Student Course Assignment

Apoorva Vijay Tamaskar

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

A dissertation presented in part fulfilment of the requirements of the Degree of Master of Science at The University of Glasgow

Date of submission placed here

Abstract

We designed and implemented an Integer Linear Programming model for student course allocation problem:-

An administrator has to assign students to over-demanded courses efficiently and fairly depending on the preference list provided by the students

Student-Course allocation is a complex problem that the universities across the world face due to the delicate nature of the assignment problem. Recently, numerous solutions have been proposed, some of which use Integer Programming, combinatorial auction design, matching theory and Artificial Intelligence. We decided to use Integer Programming due to the nature of the constraints given to us by Glasgow School of Arts.

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Name: _____ Signature: _____

Acknowledgements

I would like to thank my supervisor, Dr. David Manlove, for the patient guidance, encouragement and advice he has provided throughout my time as his student. I have been extremely lucky to have a supervisor who cared so much about my work, and who responded to my questions and queries so promptly. In particular I would like to thank William Pettersson for the suggestions he made throughtout the course of the project.
I must express my gratitude to my mother, father and brother who experienced all of the ups and downs of my project.
My peers, the postgraduate students at Glasgow, who provided a much needed form of escape from my studies, also deserve thanks for helping me keep things in perspective.
Finally, I would like to thank the University of Glasgow Computer Science department and the Glasgow School of Arts for providing the opportunity to undertake this project.

Contents

Chapter 1

Introduction

The course allocation problem is a case of multi-unit assignment problem. The multi-unit assignment problem is the problem of allocating a subset of set of objects to another set of objects with given additional constraints. The goal of these types of problems is to maximize the overall ”fairness” of the final allocation, this requires a setup which will enable the solver to gather enough information required to solve the problem.

At most undergraduate and post graduate institutions, to give a student the optimal learning experience requires putting a limit on the number of classes a student can take and number of students in a class, these along with other constraints make it impossible for all students to receive their highest preferred courses. This makes a challenging task of allocating students to courses in a fair and quitable way for the administrators. This raises the question what all should be considered when allocating students to courses from their preference list?

The course allocation is widely studied type of problem in market design theory as various institutes have subtle differences in their own requirements for the types of courses a students needs to take. It is to be noted that there are theorems for this problem which imply no perfect solution exists and that there is no method the solve the problem efficiently and fairly. Few of the current softwares either give up efficiency or take risks of unfair outcomes.

Few institutes use first come first serve mechanism, few use auction-based mechanism, few allow students with better GPA to take up courses earlier. These kinds of skewed mechanisms reduce the overall fairness. In this report we will present details for the Integer Linear Programming model we developed for the English department of the Glasgow School of Arts and will have a brief look at some works which are already done in this area.

This report has the following structure: first, we look some of the previous work done in this area, we will then look at the various constraints posed by Glasgow School of Arts, the explain the reasoning for various decisions we made and do few comparisons. Second, we look at how we translated the constraints posed into the programming language and explain various details of our implementation. Third, we look at how the program was tested and evaluated while in development. Lastly we present how we approached the development of the program and ideas for further work.

Chapter 2

Analysis and Requirements

At this point having the general idea of the problem, we dive into the full detailed requirements set by the Glasgow School of Arts.
English Literature department currently allocate students to Honors courses by a manual process on a first come first served basis. The cohort size makes this a good candidate for automation. Hence a software that will allocate students to Junior and Senior Honors courses to produce 1) a list of courses for the student and 2) class rosters for each course is required. The software should maximize student preferences as far as possible within course capacities and enrollment requirements.
English Literature allocate courses to separate cohorts of Junior and Senior Honors students. Courses are designated as Junior Honors or Senior Honors. There are approximately 190 Junior Hons (year 3) students and 180 Senior Hons (year 4) with about 70% joint and 30% single Hons.
Incoming Senior Hons students complete a form with course choices in March. Incoming Junior Hons students complete a form in July. At present this is a paper form that is processed manually.

Student Details	Compulsory courses	Optional courses
Junior Single Hons, 120 credits (60 per semester)	20 credit core course each semester	2x20 credit courses each semester
Junior Joint Hons, 60 credits across 2 semester	One 20 credit core course in any semester.	2x20 credit courses in any semester. Can include the core course not taken.
Senior Single Hons, 120 credits (60 per semester)	40 credit dissertation in first or second semester	4x20 credit courses (1 course alongside dissertation and 3 in other semester)
Senior Joint Hons, 60 credits across 2 semester	-	40 credit dissertation in first or second semester and 20 credit course taken in semester 1 or 2 OR 3x20 credit courses across 2 semesters

Optimization Parameters:-

- Student preference: students rank their optional course choices.
- Enrollment requirements: pre-1800 courses, Across the two Honors years, a Single Hons student must take two courses designated as Pre-1800 and a Joint Hons student must take one Pre-1800 course. These can be taken in Junior or Senior Hons. Data entry can ask Senior Hons students how many Pre-1800 courses they need to take.
- Dissertation choice for Senior Hons:Joint Hons students can choose to take a 40 credit dissertation in English Literature or another subject. The dissertation can be taken in first or second semester. Data entry can ask the student if they are doing an English Lit dissertation and if so in what semester. Single Hons students can take their 40 credit dissertation in first or second semester. Data entry can ask them to choose a semester.
- Upper and lower bounds: upper and lower capacity limits for each course are set by the subject. The courses all need to run, so lower bounds must be met even if this means allocating students to a less-preferred course over a higher preference with capacity. In the case of English Literature, the lower bound is 8.

