COMP4801 Final Year Project

A Web-based Project Allocation System

Final Report

LEI Wan-hong, 3035202750
Department of Computer Science
The University of Hong Kong

supervised by
Dr. T. W. Chim

April 16, 2017
Abstract

This project presents a stable matching problem between students and projects called the Student-Project Allocation problem (SPA) where students can work collaboratively as a group. A matching between two sets students $S$ and projects $P$ is stable if there are no two pairs $(s,p)$ and $(s',p')$ such that $s$ prefers $p'$ to $p$ and $s'$ prefers $p$ to $p'$. Extensive researches have been carried out in relation to the stable matching problem and yet only few have addressed the SPA problem depicted in this project.

The SPA with group projects is studied in this project. An algorithm called the Spa-Group is designed which aims to generate the stable matching. In addition, a web application called the Project Allocation System (PAS) is developed to simulate the problem in an actual scenario. It employs the Spa-Group algorithm in the back-end to generate the stable matching. Users are allowed to modify their preference lists using this system.
Acknowledgements

I would like to thank my supervisor Dr. T. W. Chim for his guidance and support. He is always kind and friendly to us. This project would not be possible without his advice. I am also indebted to Ken for his comments and suggestions on the draft of the early version of this report. I learnt invaluable report writing and presentation skills from him.

I also want to express my gratitude to the University of Hong Kong, in particular, the Department of Computer Science, for their comfortable study environments which allow me to carry out this project over the past two semesters.
# Contents

List of Figures ........................................ iii
List of Tables ........................................ iv
List of Symbols ......................................... v
Abbreviations ........................................... vi

1 Introduction ........................................... 1

2 Background ........................................... 3
   2.1 Theoretical Aspect ................................ 4
      2.1.1 Basic Definition ................................. 4
      2.1.2 Preference Lists .................................. 5
      2.1.3 Matching ........................................... 6
      2.1.4 Stability ........................................... 7
   2.2 Related Work ....................................... 8

3 Objectives ........................................... 10
   3.1 The SPA-GROUP Algorithm ......................... 10
   3.2 The Project Allocation System .................... 10
      3.2.1 Scope ............................................. 10
   3.3 Goals ............................................... 11

4 Methodology ......................................... 12
   4.1 Software Development Process .................... 12
   4.2 Algorithm Design .................................. 12
   4.3 Technical Aspect of the PAS ...................... 12
      4.3.1 Back-end Technologies ........................ 13
# List of Figures

2.1 An example to illustrate (a) Unstable matching and (b) Stable matching, with their assigned projects or students boxed. ........................................ 3

2.2 Example to reduce the SPA with Group Projects into the SPA. ....... 9

4.1 A high-level system architecture diagram. $L_i$ represents the $i$-th layer. 13

5.1 PAS Index Page ................................................................. 23

5.2 The view of student list (a) before the matching and (b) after the matching is generated. ................................................................. 24

5.3 The view of a lecturer Diana (a) before the matching and (b) after the matching is generated. ................................................................. 25
List of Tables

2.1 Matching status for each project $p_j$ in $P$ and each lecturer $l_k$ in $L$. . . . . 7
4.1 Three core technologies used in the front-end in the PAS. . . . . . . . . . 14
5.1 List of Requests handled by the PAS. . . . . . . . . . . . . . . . . . . . . . 22
8.1 Schedule. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30
List of Symbols

\(G_i\) a set of students in a group.
\(L\) a set of \(m\) lecturers.
\(M\) a matching or an assignment.
\(O\) a set of \(r\) group.
\(P\) a set of \(q\) projects.
\(P_{li}\) a subset of projects \(P\) offered by lecturer \(l_i\).
\(R\) a set of Representative-Members tuples among \(r\) group.
\(S\) a set of \(n\) students.
\(\alpha_i^j\) preference list of student \(s_i\) in \(S\).
\(\beta_i^j\) preference list of lecturer \(l_i\) in \(L\) on project \(p_j\) in \(P_{li}\).
\(c_i\) project’s capacity constraint.
\(d_i\) lecturer’s capacity constraint.
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPA</td>
<td>Student-Project Allocation problem.</td>
</tr>
<tr>
<td>CDN</td>
<td>Content Delivery Network.</td>
</tr>
<tr>
<td>PAS</td>
<td>Project Allocation System.</td>
</tr>
<tr>
<td>RM-tuple</td>
<td>Representative-Members tuple.</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

In many universities, students are generally required to undertake a project before they graduate. Lecturers first propose one or more projects for students to choose from. Students can then take any one of these projects and are supervised by the corresponding lecturers. In addition, different students can work together as a group. Usually, these projects are not expected to be fully taken up by students whilst the number of these are generally sufficient for all students.

To reduce the chance of clashing between different students or lecturers, they are required to provide a finite number of projects or students that they are interested in. This list of ordering with priority is known as the preference list. The stable\(^1\) matching between students and projects is then left to be constructed using a centralised matching algorithm based on their preferences. It is important to realise that a centralised matching scheme is preferred than that of decentralised because of the property of stability \([5]\). A decentralised approach cannot guarantee the such property in general.

The problem depicted here is similar to that of the Student-Project Allocation problem (spa) which can be further generalised by the Two-sided Matching problem \([1]\). Researchers have been studying similar matching problems for decades. A study suggested two linear-time algorithms to solve the SPA without group projects \([1]\). However, only few studies have been done on the SPA with group projects \([2, 3]\).

This project aims to make two contributions. First, it introduces an algorithm called the Spa-Group to deal with the SPA with group projects. Second, it implements a web application called the Project Allocation System (PAS) to stimulate the matching problem. Students and lecturers are able to register and submit their preferences online.

\(^1\)Definition will be discussed in Section 2.1.
This goal of this system is to facilitate secretarial staffs to construct the stable matching.

The remainder of this report proceeds as follows. First, a precise definition of the SPA with group projects are presented. Next, the PAS is introduced with clear objectives and goals. Then the implementation details of the PAS and the methodology are followed. Current progress of the project is followed. Finally, difficulties and limitations are discussed and a tentative schedule is included.
Chapter 2

Background

Constructing the stable matching between students and projects can be done manually when both the size of students and projects are fairly small, possibly within 10. A straightforward approach to construct a matching is to individually match each student with an unallocated project that has a highest priority on the student’s preference list. This decentralised approach is known to be deficient and cannot guarantee the stability property [1]. The result generated by a decentralised matching scheme is thereby not stable.

