opted for these projects during phase two.
- σ_i^P (*Mentors' preference*): In the third phase of project allocation, project mentors are allowed to order (in decreasing order of preference) students who have opted for their projects. At the end of this phase, for each project, student who have not opted it are appended to the project's preference list. These newly added students will be ordered in the decreasing order of their GPA.

Now, given the sets $\mathcal{S}$ and $\mathcal{P}$, preferences $\{\sigma_i^S, i \in \mathcal{S}\}$ and $\{\sigma_j^S, j \in \mathcal{P}\}$. We are interested in obtaining an *allocation* $f : \mathcal{S} \to \mathcal{P}$. We note that a sufficient condition for the existence of an allocation is $\sum_{j=1}^{n} m_j \geq n$. We refer to an allocation as *unstable* if there exists two tuples (s_1, p_1) and (s_2, p_2), where $s_1, s_2 \in \mathcal{S}$ and $p_1, p_2 \in \mathcal{P}$, s_1 prefers p_2 over p_1, and p_2 prefers s_1 over s_2. We refer to allocations that are not unstable as *stable allocations*. To understand this concept better, let us consider the following example.

Example: Consider $\mathcal{S} = \{s_1, s_2, s_3\}$ and $\mathcal{P} = \{p_1, p_2, p_3\}$. Let $\sigma_1^S = \{p_2, p_1, p_3\}$, $\sigma_2^S = \{p_1, p_2, p_3\}$, $\sigma_3^S = \{p_3, p_2, p_1\}$, $\sigma_1^P = \{s_1, s_2, s_3\}$, $\sigma_2^P = \{s_3, s_2, s_1\}$, and $\sigma_3^P = \{s_2, s_1, s_3\}$. Let us consider the allocations $f_1 = \{(s_1, p_2), (s_2, p_1), (s_3, p_3)\}$ and $f_2 = \{(s_1, p_3), (s_2, p_2), (s_3, p_1)\}$. f_1 is a stable allocation because all students are allocated their first preference. In f_2, s_2 prefers p_1 over p_2 (project allocated to s_2) and p_1 prefers s_1 over its current allocation s_3. Thus, f_2 is an unstable allocation.

The well-known *Gale–Shapley algorithm* can be used to compute a stable allocation. In its most basic form, this algorithm takes as input equal numbers of two types of participants, and an ordering for each participant giving their preference for whom to be matched to among the participants of the other type. We use Algorithm 1, a Gale–Shapley type algorithm, to allocate students to projects. Let $\mathcal{S}_j$ denote the set of students allocated to project j. We note that $\cup_{j=1}^{n} \mathcal{S}_j = \mathcal{S}$. Here the set $\mathcal{S}$ is ordered by decreasing order of CGPA of each student. This ordering improves the performance of the algorithm which we shall see in the example that follows.

An example to illustrate the working of Algo. 1: Let $\mathcal{S} = \{s_1, s_2, s_3, s_4\}$ and $\mathcal{P} = \{p_1, p_2, p_3\}$. Let the GPA of students in $\mathcal{S}$ be $\{9, 8, 7, 6\}$. Let the student intake for projects in $\mathcal{P}$ be $\{1, 2, 1, 2\}$. Let the preferences be $\sigma_1^S = \{p_1, p_2, p_3, p_4\}$, $\sigma_2^S = \{p_1, p_2, p_4, p_3\}$, $\sigma_3^S = \{p_2, p_3, p_4, p_1\}$, $\sigma_4^S = \{p_2, p_4, p_3, p_1\}$ and $\sigma_1^P = \{s_3, s_4, s_2, s_1\}$, $\sigma_2^P = \{s_4, s_3, s_1, s_2\}$, $\sigma_3^P = \{s_4, s_1, s_2, s_3\}$, $\sigma_4^P = \{s_2, s_1, s_4, s_3\}$. We note that $\sum_{j=1}^{n} m_j = 1 + 2 + 1 + 2 = 6 > 4 = n$. Thus, a stable allocation exists. The steps executed in different iterations of the outermost loop of Algo. 1 are as follows

1. s_1 is chosen from $\mathcal{S}$ (highest GPA). Since $\sigma_1^S = \{p_1, p_2, p_3, p_4\}$, update $f = \{(s_1, p_1)\}$ and $\mathcal{S} = \mathcal{S} \setminus \{s_1\}$.
2. s_2 is chosen from $\mathcal{S}$. We note that $\sigma_2^S = \{p_1, p_2, p_4, p_3\}$ and $\sigma_1^P = \{s_3, s_4, s_2, s_1\}$. Since $m_1 = 1$, i.e., p_1 can only accommodate one student and s_2 is preferred over s_1 in σ_1^P, we update $f = \{(s_2, p_1)\}$, $\mathcal{S} = (\mathcal{S} \setminus \{s_2\}) \cup \{s_1\}$ and p_1 is removed from the preference list of s_1, i.e., update $\sigma_1^S = \{p_2, p_3, p_4\}$.
3. s_3 is chosen from $\mathcal{S}$. Since $\sigma_3^S = \{p_2, p_3, p_4, p_1\}$, update $f = \{(s_2, p_1), (s_3, p_2)\}$ and $\mathcal{S} = \mathcal{S} \setminus \{s_3\}$.