Additional factors:-
Two additional optimisation factors are not applicable to English Literature but are relevant in other subjects that might use this software.

- Timetable Clashes: The lectures in English Literature are timetabled not to clash. However, in other subjects there can be clashes. We would like the software to include an option to designate clashes, e.g. Course A cannot be taken with Course B.
- Priority: in English Literature, Senior Hons students can include Junior Hons courses in their preferences (the opposite is not allowed). This can be handled manually by allocating Senior Hons students into the Junior Hons courses and adjusting the upper bound accordingly before running the data to allocate Junior Hons students. In other subjects, both Senior and Junior Hons students compete for the same courses with Seniors given priority. It would be helpful if the software could include a factor for priority.

From the given requirements we split these into Must have, Should have, could have and Would have as follows:-

- Must:-
  Should output the assignment while trying to follow the maximum number of student preference, the dissertation preference, enrollment requirements and lower and upper bound on course quotas.
- Should:-
  First, a checker function can be implemented which checks whether the input is in the correct format and can correct small errors in the input or at least specify the errors.
  Second, the optimization parameter can be incorporated into the objective function with proper penalties so that the order of preference is as required.
  Third, as suggested can add in a extra step to check for timetable clash, but this will require

the input data to contain all this information, which might be difficult to obtain. Fourth, can give priority to Senior students over Junior students.

- Could:-
  As the current program is run through terminal/command line new users can find this challenging, hence a GUI can be of help.

- Would:-
  First, generalize the program to such a levels such that no constraints need to be hardwired, the administrators(users) can provide a simple text file with the constraint description and the program will implement them accordingly.
  Second, integrate the program into an online application so that the entire process from data collection to final output is a single pipeline.

To approach this problem we had various methods available, but we decided to use Integer Linear Programming approach as the parameter to optimize were not clear at the beginning of the project and Integer Linear Programming provides the flexibility to modify the objective function as the requirements and parameters to optimize change. This begs the question, what is Integer Linear Programming?
In general sense, Integer Linear Programming problem comprise of an objective function which is to be maximized or minimized subject to few constraints.
Mathematically there are two forms of ILP, Standard form:-

$$\begin{gathered}\text{Maximize } c^T x \\ \text{Subject to } Ax + s = b \\ s \geq 0 \\ x \geq 0 \\ \text{and } x \in \mathbb{Z}^n\end{gathered}$$

The canonical form:-

$$\begin{gathered}\text{Maximize } c^T x \\ \text{Subject to } Ax \leq b \\ x \geq 0 \\ \text{and } x \in \mathbb{Z}^n\end{gathered}$$

Here, c, b and s are vectors which need not be of the same dimension and A is a matrix with the obvious restriction that all entries must be integers. It should be noted that a canonical form can be converted to standard form by introducing dummy variables to remove inequalities.
We give few constraints on the dimensions of the vectors and martix.

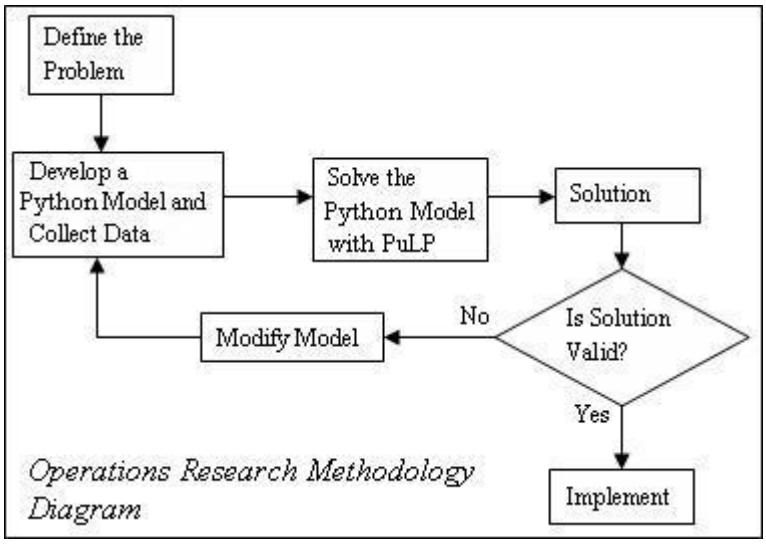
$$\begin{gathered}c \in \mathbb{Z}^n \\ \text{If } A \in \mathbb{Z}^{m*n}, \text{ where } m \in \mathbb{Z} \\ \text{then } b \in \mathbb{Z}^m \\ \text{and } s \in \mathbb{Z}^m\end{gathered}$$

As the student course allocation is a very well studied problem many other approaches have been used to tackle it. Various school use distinct methods as discussed earlier. To discuss the merits and demerits of our program, we needed to look into lots of material online, below we present some of the things we found. Later we will give a more detailed explanation of the methods used in these softwares.