Intuitively, the stability of a matching may be defined as follows [4]:

A matching $M$ is called stable if it does not exist two students $s_1$ with project $p_1$ and $s_2$ with projects $p_2$ that $s_1$ prefers $p_2$ to $p_1$ and $s_2$ prefers $p_1$ to $p_2$.

Figure 2.1 illustrates the difference between stable and unstable matching. Suppose lecturer $l_1$ offers project $p_1$ and that of $l_2$ offers project $p_2$. Student $s_1$ has a preference list of $\alpha_1 = \langle p_2, p_1 \rangle$ such that $p_2$ is more preferable than $p_1$. This priority, as defined, applies to preference list of students or lecturers.

Figure 2.1: An example to illustrate (a) Unstable matching and (b) Stable matching, with their assigned projects or students boxed.
An unstable matching $N$ happens when $N = \{(s_1, p_1), (s_2, p_2)\}$ (see Figure 2.1(a)). When such a pair exists, a stable matching $M = \{(s_1, p_2), (s_2, p_1)\}$ (see Figure 2.1(b)) can always obtain by swapping their projects. This stable matching is better than that of the unstable because it prevents a “private” re-assignment, that is, $s_1$ and $s_2$ agree to swap their projects given an unstable matching $N$.

In addition, when substantial number of students and projects are required to be matched, constructing such a matching manually can take up to few months or even impossible. A software system that can generate the stable matching is therefore desperately needed.

2.1 Theoretical Aspect

2.1.1 Basic Definition

An instance of the SPA with group projects can be defined as follows. Let

- $S = \{s_1, s_2, \ldots, s_n\}$ be the set of $n$ students,
- $L = \{l_1, l_2, \ldots, l_m\}$ be the set of $m$ lecturers,
- $P = \{p_1, p_2, \ldots, p_q\}$ be the set of $q$ projects and
- $O = \{G_1, G_2, \ldots, G_r\}$ be the set of $r$ group.

Each lecturer $l_i$ offers a subset of project $P_i$, which is a partition of $P$. Assume each project is uniquely offered by one lecturer, the disjoint union of $P_{l_1}, \ldots, P_{l_m}$ is then $P$. Each project $p_i$ has a capacity constraint $c_i$ which limits the maximum number of students in the project. Likewise, each lecturer $l_i$ has a capacity constraint $d_i$ which limits the maximum number of students that the lecturer can supervise.

It follow that each group $G_i$ in $O$ is a subset of students $S$ such that each student $s_k$ in $S$ is uniquely existed in $O$. Specifically, if $s$ in $G_i$ then $s$ will not in $O \setminus G_i$. This means a student cannot belong to 2 different group. A single student $s$ can also in a group $G_i = \{s\}$. The cardinality of each group $|G_i|$, or the group capacity, is the total number of members in the group $G_i$. It follows that a set of $r$ group $O$ can be expressed as

$$O = \bigcup_{G_i} \left\{ \begin{array}{c}
\{s_{1,1}, \ldots, s_{1,|G_i|}\} \\
\{s_{2,1}, \ldots, s_{2,|G_i|}\} \\
\vdots \\
\{s_{r,1}, \ldots, s_{r,|G_i|}\}
\end{array} \right\},$$

4
where each \( s_{i,j} \) is the \( j \)-th student in the \( i \)-th group \( G_i \). A *representative* of each group \( G_i \) is elected among themselves who delegates the group \( G_i \). Without loss of generality, the first student \( s_{i,1} \) in each group \( G_i \) is assumed to be the representative of the group \( G_i \). A *Representative-Members tuple* (RM-tuple) \( r_i \) for each group \( G_i \) is defined as

\[
r_i = (\text{representative of } G_i, \text{ group capacity of } G_i).
\]

Furthermore, a set of RM-tuples \( R \) can then be constructed as follows.

\[
R = \left\{ (s_{1,1}, |G_1|), (s_{2,1}, |G_2|), \ldots, (s_{r,1}, |G_r|) \right\} = \{ r_1, r_2, \ldots, r_r \}.
\]

Each group \( G_i \) is formed for a single project only. Each lecturer \( l_k \) only needs to rank the RM-tuple of group. This provides necessary information to construct a matching. Details are given in the next subsection.

### 2.1.2 Preference Lists

Each student \( s_i \) has a preference list \( \alpha_{i}^j \) for each project \( p_j \). This list contains a sequence of group; that is, a subset of \( O \). The student \( s_i \) will then rank each project in a sequence in the preference list \( \alpha_i \). A project \( p \) is said the to be *acceptable* if it is in \( s_i \)'s preference list; that is \( p \) in \( \alpha_i \). For example, if student \( s_1 \) finds project \( p_1 \) and \( p_2 \) acceptable and prefers \( p_2 \) over \( p_1 \), then \( s_i \)'s preference list \( \alpha_1 \) is

\[
\alpha_1 = \langle p_2, p_1 \rangle.
\]

In addition, for project \( p_1 \), \( s_1 \) prefers group \( G_3 \) than that of \( G_2 \); and for \( p_2 \), \( s_1 \) prefers group \( G_1 \) than that of \( G_3 \). It follows that

\[
\alpha_1^1 = \langle G_3, G_2 \rangle \quad \text{and} \quad \alpha_1^2 = \langle G_1, G_3 \rangle.
\]

In here, it is assumed that a group \( G \) cannot enrol a project \( p_i \) that has a smaller capacity than their group capacity; that is \( c_i < |G| \). In such a case, the project will not have enough capacity to admit the entire group and this pair cannot be matched.

Similarly, each lecturer \( l_i \) has a preference list \( \beta_{i}^j \) for each project \( p_j \) in \( P_i \) (i.e. offered project) in a sequence. The lecturer \( l_k \) will then each project in a sequence in the preference list \( \beta_k \).
2.1.3 Matching

A matching or an assignment $M$ is a subset of $R \times P$ such that:

1. If $(r_i, p_j)$ belongs to $M$, then $p_j$ is in $s_{i,1}$ preference list;
2. For each group $G_i$, at most one project $p_j$ is matched to $G_i$ in $M$;
3. For each project $p_i$ in $P$,
\[ \sum_{\forall (r_k, p_j) \in M} |G_k| \leq c_i, \]
that is, the total number of students in all group matched must not be greater than $c_i$; and
4. For each lecturer $l_i$ in $L$, for all $p_j$ in $P_{l_i}$
\[ \sum_{\forall (r_k, p_j) \in M} |G_k| \leq d_i, \]
that is, the total number of students in all group matched must not be greater than $d_i$.

Then $(r_i, p_j)$ means that all students in the group $G_i$ are assigned to $p_j$. If $p_j$ is offered by $l_k$ then $l_k$ supervises all students in the group $G_i$.