Algorithm 1: Allocation algorithm

set $\mathcal{P}_i = \{1, 2, \ldots, n\}$ for all $i \in \mathcal{S}$
set $\mathcal{S}_j = \emptyset$ for all $j \in \mathcal{P}$
while $\mathcal{S} \neq \emptyset$ **do**
  let i be the student with the highest GPA in $\mathcal{S}$
  let $j = \arg\max_{j' \in \mathcal{P}_i} \sigma_i^S(j')$ — most preferred project by student i from set $\mathcal{P}_i$
  if $|\mathcal{S}_j| < m_j$ **then**
    $\mathcal{S}_j = \mathcal{S}_j \cup \{i\}$
    $\mathcal{S} = \mathcal{S} \setminus \{i\}$
  else
    $k_{min} = \arg\min_{k \in \mathcal{S}_j} \sigma_j^P(k)$
    if $\sigma_j^P(i) > \sigma_j^P(k_{min})$ **then**
      $\mathcal{S}_j = \mathcal{S}_j \cup \{i\}$
      $\mathcal{S} = \mathcal{S} \setminus \{i\}$
      $\mathcal{P}_{k_{min}} = \mathcal{P}_{k_{min}} \setminus \{j\}$
      $\mathcal{S}_j = \mathcal{S}_j \cup \{k_{min}\}$
      $\mathcal{S} = \mathcal{S} \cup \{k_{min}\}$
    else
      $\mathcal{P}_i = \mathcal{P}_i \setminus \{j\}$
    end if
  end if
end while

4. s_4 is chosen from $\mathcal{S}$. Since $\sigma_4^S = \{p_2, p_4, p_3, p_1\}$, update $f = \{(s_2, p_1), (s_3, p_2), (s_4, p_2).\}$ and $\mathcal{S} = \mathcal{S} \setminus \{s_3\}$.

5. s_1 is chosen from $\mathcal{S}$. We note that $\sigma_1^S = \{p_2, p_3, p_4\}$. Thus, p_2 is now the most preferred project of s_1. Since $m_2 = 2$ and p_2 already has two students we have to check the preferences of p_2. We can see that s_2 is least preferred by p_2. So, no changes are made to the allocation, and we have $f = \{(s_2, p_1), (s_3, p_2), (s_4, p_2)\}$, $\mathcal{S} = \{s_1\}$, and p_2 is removed from the preference list of s_1, i.e., update $\sigma_1^S = \{p_3, p_4\}$.

6. s_1 is chosen from $\mathcal{S}$. We note that $\sigma_1^S = \{p_3, p_4\}$. Thus, p_3 is now the most preferred project of s_1. Since p_3 is not allocated to any student yet, we allocate s_1 to p_3. We update $f = \{(s_2, p_1), (s_3, p_2), (s_4, p_2), (s_1, p_3)\}$ and $\mathcal{S} = \emptyset$. Since $\mathcal{S}$ is empty, the algorithm terminates and returns the stable allocation f.

4 Technologies Used to Build a Complete End-to-End Solution

1. **MongoDB**: A cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with schema. MongoDB is developed by MongoDB Inc. and licensed under the Server Side Public License (SSPL).

2. **Javascript**: A lightweight, interpreted, or just-in-time compiled programming language with first-class functions. While it is most well-known as the scripting language for Web pages, many non-browser environments also use it, such as Node.js, Apache CouchDB and Adobe Acrobat.

3. **Node.js**: An open-source, cross-platform, JavaScript run-time environment that executes JavaScript code outside of a web browser. Node.js lets developers use JavaScript to write server-side script. Consequently, Node.js represents a "JavaScript everywhere" paradigm, unifying web-application development around a single programming language, rather than different languages for server and client-side scripts

4. **Express JS**: A minimal and flexible Node.js framework that provides a robust set of features for web and mobile applications.

5. **Angular**: A TypeScript-based open-source web application framework led by the Angular Team at Google and by a community of individuals and corporations.

5 Summary and Conclusions

We have tested the allocation algorithm with several tests cases, and it seems to work as expected. We stongly believe that the full-stack solution that we have developed will make academic project allocation an effortless and streamlined process. We also plan to enhance our solution by addressing bugs/suggestions that we receive after deploying the solution within our institute.

References

[1] D.Gale, L.S.Shapley, *College Admissions and the Stability of Marriage*,
Available at `https://www.eecs.harvard.edu/cs286r/courses/fall09/papers/galeshapley.pdf`.

[2] *Applications of Gale-Shapley*,
Available at `https://medium.com/@UofCalifornia/how-a-matchmaking-algorithm-saved-lives-2a65ac448698`.