Chapter 3

Design and Implementation

For the purpose of the project, we choose to implement the Integer Linear Programming model in python, using the pulp package.
We decided to use python due to the ease of writing in, flexibility and portability of the language. The intended use of the program is to assign students of the Glasgow School of Arts to courses, we were told that the program should have less than 30 secs of runtime.
Compared to Python, Java required much more memory, requires much more structure in the code along with a clear goal which would otherwise lead to various errors in the code already written if the requirements change. As it was difficult to communicate with people in charge in the Glasgow School of Arts, we started coding in a much more general sense i.e. we though about what sort of constraints will need to be there in the program, what will be the nature of constraints.
Compared to Python, C++ is indeed time and memory efficient, but as we had previous experience with coding Integer Linear Programs in python, we discarded the thoughts of coding in C++.
As discussed, we decided to implement out model in python using the pulp package. Earlier we mentioned the optimization process, the modeling process can be considered a well thought out and structured way of the same.



- Getting the Problem Description
  The goal here is to establish a formal and rigorous model for the problem. Similar to our case, at the start of any optimization project the people working on the problem will only have a rough idea of the problem and the requirements. This needs to be refined through multiple discussions with the people providing the problem, the client, which in our case was the Glasgow School of Arts. These discussions will in turn bring around more restrictions on the problem or will highlight few key points which were earlier missed in informal discussions about the problem and will give an idea about the data available to work with. Not all data provided is relevant to the project or sometime the required data is not available, which will again require discussion with the client. These limitations may lead to change in model description and constraint formulation.

- Formulating the mathematical program
  This step requires the programmer to identify the decision required to be made, the restrictions posed in the problem, as well as the final goal, all from the problem description. At this step it is beneficial to capture the interdependencies in a mathematical model. Typically the formulation process has four key steps:-

  – Identify the data needed for the objective function and constraints.
    In our case, firstly we required the knowledge of an identifier for each student and course for which we used the student ID and course ID. Then for each student we need to know which year they are in, 3^{rd} year or 4^{th} year which was represented by 3 or 4, whether they are single or joint honors students, the number of pre-1800s courses they are required to take, for the students 3^{rd} year joint honors we need to know when they are taking the core course, for 4^{th} year honors we need to know the dissertation preference if at all they are doing the dissertation and finally the preference list for each student. For each course we require if they course is offered in 1^{st} or 2^{nd} semester, if the course is pre-1800s or not and the course capacity of each course.
    Due to the nature of the problem give to us we also thought about obtaining the lower bound on number of students in a course data, but decided to hard code it.

  – Decide the decision variables.
    The goal of this program is to assign students to courses taking into account the curriculum, the preference list while minimizing the violations of constraints and maximizing the student happiness.
    This encourages create a decision variable for each student-course pair. In ideal situations this should have been enough, by ideal we mean feasible situations, but as this need not always be the case, we decided to add dummy variables which will take into account the severity of the constraint violated.
    Hence we decided to keep dummy variables for the cases when course does not have enough students, when the number of students in a course exceed the course quota, when a student has not taken enough courses, when a student has not taken enough pre-1800s courses and four variables for when the load of courses a student is taking in a semester is unbalanced,

  – Formulate the objective Function using the decision variables, we can construct a minimize or maximize objective function.
    First we look at minimization problem in the case where we know the problem is feasible without the dummy variables. Initially decided to minimize the total sum of ranks of the courses which students were assigned. This objective function has obvious bias towards students who require more number of courses.

Hence we changed our object function to minimize the sum of average of ranks of courses assigned to each student.
Later we were told that 4^{th} year student have to be given preference over 3^{rd} year students. Hence we decided to give a penalty to the sum of average of ranks of courses assigned to 3^{rd} year students. The penalty was a safe upper bound on the maximum possible objective for 4^{th} year students.
After we have an objective function for the simple case, i.e. cases where dummy variables are not required, we proceed to incorporate the dummy variables in the objective function with the help of penalties.
The Glasgow School of Arts assigned the highest preference to every student being assigned courses so that their curriculum is satisfied, second preference was given to the course restrictions such as minimum and maximum number of students in a course, third comes the courses assigned to the 4^{th} year honors students, fifth their whether they got their dissertation preference and lastly sixth, the courses assigned to the 3^{rd} year students.
For each stage of preference we have a penalty which is a safe upper bound on the sum of all the values below it, we call these the limit $l_i \forall i \in [1, 6]$. We will present the values of these limits in the next chapter.