It follows that the matching status of $p_j$ in the matching $M$ is defined as:

**Definition 1.** If $M \in R \times P$, then
\[ M(p_j) = \{ G_i \in S \mid (r_i, p_j) \in M \} \]
is called the matching status of $p_j$ in $M$ and is denoted by $M(p_j)$.

Likewise, the matching status of $l_k$ in $M$ is defined as:

**Definition 2.** If $M \in R \times P$, then
\[ M(l_k) = \{ G_i \in S \mid (r_i, p_j) \in M \text{ and } p_j \in P_{l_k} \} \]
is called the matching status of $l_k$ in the matching $M$ and is denoted by $M(l_k)$. 

6
It follows that the matching status of projects or lecturers is the same as in Table 2.1. Essentially, the status is summing all students from all group matched instead of all students matched.

It follows that project $p_i$ is under-subscribed, full or over-subscribed if the total number of students in all group assigned to $p_i$ is less than, equal to or greater than the project capacity $c_i$ [1]. This applies to lecturer $l_k$ with capacity $d_k$ who has assigned to $M(l_k)$. [5]. Table 2.1 outlines all possible matching status for each project $p_j$ in $P$ and each lecturer $l_k$ in $L$. The criteria for different matching status are tabulated with the corresponding status in the column title. It is clear that for any legitimate matching, every project and every lecturer should not be over-subscribed.

### 2.1.4 Stability

The stability of a matching can be established rigorously using the concept of blocking pair. This is an based on a research that studied SPA without group projects [1].

**Definition 3.** A pair $(r_i, p_j) \in R \times P$, where $p_j$ is offered by $l_k$, blocks the matching $M$ if

1. $r_i$’s preference list contains $p_j$;
2. either $r_i$ is not in $M$ or $r_i$ prefers $p_j$ to their current assignment; and
3. either
   
   (a) $p_j$ is under-subscribed and $l_k$ has enough capacity to admit group $G_i$, that is, $|G_i| \leq |M(p_j)|$ and $|G_i| \leq |M(l_k)|$; or
   
   (b) $p_j$ is under-subscribed with $|G_i| \leq |M(p_j)|$ but $l_k$ is full, and if $r_i$ is assigned to a different project offered by $l_k$ or $l_k$ prefers $r_i$ to the worst group matched in $M(l_k)$ say $r'$ with $|G_i| \leq |G'|$; or
(c) $p_j$ is full and $l_k$ prefers $r_i$ to the worst group matched in $M(p_j)$ say $r'$ with $|G_i| \leq |G'|$.

Then $(r_i, p_j)$ forms a blocking pair.

For a matching to be stable, it should not admit any blocking pair. The correctness of any stable matching algorithm depends on this definition.

Conceivably, this definition of blocking pair try to capture every possible pair of assignment $(r_i, p_j)$ that can possibly improve the current matching $M$. First, $r_i$ must find $p_j$ acceptable (Condition 1). Second, $r_i$ cannot be assigned to a project say $p'$ in the matching, that is $(r_i, p') \notin M'$, or otherwise $r_i$ must prefer $p_j$ to $p'$ or $r_i$ would just stay with project $p'$ (Condition 2). Now, for lecturer $l_k$ to be matched with $r_i$, it must be either (1) both $p_j$ and $l_k$ are under-subscribed and have enough capacity to admit the group $G_i$ (Condition 3(a)); or (2) $p_j$ is under-subscribed and has capacity to admit $G_i$ but lecturer $l_k$ is full. In such a case, $l_k$ must be either (i) assigned to $G_i$ with different project $p'$ so that $l_k$ can switch the project to $p_j$ or (ii) $l_k$ prefers $r_i$ to the worst group $r'$ in $l_k$’s current matching. The group capacity of $G_i$ must not be larger than the worst group $G'$ so that $l_k$ can admit the new group $G_i$ (Condition 3(b)); or (3) $p_j$ is full and it happens that $l_k$ prefers $G_i$ to the worst group say $G'$ that is assigned to $p_j$. In such a case, $l_k$ can reject $G'$ and admit $G_i$, provided that the group capacity of $G_i$ is less than or equal to $G'$ (Condition 3(c)).

There are potentially many stable matchings existed in a SPA instance. However the optimal one is preferred among all these matchings [4].

**Definition 4.** A stable matching is called optimal if every student is at least as well off in this matching as in any other stable matchings.

This definition says that a stable matching is optimal if and only if it matches each student with a project that has a highest priority on the its preference list than that of the other projects among other stable matchings. This optimal matching is unique if it is existed [4].

### 2.2 Related Work

There are two linear-time algorithms, SPA-STUDENT and SPA-LECTURER, suggested to solve the original SPA without group projects using appropriate data structures [1]. Conceivably, either students or lecturers are allowed to reject their temporary
assignments in order to subsequently allocate better assignments. Therefore the SPA-STUDENT algorithm generates a student-oriented matching but the SPA-LECTURER algorithm generates a lecturer-oriented matching. However, this paper defined lecturers’ preference lists differently. Each lecturer \( l_k \) has a single preference list which contains all students who are enrolled in \( l_k \)'s projects. All students in all projects are then rank accordingly. This allows lecturers to rank their students once only.

Figure 2.2 illustrates an example of reduction from the SPA without group projects to the new SPA with group projects. There are 6 students \( S = \{s_1 \cdots s_6\} \) with 3 groups. Group \( G_1 \) has students \( s_1, s_2, s_3 \), group \( G_2 \) has students \( s_4, s_5 \) and group \( G_3 \) has student \( s_6 \) only. The RM-tuple of \( G_1 \) is therefore \( r_1 = (s_1, 3) \), and \( G_2 \) is \( r_2 = (s_4, 2) \), and \( G_3 \) is \( r_3 = (s_6, 1) \). The SPA-GROUP will then generate a stable matching \( M \) from the preferences and the RM-tuples.

Another research studied a SPA problem with grouping [3]. But the preference list of lecturers is absent. Instead, the relationship between students and projects is determined by a weighting factor. They proved that such problem is NP-COMPLETE unless each project has a capacity smaller than or equal to 2, that is, \( c_i \leq 2 \) for all \( p_i \) in \( P \). An approximation algorithm is suggested to tackle this problem in general. A linear programming technique is also suggested by a study [2].
Chapter 3

Objectives

This project has two main objectives. First, it aims to design an enhanced algorithm called the Spa-GROUP to solve the Student-Project Allocation problem (SPA) with group projects. This algorithm generates stable matching based the constraints stipulated in Section 2.1. Second, it aims to implement a web application called the Project Allocation System (PAS). This web-based system utilises the enhanced algorithm to generate the stable matching.

3.1 The Spa-GROUP Algorithm

The Spa-GROUP algorithm should contain following properties:

1. Stable matching. The matching generated should be stable Lecturer-oriented matching. Only lecturers are allowed to reject their temporary assignments.

3.2 The Project Allocation System

3.2.1 Scope

The PAS should contain following major functions:

1. Modify Preference Users are allowed to modify their preference lists through the system.

2. Generate the stable matching. After the system collected preference lists from all users, the stable matching is generated.
3. *Administrator panel.* For administrator to manage the system. One major function is to approve generated matching.

### 3.3 Goals

The ultimate goal of this project is to improve the efficiency in constructing the stable matching. It is a tedious and painstaking task to construct this matching manually. The web application PAS is built to achieve this goal. By utilising the web nature of the system, the time and cost in collecting user’s preferences can be greatly reduced. The stable matching between students and projects can then be easily generated. In addition, the SPA-GROUP algorithm allows different students work together as a group while maintaining the stability of the matching.
Chapter 4

Methodology

4.1 Software Development Process

An unified software development process is used in this project. The advantage of this process is that error can be readily rectified even at the later phase of the project. The entire development process is divided into three distinct phases where each phase contains iterative and incremental cycles. Changes are constantly made in each cycle and followed by a testing. This can accelerate the development process. The three phases are Inception, Elaboration and Construction phases. The details of each phase are given in Table 8.1.

4.2 Algorithm Design

The Spa-Group algorithm may be designed based on the Spa-Lectuer algorithm which solves the spa without group projects [1]. The pseduocode of Spa-Lectuer is attached to the Appendix A. It has been proved that the generated is optimal by a rigorous proof on its correctness. The Spa-Group algorithm is expected to be based on the Spa-Lecturer algorithm upon modification.

4.3 Technical Aspect of the PAS

Since the Project Allocation System (PAS) is a web application, both web server and database management system are necessary. A high-level system architecture diagram is illustrated in Figure 4.1. The system consist of front-end and back-end (see the left
hand side of Figure 4.1) and is categorised into 5 layers (see the right hand side of Figure 4.1). All software used in this project are free and open source.

- Front-end: Project Allocation System (PAS)
  - NGINX
  - Django

- Back-end: Proxy to Django using uWSGI module
  - SQLite
  - GNU/Linux
  - Hardware

$L_i$ represents the $i$-th layer.

4.3.1 Back-end Technologies

The details of each layer are discussed below. Each layer can be referred to the right hand side of Figure 4.1.

$L_0$: Hardware Layer

This layer contains necessary hardware for the PAS to deploy.

$L_1$: Operating System

The PAS will deploy under the GNU/Linux operating system which operated above the $L_0$: Hardware Layer. This offers a robust and security platform for software to run.

$L_2$: Database System

SQLite is chosen as the database system. It is an object-relational database system (ORDS) that is self-contained and cross-platform.

$L_3$: Web Server

The PAS uses the Django web framework as its back-end system. The scripting language for the Django is Python. With comprehensive built-in functions in Django, it provides a rapid development environment.
To handle users’ requests, Nginx is chosen for the web server. It is a lightweight and yet a highly scalable server compared with Apache. By adding the uWSGI module, it serves as a proxy server to redirect requests to the Django web framework so as to provide dynamic web contents. Whenever there is a request sent from the user, Nginx will redirect the request to Django through the uWSGI module.

Together with the PostgreSQL database system, the performance of the PAS will outperform the traditional LAMP stack which comprises GNU/Linux, Apache web server, MySQL database system and PHP.

### 4.3.2 Front-end Technologies

The front-end comprises of the web application PAS which operated in the $L_4$: Application Layer (see Figure 4.1). There are three core web technologies used in this layer. These technologies are HTML, CSS and JavaScript. Each following technology has following function on the PAS:

<table>
<thead>
<tr>
<th>Technology</th>
<th>Full name</th>
<th>Major functions on a web page</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTML</td>
<td>Hypertext Markup Language</td>
<td>Outline the structure of a web page, it serves as the building blocks</td>
</tr>
<tr>
<td>CSS</td>
<td>Cascading Style Sheets</td>
<td>Set the visual style to provide a better user interface</td>
</tr>
<tr>
<td>AngularJS</td>
<td>N/A</td>
<td>Provide interactive elements. Bind data from the JSON to the web page</td>
</tr>
</tbody>
</table>

Table 4.1: Three core technologies used in the front-end in the PAS.
Chapter 5

Deliverables

At present, a SPA-LECTURER prototype has been implemented to solve the Student-Project Allocation problem (SPA) without group projects. The aim of the prototype is to provide a pragmatic solution for the SPA without group projects. In addition, a preliminary draft on the SPA-GROUP algorithm has been done. It is based on the existing SPA-LECTURER algorithm.

A generator is also developed to generate random data set for testing. Currently, matching size up to ten thousand has been tested and the result is stable.

5.1 The SPA-GROUP Algorithm

The pseudocode of the SPA-GROUP algorithm is listed in Algorithm 1. This is largely contingent upon the SPA-LECTURER algorithm [1].

The algorithm takes an instance $I$ of SPA with group projects as an input. Students and lecturers are initialised as free which mean no project has been matched to them. The matching $M \in R \times P$ is also initialised as an empty set. The algorithm then continue search for a lecturer $l_k$ with an under-subscribed project $p_j$ in $P_{l_k}$ (i.e. offered project $p_j$ is under-subscribed). If such a project $p_j$ exists for lecturer $l_k$, a group $r_i = (s_{i,1}, |G_i|)$ has following conditions in order to match with $p_j$:

1. on the $l_k$’s preference list on project $p_j$ (i.e. $r_i$ in $\beta^j_k$), the group $G_i$ must be preferred $p_j$ to their current temporary assignment, if any; and

2. on the $s_{i,1}$ preference list $\alpha_{i,1}$, $p_j$ must be the first such under-subscribed project; and
3. the group capacity $|G_i|$ must not be larger than the current current $l_k$ availability, that is

$$|G_i| < (d_k - |M(l_k)|)$$

such that $l_k$ can supervise all students from group $G_i$. The $d_k$ is the capacity of lecturer $l_k$ and $|M(l_k)|$ gives the total number of students that $l_k$ is currently supervising.