– Formulate the Constraints
Suppose, the decision variables are $X[i][j]$ i is the student number, j is the course number, number of students is n, number of courses is m.
Dummy variables for each course $oveCor$ and $minCor$ which denote number of students above the course quota and number of students below the limit respectively.
Dummy variables for each student $maxStd$, $preStd$, $firDis$, $secDis$, $firStd$ and $secStd$ which represent the extra number of courses the student requires, the extra number of pre-1800s course the student requires, the extra credits taken in the first semester along with dissertation, the extra credits taken in the second semester with dissertation, the extra credits taken in the first semester and the extra credits taken in the second semester, respectively.
Here is an example of the j^{th} course which has upper limit of 40 and lower limit of 8.

$$\sum_{i=0}^{n-1} X[i][j] - oveCor[j] \leq 40$$

$$\sum_{i=0}^{n-1} X[i][j] + minCor[j] \geq 8$$

The first constraint is equivalent to the condition desired as, we are summing over all the students who have been assigned this course and checking if less than 40 are assigned to it, but if more than 40 are assigned to the course then the $oveCor[j]$ dummy variable is used to keep the constraint satisfied by quantifying the extra number of students in the course and giving a penalty depending upon the quantity exceeded by.
The second constraint is equivalent to the condition desired as, we are summing over all the students who have been assigned this course and checking if more than 8 are assigned to it, but if less than 40 are assigned to the course then the $minCor[i]$ dummy variable is used to keep the constraint satisfied by telling how many more students are needed in the course and giving a penalty depending upon the required courses.
Now an example of a 3^{rd} year Single student(say the i^{th} student) who has to take 2

pre-1800s courses. As the student is 3^{rd} year, the program will interpret it as 0

$$\sum_{j=0}^{m-1} t[j] * X[i][j] - secStd[i] == 2$$

$$\sum_{j=0}^{m-1} (1 - t[j]) * X[i][j] - firStd[i] == 2$$

$$\sum_{j=0}^{m-1} p[j] * X[i][j] + preStd[i] \geq 0$$

$$X[i][m] + X[i][m+1] == 0$$

$$\sum_{j=0}^{m-1} X[i][j] + maxStd[i] \geq 4$$

The first constraint is equivalent to saying that the number of courses taken in the second semester is 2, else there is a penalty depending upon the value of $secStd[i]$. We sum up overall the courses in second semester and see if the value is equal to 2, else we have a penalty depending on the courses needed.
The second constraint is equivalent to saying that the number of courses taken in the first semester is 2, else there is a penalty depending upon the value of $firStd[i]$. We sum up overall the courses in first semester and see if the value is equal to 2, else we have a penalty depending on the courses needed. The third constraint is equivalent to saying that the number of pre-1800s courses taken is 0 or more, if not the $preStd[i]$ dummy variable is used to make up for the required number of courses and apply a proportional penalty.
The fourth constraint simply says that the student is not taking a dissertation.
The last constraint is equivalent to saying that the number of courses taken is 4 or more and if it is not the case then $maxStd[i]$ dummy variable takes a value so that the constraint is satisfied and applies a proportional penalty.

- Solving the mathematical program
  For simple and well understood problems the model can be made so that an optimal solution can be found. Various algorithms can be used for this purpose, such as Revised Simplex Method, Interior Point Methods. However in most cases the problems are not this easily solvable due to the sheer size of the problem, that is the number of variables and constraints. Hence, these are mostly solved using heuristics at the cost of optimality. Pulp has the following status for the solutions:-
  - LpStatusOptimal,"Optimal",1
  - LpStatusNotSolved,"Not Solved",0
  - LpStatusInfeasible,"Infeasible",-1
  - LpStatusUnbounded,"Unbounded",-2
  - LpStatusUndefined,"Undefined",-3

The modeling process includes more steps which are discussed in the next section.
We do not provide a hardness proof for the problem in this report

Chapter 4

Text and Evaluation

- Performing some post-optimal analysis
  It is often the case that the problem description has some uncertainty in it. In our case the uncertainty was mostly due in the format of input data and the restrictions we were required to impose. The final input format was agreed and needed no further modifications in around mid July and the restrictions we implemented were modified till the end of August. Having guessed such development, the robustness and flexibility of our model in Pulp, Python was quite essential. As the restrictions requirements were modified, we were able to see the amount of change it had on objective function and the actual assignment of courses, This is called the post-optimal analysis, initially we assumed all students have equal preference and were all taking English Literature courses, but we were later informed that the students can take courses outside of English literature. This complicates the problem as, first we were required the data for the course and the various restrictions which go along with it. Second, courses outside of English literature do not have the minimum students requirement hence we need to modify the program to only apply the minimum students requirements to courses from English literature. This in turn requires to implement a method to differentiate courses of English literature from others. The method we implemented include changing the course ID from a only numeric format to an alpha numeric format and identifying the course category based on it. As mentioned earlier we modified our objective function tremendously due to the changing restrictions this in turn helped in debugging the code and identifying smaller issues. This analysis was useful as it aided in formalizing the priorities of the Glasgow School of Arts administration and also helped us to implement a software for the same. After implementing the model which is planned out, it is important to verify that the implementation is consistent with the planned out model as there may be programming errors which were missed out while debugging, which lead to sub-optimal solutions and to actually check the correctness of the mathematical model. We have already shown informally the equivalence of the coded constraints and the planned ones and have documented the same in the program with help of comments.