**Algorithm 1** The pseudocode of the SPA-GROUP.

```plaintext
1: procedure SPA-GROUP(I)
2: assign each student, project and lecturer to be free
3: $M \leftarrow \emptyset$ \quad \triangleright \text{the matching result}
4: while there is a lecturer $l_k$ with a project $p_j \in P_{l_k}$ under-subscribed do
5: $r_i \leftarrow$ first such RM-tuple on $l_k$’s preference list on project $p_j$
6: $p_j \leftarrow$ first such project on $s_{i,1}$’s preference list
7: if $|G_i| > (d_k - |M(l_k)|)$ then \quad \triangleright $|G_i|$ larger than $l_k$ capacity
8: continue
9: end if
10: if $(r_i, p_j) \not\in M$ then
11: if $s \in G_i$ but $s \in G'$ such that $(r', p) \in M$ then
12: continue \quad \triangleright s has been matched
13: end if
14: if $s_{i,1}$ is assigned to a project $p$ then
15: $M \leftarrow M \setminus \{(r_i, p)\}$
16: end if
17: $M \leftarrow M \cup \{(r_i, p_j)\}$ \quad \triangleright l_k offers $p_j$ to $G_i$
18: for all successor $p_l$ of $p_j$ on $\alpha_{i,1}$ do
19: delete $p_l$ on $\alpha_{i,1}$ and $\beta_d$ \quad \triangleright if $p_l$ is offered by $l_d$
20: end for
21: for all group $G_i$ involved do
22: remove $G_i$ on students’ preference lists
23: end for
24: else
25: continue \quad \triangleright next lecturer or project
26: end if
27: end while
28: return $M \in R \times P$
29: end procedure
```

If the condition 3 fails, then the algorithm will remove the RM-tuple $(s_{i,1}, G_i)$ from $l_k$’s preference list $\beta^i_k$ and continue search for next group $(s_{i+1,1}, G_{i+1})$ on the list. These
criteria guarantee the optimality of a matching. When such a group $G_i$ is found, the algorithm will break the current assignment of $G_i$, if any, such that $p_j$ offered by $l_k$ can be assigned to $G_i$. It follows that for each projects $p_l$ such that $s_{i,1}$ prefers $p_j$ to $p_l$ is removed from $s_{i,1}$’s preference list $\alpha_{i,1}$. If $p_l$ is offered by $l_d$, then $p_l$ is also removed from $l_d$’s preference list $\beta_{d,l}$. It is because both of them will not prefer a project (or a student) that has a lower preference than their current assignment. The algorithm terminates when no $l_k$, $p_j$ and $G_i$ can be found.

Note that since the algorithm is lecturer-optimal, the students’ preferences over projects are not considered; that is $\alpha_i$ for each student $s_i$ is irrelevant in generating a lecturer-optimal stable matching. Students are in fact not necessary provide their preference over projects as long as the group has preference.

\subsection{Complexity}

The SPA-GROUP algorithm iterates through the lecturer’s preference list $\beta_{j,k}$ for each lecturer $l_k$ for project $p_j$. Let the total length of lecturers’ preference list be $n$. Now, during each iteration when a lecturer offers a project $p_j$ to group $G_i$, the group need to check against its preference $\alpha_{s_{i,1}}$ to decide whether to accept this project $p_j$ or not. Generally speaking, if each student $s_i$ is limited to have $m$ number of projects in $\alpha_i$, then the algorithm will run in $O(mn)$.

\section{Correctness of the SPA-GROUP Algorithm}

The correctness of the SPA-GROUP algorithm may be established using following lemmata.

\textbf{Lemma 1.} The SPA-GROUP algorithm terminates with a matching $M$.

\textit{Proof.} During each iteration, a lecturer $l_k$ who is under-subscribed will offer a project $p_j$ to a group $G_i$ which is in $l_k$’s preference list on the project $p_j$. If $l_j$ gets rejected from the group $G_i$ then $l_j$ will continue offer the project $p_j$ to the next group in the preference list of $p_j$; that is $\beta_{k,j}$ until there is a group to accept the offer or the end of the preference list. The group $G_i$ will reject the offer if $G_i$ is already assigned to a project $p'$ such that $G_i$ prefers $p'$ to $p_j$ (i.e. the offered project). Otherwise $G_i$ will accept the project $p_j$ provided that lecturer $l_k$ has enough capacity to admit the group. Therefore it is clear that this set of assignments is a subset of $M \in R \times P$ which is a matching. \hfill \Box
Lemma 2. No pair of assignment \((r_i, p_j)\) can block the resulted matching \(M\) if it is removed during the execution of the Spa-Group algorithm.

Proof. Suppose for a contradiction that \((r_i, p_j)\) blocks the matching \(M\). This can only happen when the group \(G_i\) receives a project \(p'\) such that \(G_i\) prefers \(p'\) than \(p_j\). This applies to every subsequent iteration because otherwise the group \(G_i\) will only stay with the current assignment \(p_j\). Therefore at the end of the execution, \(G_i\) can only prefer the resulted project than \(p_j\) and \((r_i, p_j)\) cannot be a blocking pair. \(\square\)

Lemma 3. The Spa-Group algorithm returns a stable matching.

Proof. By Lemma 1, let the resulted matching be \(M\). Suppose the algorithm does not return a stable matching, then there exists a blocking pair \((r_i, p_j)\) in the final matching that can improve \(M\). Further suppose the project \(p_j\) is offered by \(l_k\), then such a pair must satisfy the condition stipulated in Definition 3. Clearly, if \(r_i\) is assigned to \(p_j\) then \(r_i\) must find \(p_j\) acceptable (Condition 1) and prefer \(p_j\) to any provisional assignment (Condition 2). It follows that the Condition 3 must not be satisfied so that a contradiction occurs. The technique of proofing is similar to that of the Spa-Lecturer algorithm [1].

(a) \(p_j\) is under-subscribed and \(l_k\) has enough capacity to admit group \(G_i\), that is, \(|G_i| \leq |M(p_j)|\) and \(|G_i| \leq |M(l_k)|\).

This satisfies the loop condition and the group capacity condition such that the resulted matching \(M\) is not yet terminated which contradicts to Lemma 1.

(b) \(p_j\) is under-subscribed with \(|G_i| \leq |M(p_j)|\) but \(l_k\) is full, and if \(r_i\) is assigned to \(p_j\) then \(r_i\) must find \(p_j\) acceptable (Condition 1) and prefer \(p_j\) to any provisional assignment (Condition 2). It follows that the Condition 3 must not be satisfied so that a contradiction occurs. The technique of proofing is similar to that of the Spa-Lecturer algorithm [1].

Consider at time \(T\) where the worst group \(r' = (s', |G'|)\) is assigned to a final project \(p'\) where no more deletion will take place. Let the matching at time \(T\) be \(M'\) such that \((r', p') \in M'\). Observe that on any subsequent assignment of \(l_k\) must prefer a group \(G_p\) than \(G'\) for some project \(p_s\) in \(\beta^*_k\) as \(l_k\) is full. Also, this project \((r_p, p_s) \notin M\) but has not yet been removed before time \(T\). Let this set of projects \(p_s\) be \(S\). Note that \(p_s\) must be full at time \(T\) because otherwise \(l_k\) will be matched with \(G_p\) as \(l_k\) prefers \(G_p\) than \(G'\). It follows that at time \(T\), the total
number of students matched with $l_k$ is given by

$$|M'(l_k)| = \sum_{p_i \in S \setminus \{p_j\}} |M'(p_i)| + |M'(p_j)| + \sum_{p_k \in P_k \setminus S} |M'(p_k)| \leq d_k.$$  

Since any assignment after $T$ can only happen for some projects in $S$, and since all projects in $S$ are full, it follows that

$$|M(l_k)| \leq \sum_{p_i \in S \setminus \{p_j\}} |M'(p_i)| + |M(p_j)| + \sum_{p_k \in P_k \setminus S} |M'(p_k)|.$$  

Given the fact that $p_j$ is under-subscribed in $M$,

$$|M(l_k)| < \sum_{p_i \in S \setminus \{p_j\}} |M'(p_i)| + |M'(p_j)| + \sum_{p_k \in P_k \setminus S} |M'(p_k)| = |M'(l_k)| \leq d_k.$$  

This means lecturer $l_k$ is under-subscribed which contradicts to the assumption.

(c) $p_j$ is full and $l_k$ prefers $r_i$ to the worst group matched in $M(p_j)$ say $r'$ with $|G_i| \leq |G'_i|$.  

Suppose it is true that lecturer $l_k$ prefers $r_i$ to $r'$, then $(s_i, p_j)$ must not be in the resulted matching $M$ otherwise $l_k$ would offer $p_j$ to $r_i$. This contradicts to Lemma 2 as $(s_i, p_j)$ does not exist to block the matching.

---

**Lemma 4.** No stable pair is removed during the execution of the SPA-GROUP algorithm.

**Proof.** Suppose for a contradiction that $(r_i, p_j)$ is the first stable pair that is removed during an execution $E$. It is clear that $r_i$ has received an project $p'$ offered by lecturer $l'$ such that $r_i$ prefers $p'$ to $p_j$. But it can be shown that $(r_i, p')$ blocks the matching $M$; that is $(r_i, p')$ forms a blocking pair stipulated in Definition 3. The first two conditions are clearly satisfied. It remains to show this assignment satisfies one of the Condition 3(a), (b) or (c). If $p'$ is full and $|G_i| \leq |G'_i|$ then (c) holds. Otherwise $p'$ is under-subscribed. This leads to two conditions:

1. If lecturer $l'$ is under-subscribed with $|G_i| \leq |M(l')|$ then (a) holds; or
2. If lecturer $l'$ is full, then only if $l'$ prefers all group matched in $M(l')$ than $r_i$ will make (b) hold. Suppose for a contradiction that it is the case. It follows that any assignment to $l'$ must be stable. Since $l'$ makes an offer to $r_i$, $l'$ must have at least $d' - |G_i|$ capacity to admit group $G_i$. This impels that $l'$ must first delete some group assigned. This contradicts to the assumption that $(r_i, p_j)$ is the first such stable pair to be removed.

\[\Box\]

Given these lemmata, following theorem can be establish.

**Theorem 1.** Given any instance of spa defined in 2.1, the SPA-GROUP algorithm returns a stable matching $M$ between students and lecturers.

*Proof.* This is immediate followed after Lemma 3 and 4. From Lemma 3, it is known that the SPA-GROUP algorithm returns a stable matching. And from Lemma 4, there is no stable pair deleted during the execution of the algorithm. Therefore any assignment during the execution of the algorithm can only improve the matching but not producing blocking pairs. Otherwise it will contradict with 3. \[\Box\]

5.3 Implementation Details

5.3.1 SPA-GROUP Algorithm

The SPA-GROUP listed in Algorithm 1 is implemented as `spa-group.py` in Python. The program requires a valid JSON file as an input of the data. The input JSON file has to contain following necessary information.

```json
{
    "project_list": [
    {
        "id": "(int) project ID",
        "name": "(int) project name",
        "capacity": "(int) project capacity",
        "lid": "(int) LID of the lecturer who offered this project",
    },
    {
        "..."
    }
    ],
    "group_list": [
```
{  "id": "(int) group ID",
   "representative_id": "(int) student representative ID",
   "preflist": "(str) preference list of projects",
   "capacity": "(int) group capacity",
   "members": "(str) list of members ID"
},
{
   "...

",
"lecturer_list": [
{
   "id": "(int) lecturer ID",
   "capacity": "(int) lecturer capacity",
   "projects": [
   {
      "id": "(int) offered project ID",
      "capacity": "(int) project capacity",
      "preflist": "(str) preference list of group",
   },
   {
      "...
   }
   ],
   {
      "...
   }
  }
],
{
   "...
}]
}

Note that in preflist, the values are separated by comma. For example, if a student prefers project $p_3$, $p_1$ than $p_2$, then preflist field will be $3,1,2$. This also applies to the members field in group_list where students’ IDs are separated by comma.

Likewise, the program returns the matching in a JSON file called matching.json under the same directory. The file has a structure as follows.

{
   "project_matched": [
   {
      "project ID": ["list of lecturer IDs matched"],
      ".........": ["...
   },
   "group_matched": [
   {
      "project ID": ["list of group IDs matched"],

21
The Project Allocation System is built on top of Django web framework. The framework is responsible for receiving users’ requests and responding to them accordingly. The back-end only responses requests in JSON form. It does not render the web page. This allows a great flexibility in modifying the web page. The system handles following requests:

<table>
<thead>
<tr>
<th>Request</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get/student/&lt;sid&gt;</td>
<td>GET</td>
<td>Get the information of a student with id sid.</td>
</tr>
<tr>
<td>get/studentList/</td>
<td>GET</td>
<td>Get the list of students.</td>
</tr>
<tr>
<td>get/project/&lt;pid&gt;</td>
<td>GET</td>
<td>Get the information of a project with id pid.</td>
</tr>
<tr>
<td>get/projectList/</td>
<td>GET</td>
<td>Get the list of projects.</td>
</tr>
<tr>
<td>get/lecturer/&lt;lid&gt;</td>
<td>GET</td>
<td>Get the information of a lecturer with id lid.</td>
</tr>
<tr>
<td>get/lecturerList/</td>
<td>GET</td>
<td>Get the list of lecturers.</td>
</tr>
<tr>
<td>get/listAll</td>
<td>GET</td>
<td>Get the list of students, projects and lecturers at once. The format is comply with the input format of spa-group.py</td>
</tr>
<tr>
<td>get/matching</td>
<td>GET</td>
<td>Get the matching result. The format is the same as in matching.json generated by spa-group.py</td>
</tr>
<tr>
<td>get/clearMatching</td>
<td>GET</td>
<td>Clear the matching result.</td>
</tr>
</tbody>
</table>

Table 5.1: List of Requests handled by the PAS.