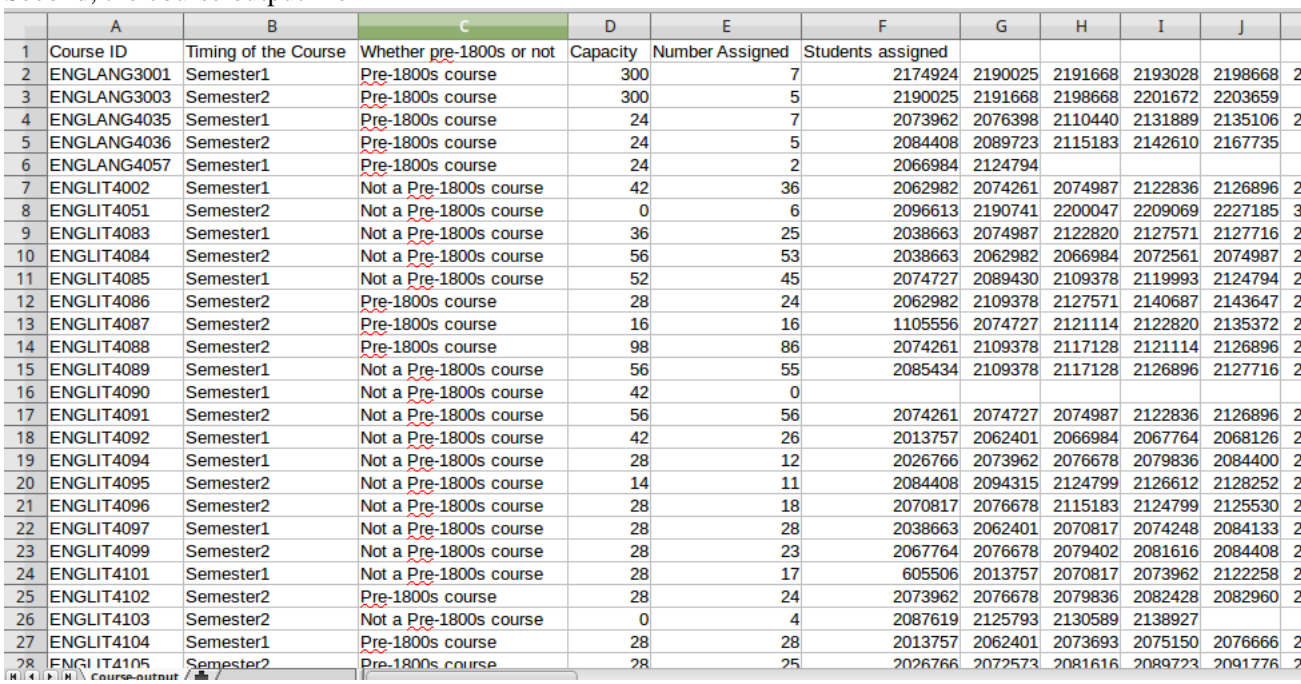
- Presenting the solution and analysis
  The entire process, gathering data, planning, modeling, implementation, testing and debugging(post-optimal analysis) finally boils down to the presentation of the solution obtained, the easy of use of the program and the simplicity of use.
  For testing purpose, we created a random test case generator which generates sample input files for the program, which we then ran to check for bugs, run time errors and quality of

output so that we will be able to get heuristics.
The random test case generator works as follow:-
We provide the test case generator with the number of courses and the number of students we want in the sample test case. It is important to note that there is a relation between number of students and number of courses due to the nature of restrictions given. As no student can have more than 6 courses and all English Literature course must have at least 8 students. We initially assign 2 courses to each of the category of pre-1800s course in semester 1, pre-1800s course in semester 2, not pre-1800s course in semester 1, not pre-1800s course in semester 2. Then we randomly assign capacity to them. Then we proceed to add the required number of courses and assigning them as pre-1800s course or not and the semester they are scheduled in along with a capacity. After assigning the details for each course, we added them in one of the 4 category depending on the details generated. After constructing the courses data, proceeded to generate data for each student. For each student we randomly decided if they are in third or fourth year and if they are doing a single or joint program. Then depending on this, we decided the number of pre-1800s courses they need to take and they dissertation preference or if they are doing dissertation at all or when they are taking the core course. After this we created a random preference list of twice the length of the number of courses they are required to take equally distributed so that they have choice in step of the process.
We outputted this in Students.csv and Courses.csv file, during the testing of the program, we kept the number of students to be around 190 and number of course to be around 20. which were almost half of the real number, hence they actual running time was much higher compared to the testing.
After obtaining the values for the decision variables we need to translate it back to values which are meaningful to the user. Each step of converting the problem description to model, obtaining values and reconverting the values into concise and comprehensible summary is equally important. The key observations need to be mentioned in a clear wave. We have done this with by outputting outputting the student assigned to course file, the course assigned to students file and the admin file which tells the details of the assignments.
Presenting sample output of the program on the data given by the Glasgow School of Arts.
First, the student output file