In generating the matching in /get/matching, the request will trigger following actions:
1. Pass the request to /get/listAll to generate the list of all necessary information and store it into the data.json file; then

2. Use Python subprocess module to run spa-group.py with the data.json as the input file. This function is a blocking call. The result is then store in the matching.json file; then

3. Update corresponding fields using the data in matching.json and send this file through JsonResponse to the user.

The detail to setup the development server is given in Appendix B.

5.3.3 Front-end of the PAS

The front-end of the PAS uses AngularJS 1.0. This allows the web application to be a single-page application. The styling of the webpage is accomplished by Bootstrap with theme Flaty provided by Bootswatch. Figure 5.1 depicts the index page of the PAS. Users are allowed to view the list of students, projects or lecturers through the web interface.

The details of each page are too tedious to describe in here. However, it is worthwhile to give the screenshots of both before and after the matching is generated. For students, figure 5.2 gives the matching result in the fourth column of student list. If the student
s is assigned to any project, a link to the project will be provided for s. Otherwise it will display None which means s is unassigned.

![Student List](image)

**Student List**

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Project Preference</th>
<th>Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alice</td>
<td>1</td>
<td>None</td>
</tr>
<tr>
<td>2</td>
<td>Bob</td>
<td>2</td>
<td>None</td>
</tr>
<tr>
<td>7</td>
<td>Carol</td>
<td>2,1</td>
<td>None</td>
</tr>
<tr>
<td>8</td>
<td>David</td>
<td>1</td>
<td>None</td>
</tr>
<tr>
<td>9</td>
<td>Eliot</td>
<td></td>
<td>None</td>
</tr>
<tr>
<td>10</td>
<td>Fred</td>
<td></td>
<td>None</td>
</tr>
<tr>
<td>11</td>
<td>Geoffrey</td>
<td></td>
<td>None</td>
</tr>
<tr>
<td>12</td>
<td>Henry</td>
<td></td>
<td>None</td>
</tr>
</tbody>
</table>

(a) Before generating the matching

![Student List](image)

**Student List**

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Project Preference</th>
<th>Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alice</td>
<td>1</td>
<td>Project F</td>
</tr>
<tr>
<td>2</td>
<td>Bob</td>
<td>2</td>
<td>Project B</td>
</tr>
<tr>
<td>7</td>
<td>Carol</td>
<td>2,1</td>
<td>Project B</td>
</tr>
<tr>
<td>8</td>
<td>David</td>
<td>1</td>
<td>Project F</td>
</tr>
<tr>
<td>9</td>
<td>Eliot</td>
<td></td>
<td>None</td>
</tr>
<tr>
<td>10</td>
<td>Fred</td>
<td></td>
<td>Project D</td>
</tr>
<tr>
<td>11</td>
<td>Geoffrey</td>
<td></td>
<td>Project F</td>
</tr>
<tr>
<td>12</td>
<td>Henry</td>
<td></td>
<td>Project F</td>
</tr>
</tbody>
</table>

(b) After generating the matching

Figure 5.2: The view of student list (a) before the matching and (b) after the matching is generated.

For lecturer, a lecturer called Diana is given as an example in Figure 5.3. If the lecturer is assigned to any group, the project box will be highlighted and a caption “MATCHED” will be appeared in the heading of the project. In this example, Diana is
assigned to project F with group ID 1.

Figure 5.3: The view of a lecturer Diana (a) before the matching and (b) after the matching is generated.
Chapter 6

Technical Challenges

6.1 Optimality

The Spa-Group algorithm only generates a lecturer-optimal stable matching. Checking the optimality of the result, however, is a Np-COMPLETE problem because there are totally $O(2^n)$ possible matching. Generally, it would not be possible to check the optimality of resulted matching for large number of students and lecturers.

6.2 Matching Generation in PAS

In section 5.3.2, it is known that the subprocess.call() is used in executing the spa-group.py to generate the resulted matching. This function call provided by Python is a blocking function which means the entire program is blocked by this function call once it is invoked. The program will continue execute until the subprocess.call() has finished its execution. A better approach is to merged the entire spa-group.py into the PAS so that no blocking function is called.

6.3 Browsers Compatibility

Since only web interface will be implemented, there may have discrepancy between different devices. This will lead to difficulties for users to use the system. It is hoped that by using some well-written web front-end frameworks, the user interface can maintain its consistency across different devices.
Chapter 7

Future Work

7.1 Lower Bound for Project Capacity

Currently, the project capacity limits the maximum number of students can be assigned to a project. A more general situation is that each project has a minimum number of students in order to be matched; that is the lower bound of the project capacity. It is particularly useful when lecturer offers an industrial-based project where the company requires a fixed number of students to be participated in a project. In such a case, the lower bound of the project is equal to that of the upper bound so that a fixed number of students is required to be matched.

7.2 Definition of SPA with group projects

The definition of SPA with group projects in section 2.1 has an assumption that each group $G_i$’s representative has delegated all $G_i$ students’ preferences. All members from the same group $G_i$ have assumed to agree representative preference. In general, this assumption is acceptable because otherwise the student $s_d$ will not join the group $G_i$ if $s_d$ has a different preference than that of $s_{i,1}$. However, a more general situation is illustrated as follows. The representative does not delegate the preference for all members. Instead, whenever there is a project $p$ offered by a lecturer, the representative is obliged to seek for members’ preferences. If a threshold number of members are preferred to $p$ than any other projects, then the group will be assigned to $p$. For example, if more than half of the members are preferred the project $p$, then the representative will accept the offer. Otherwise the project $p$ is rejected.
7.3 Unallocated Group

Currently, any unallocated project will not be allocated to any student who is also unallocated. It is clear that any when the SPA-GROUP algorithm is terminated, any student who does not receive an assignment will remain unallocated. In reality, this may not be desirable to have any student unallocated. Taking school project as an example, all student need to complete a project before graduate. Therefore each student must be allocated to at least a project. There are few possible methods to solve the problem.