	A	B	C	D	E	F	G	H	I	J	K
1	605506	4	J	Semester 1	20	ENGLIT4101					
2	605506	4	J	Semester 2	40	Dissertation					
3	1105556	4	J	Semester 1	40	Dissertation					
4	1105556	4	J	Semester 2	20	ENGLIT4087					
5	2013757	4	S	Semester 1	60	ENGLIT4092	ENGLIT4101	ENGLIT4104			
6	2013757	4	S	Semester 2	60	ENGLIT4111	Dissertation				
7	2026766	4	S	Semester 1	60	ENGLIT4094	ENGLIT4109	ENGLIT4122			
8	2026766	4	S	Semester 2	60	ENGLIT4105	Dissertation				
9	2030262	4	J	Semester 1	20	ENGLIT4107					
10	2030262	4	J	Semester 2	40	Dissertation					
11	2038663	4	S	Semester 1	60	ENGLIT4083	ENGLIT4097	ENGLIT4106			
12	2038663	4	S	Semester 2	60	ENGLIT4084	Dissertation				
13	2062401	4	S	Semester 1	60	ENGLIT4092	ENGLIT4097	ENGLIT4104			
14	2062401	4	S	Semester 2	60	ENGLIT4125	Dissertation				
15	2062982	3	S	Semester 1	40	ENGLIT4002	ENGLIT4121				
16	2062982	3	S	Semester 2	40	ENGLIT4084	ENGLIT4086				
17	2063534	4	J	Semester 1	20	ENGLIT4110					
18	2063534	4	J	Semester 2	40	Dissertation					
19	2066935	4	J	Semester 1	40	Dissertation					
20	2066935	4	J	Semester 2	20	ENGLIT4111					
21	2066984	4	S	Semester 1	60	ENGLANG40	ENGLIT4092	ENGLIT4106			
22	2066984	4	S	Semester 2	60	ENGLIT4084	Dissertation				
23	2067764	4	S	Semester 1	60	ENGLIT4092	ENGLIT4106	ENGLIT4110			
24	2067764	4	S	Semester 2	60	ENGLIT4099	Dissertation				
25	2068126	4	J	Semester 1	20	ENGLIT4092					
26	2068126	4	J	Semester 2	40	Dissertation					
27	2070817	4	S	Semester 1	60	ENGLIT4097	ENGLIT4101	ENGLIT4110			
28	2070817	4	S	Semester 2	60	ENGLIT4096	Dissertation				

The first column of each row indicates the unique student ID, this is an identifier for each student. the next 2 columns show the details of the student, 4 indicates a fourth year honors student and 3 represents a third year honors student, S indicates a Single honors student and J indicates a Joint honors student. The next 2 columns show the which semester details are going to be shown and how many credits were taken in the semester. All the columns after is show the courses assigned to the student.

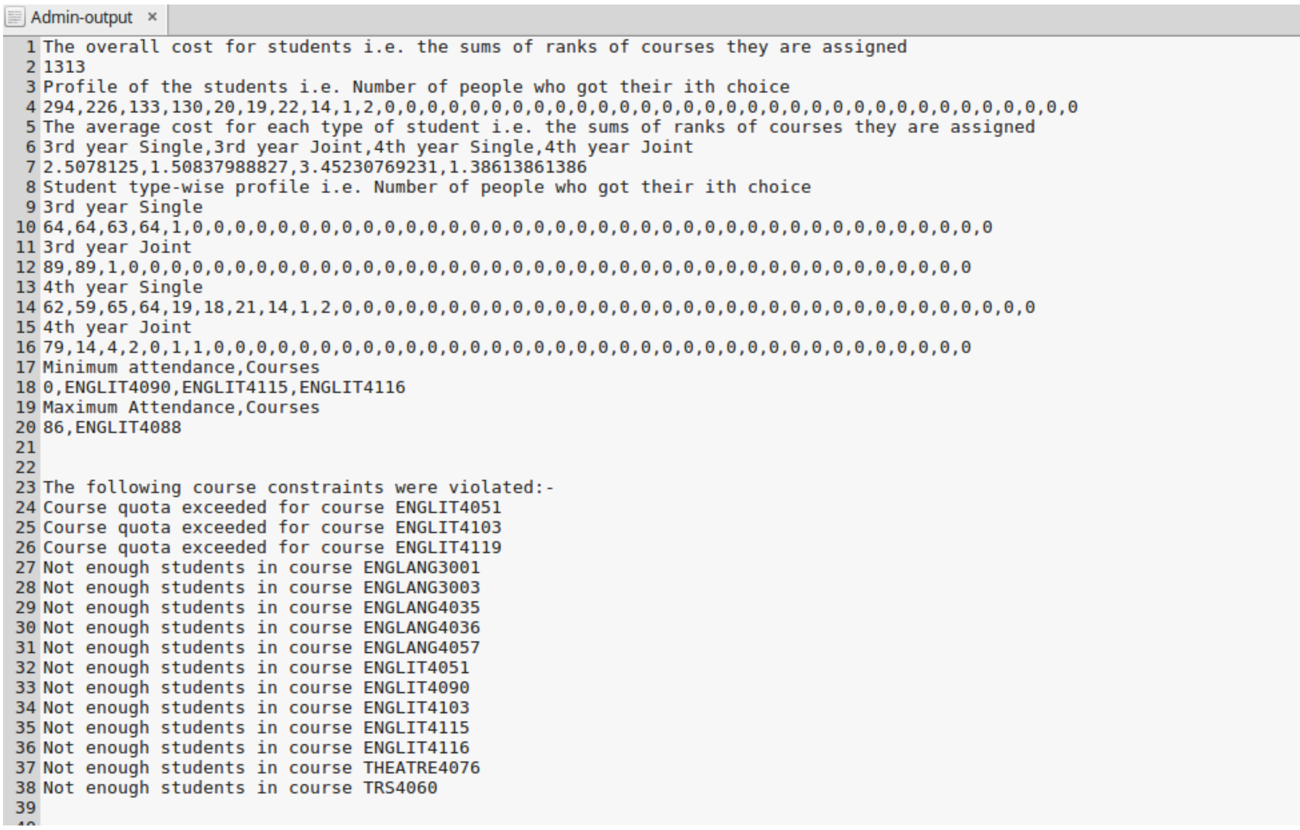
Second, the course output file



	A	B	C	D	E	F	G	H	I	J
1	Course ID	Timing of the Course	Whether pre-1800s or not	Capacity	Number Assigned	Students assigned				
2	ENGLANG3001	Semester1	Pre-1800s course	300	7	2174924	2190025	2191668	2193028	2198668
3	ENGLANG3003	Semester2	Pre-1800s course	300	5	2190025	2191668	2198668	2201672	2203659
4	ENGLANG4035	Semester1	Pre-1800s course	24	7	2073962	2076398	2110440	2131889	2135106
5	ENGLANG4036	Semester2	Pre-1800s course	24	5	2084408	2089723	2115183	2142610	2167735
6	ENGLANG4057	Semester1	Pre-1800s course	24	2	2066984	2124794			
7	ENGLIT4002	Semester1	Not a Pre-1800s course	42	36	2062982	2074261	2074987	2122836	2126896
8	ENGLIT4051	Semester2	Not a Pre-1800s course	0	6	2096613	2190741	2200047	2209069	2227185
9	ENGLIT4083	Semester1	Not a Pre-1800s course	36	25	2038663	2074987	2122820	2127571	2127716
10	ENGLIT4084	Semester2	Not a Pre-1800s course	56	53	2038663	2062982	2066984	2072561	2074987
11	ENGLIT4085	Semester1	Not a Pre-1800s course	52	45	2074727	2089430	2109378	2119993	2124794
12	ENGLIT4086	Semester2	Pre-1800s course	28	24	2062982	2109378	2127571	2140687	2143647
13	ENGLIT4087	Semester2	Pre-1800s course	16	16	1105556	2074727	2121114	2122820	2135372
14	ENGLIT4088	Semester2	Pre-1800s course	98	86	2074261	2109378	2117128	2121114	2126896
15	ENGLIT4089	Semester1	Not a Pre-1800s course	56	55	2085434	2109378	2117128	2126896	2127716
16	ENGLIT4090	Semester1	Not a Pre-1800s course	42	0					
17	ENGLIT4091	Semester2	Not a Pre-1800s course	56	56	2074261	2074727	2074987	2122836	2126896
18	ENGLIT4092	Semester1	Not a Pre-1800s course	42	26	2013757	2062401	2066984	2067764	2068126
19	ENGLIT4094	Semester1	Not a Pre-1800s course	28	12	2026766	2073962	2076678	2079836	2084400
20	ENGLIT4095	Semester2	Not a Pre-1800s course	14	11	2084408	2094315	2124799	2126612	2128252
21	ENGLIT4096	Semester2	Not a Pre-1800s course	28	18	2070817	2076678	2115183	2124799	2125530
22	ENGLIT4097	Semester1	Not a Pre-1800s course	28	28	2038663	2062401	2070817	2074248	2084133
23	ENGLIT4099	Semester2	Not a Pre-1800s course	28	23	2067764	2076678	2079402	2081616	2084408
24	ENGLIT4101	Semester1	Not a Pre-1800s course	28	17	605506	2013757	2070817	2073962	2122258
25	ENGLIT4102	Semester2	Pre-1800s course	28	24	2073962	2076678	2079836	2082428	2082960
26	ENGLIT4103	Semester2	Not a Pre-1800s course	0	4	2087619	2125793	2130589	2138927	
27	ENGLIT4104	Semester1	Pre-1800s course	28	28	2013757	2062401	2073693	2075150	2076666
28	ENGLIT4105	Semester2	Pre-1800s course	28	25	2026766	2072573	2081616	2089723	2091776

As indicated, the first column in each row shows the course ID of the course for which the details are going to be presented. The second columns tells in which semester the course is going to be offered. The third columns tells whether the course qualifies as a pre-1800s course. The fourth and fifth column tell indicate the capacity of the course and the number of students assigned to it. From there on, it is the list of students who are assigned to this particular course.