1. Require lecturers to re-rank group that are unallocated; or

2. Generate a subset of unallocated students and projects. Then require both students and lecturers to rank their preferences over project or group. Run the SPA-GROUP algorithm to generate the matching. Repeat this process until all student have been assigned.

It is clear that there are other possible methods to solve this unallocated problem and here only depicts few heuristics solution.
Chapter 8

Schedule

The tentative schedule to design and implement the Project Allocation System (PAS) is given in Table 8.1. There are distinct milestones to be achieved on each phase. There are 3 major deliverables on each phase.


   This Spa-lecturer prototype implemented the Spa-lecturer algorithm. This prototype is programmed in Python and is aimed to solve the spa without group projects.

2. The PAS, completed, delivered in April 2017

   All functions of the PAS, including matching generation and administrator panel, will be completed together with a meticulous final report that outlines the details.
<table>
<thead>
<tr>
<th>Date</th>
<th>Tasks</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 Oct 2016</td>
<td><strong>Deliverables of Phase 1</strong> (Inception)</td>
<td>Completed</td>
</tr>
<tr>
<td></td>
<td>• Detailed project plan</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Project website</td>
<td></td>
</tr>
<tr>
<td>Oct 2016</td>
<td>Requirements gathering and analysis</td>
<td>Completed</td>
</tr>
<tr>
<td></td>
<td>• Collect user requirements</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Further study on SPA based on the requirements</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Implement a prototype to solve the SPA</td>
<td></td>
</tr>
<tr>
<td>Nov 2016</td>
<td>Deliver the SPA-LECTURER prototype and the preliminary draft on the SPA-GROUP algorithm</td>
<td>Completed</td>
</tr>
<tr>
<td>Dec 2016</td>
<td>Development on the PAS Demo System</td>
<td>Completed</td>
</tr>
<tr>
<td></td>
<td>• Database and user interface design</td>
<td></td>
</tr>
<tr>
<td>22 Jan 2017</td>
<td><strong>Deliverables of Phase 2</strong> (Elaboration)</td>
<td>Completed</td>
</tr>
<tr>
<td></td>
<td>• Detailed interim report</td>
<td></td>
</tr>
<tr>
<td>Feb 2017 to Mar 2017</td>
<td>System Design and Construction</td>
<td>Completed</td>
</tr>
<tr>
<td></td>
<td>• Refine the PAS Demo System</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Implement the administrator user interface</td>
<td></td>
</tr>
<tr>
<td>Apr 2017</td>
<td>Deployment and Testing</td>
<td>Completed</td>
</tr>
<tr>
<td></td>
<td>• Final testing and optimisation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Deploy the PAS</td>
<td></td>
</tr>
<tr>
<td>17 Apr 2017</td>
<td><strong>Deliverables of Phase 3</strong> (Construction)</td>
<td>Completed</td>
</tr>
<tr>
<td></td>
<td>• Finalised PAS implementation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Final report</td>
<td></td>
</tr>
</tbody>
</table>

Table 8.1: Schedule.
Chapter 9

Conclusion

This project has introduced the web-based *Project Allocation System* (PAS) that aimed to model the Student-Project Allocation problem (SPA) with group projects in a real-life scenario. The web-based nature of the system allows great flexibility across devices. This SPA with group projects allows different students work together as a group which is commonly adopted by many universities. Also, the project has presented an algorithm called the SPA-GROUP that targeted to generate the *stable matching* between students and projects based on the preferences and the capacities. Unlike previous algorithms, it supports grouping between different students.

This report described the implementation details of the PAS, in particular, the system architecture is presented. The PAS provided a intuitive way in viewing the matching result. It greatly enhances the efficient in generating a stable matching.

Despite the project only describes a matching problem between students and projects, there are versatile applications of this centralised matching algorithm. It is in fact constantly appears on different contexts. For example, in a Content Delivery Network\(^1\) (CDN), a group of users (group of students) from related regions may send requests to the CDN. It will then list out a number of available servers (projects) that can be used to respond to the requests with a priority that possibly based on the geographical location. A centralised matching scheme is run to generate the stable matching that maps each user to the closest sever. There are potentially many uses of this algorithm that are left to be discovered.

---

\(^1\)a distributed network of servers that provides high performance service in delivering web content
Bibliography


Appendix A

Pseudocode of the SPA-LECTURER

The pseudocode of SPA-LECTURER algorithm is listed in Algorithm 2. This algorithm is an adaption from a study [1]. Changes have been made to conform the definition stipulated in Section 2.1. The main logic of this algorithm is similar to that of the SPA-GROUP algorithm except that no group capacity is checked.

Algorithm 2 The pseudocode of the SPA-LECTURER.

1: procedure Spa-Lecturer(I)
2: initialise each student, project and lecturer to be free
3: $M = \emptyset$ \hspace{1cm} ▷ the matching result
4: while there is a lecturer $l_k$ with a $p_j \in P_{l_k}$ under-subscribed do
5: $s_i =$ first student on $l_k$‘s preference list \hspace{1cm} ▷ $s_i \in \beta^i_k$
6: $p_j =$ first project on $s_i$‘s preference list \hspace{1cm} ▷ $p_j \in \alpha_i$
7: if $(s_i, p_j) \notin M$ then
8: if $s_i$ is assigned to a project $p$ then
9: delete $(s_i, p) \in M$ \hspace{1cm} ▷ delete current $s_i$‘s matching
10: end if
11: assign $(s_i, p_j) \in M$ \hspace{1cm} ▷ $l_k$ offers $p_j$ to $s_i$
12: for all successor $p_l$ of $p_j$ on $\alpha_i$ do
13: delete $p_l$ on $\alpha_i$ and $\beta^l_d$ \hspace{1cm} ▷ if $p_l$ is offered by $l_d$
14: end for
15: else
16: continue
17: end if
18: end while
19: return $M \in S \times P$
20: end procedure
Appendix B

Local Development Server Setup

The Project Allocation System is developed under Python version 3 using Django web framework version 1.11. Therefore it is recommend to install Python version 3 instead of 2.

To setup the local development environment, first change the directory to websys, then apply follow commands:

$ python3 manage.py makemigrations
$ python3 manage.py migrate

If there is no error induced, the local development server can be setup using following command.

$ python3 manage.py runserver

If there is no error induced, following message will appear

Performing system checks...

System check identified no issues (0 silenced).
April 10, 2017 - 12:02:26
Django version 1.11, using settings 'websys.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.

and the local development server can be accessed at http://127.0.0.1:8000/.