Third and last, the admin output file



```
The overall cost for students i.e. the sums of ranks of courses they are assigned
1313
Profile of the students i.e. Number of people who got their ith choice
294,226,133,130,20,19,22,14,1,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
The average cost for each type of student i.e. the sums of ranks of courses they are assigned
3rd year Single,3rd year Joint,4th year Single,4th year Joint
2.5078125,1.50837988827,3.45230769231,1.38613861386
Student type-wise profile i.e. Number of people who got their ith choice
3rd year Single
64,64,63,64,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
3rd year Joint
89,89,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
4th year Single
62,59,65,64,19,18,21,14,1,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
4th year Joint
79,14,4,2,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
Minimum attendance,Courses
0,ENGLIT4090,ENGLIT4115,ENGLIT4116
Maximum Attendance,Courses
86,ENGLIT4088


The following course constraints were violated:-
Course quota exceeded for course ENGLIT4051
Course quota exceeded for course ENGLIT4103
Course quota exceeded for course ENGLIT4119
Not enough students in course ENGLANG3001
Not enough students in course ENGLANG3003
Not enough students in course ENGLANG4035
Not enough students in course ENGLANG4036
Not enough students in course ENGLANG4057
Not enough students in course ENGLIT4051
Not enough students in course ENGLIT4090
Not enough students in course ENGLIT4103
Not enough students in course ENGLIT4115
Not enough students in course ENGLIT4116
Not enough students in course THEATRE4076
Not enough students in course TRS4060
```

The admin output file has many components and required the most work to output. This file is important as this tells the administrators about the quality of the output obtained from the program, these quality and observation documents can help the administration decide whether to use the program or not.

First, we have the total score of the ranks of courses assigned. Second, we show the number of students getting their i^{th} choice. Third, we have the average choice of course each type of student has obtained. Fourth, we show category wise the number of students getting their i^{th} choice. Fifth, we have the courses with minimum attendance and the list of courses with that attendance and similar with maximum attendance. Sixth and last a description of all the constraints which are violated and a small reason for violation, the real cause for violation might require to manually look into it.

In general this step can help understand the further development of the project, we discuss more details in the next chapter, which can include:-

- As the restrictions may change in future it is require to check if the program is still applicable and make necessary changes.
- As the time frame in which we implemented this program was quite short for a detailed study of the problem, further analysis and more restrictions can be incorporated so that the client can benefit more.
- With the success of this program further applications for optimization maybe found.

We also got feedback from the administrators at the Glasgow School of Arts regarding the output of our program.

Chapter 5

Conclusion

Before the start of this project I had some knowledge regarding Integer Linear Programming and about how to write a literature review. This project has helped me improve my literature review writing skills, provided me first hand experience with developing a Integer Linear Programming model with practical applications.
When I was writing the literature review for the project, I had to prioritize obtaining literature relevant to the project. For instance, the vast amounts of papers related to Integer Linear Programing and mixed Integer Linear Programing are available. However finding material which is closely related to this project required use to many keywords and many websites.
I believe the experience of efficient in finding related quickly offers substantial benefits, as it saves time and increases validity and overall knowledge. Critically analyzing the literature and the software available helped me understand more about the topic, pros and cons of each . This has overall helped develop critical thinking skills in areas except for mathematics.

I have obtained research skills and software development skills during this project. This was a valuable experience where I participated in data formatting, analysis and presentation. I got first hand knowledge of qualitative and quantitative research methods thanks to this project.
The real data this time was obtained by manually creating an excel sheet from the data given on paper individually by each student. We thought about methods to make this into an online system so that the process is more efficient.
By viewing the real data, we noticed the importance of the client explaining the process of data collection to the student and why it is important for them. As in some cases students provided such preference list that it is clear that there is not possible way to assign them courses without violating the constraints.
I believe that I have harvested many benefits from the research and software development experience along with my time management skills, as the the project required planning and preparation for all steps, as there were project specific deadlines to be met.
The most challenging part in ensuring the progress is up to mark was in the literature review step. I overestimated my ability to read through the papers quickly and was behind the schedule in this step. The issue was taken care of by speeding up other steps of the process and following the schedule strictly.
During the project, other activities were given less priority by establishing daily goals and recreational activities were limited by the progress made towards the goal. Hence I believe this project has helped me improve my time management skills which are helpful to me on both professional

and personal level.

This project provided me with opportunities to speak directly to the client, asking them for clarification and suggesting work around. This resulted in increase of my self-confidence.
At the start of the project I was a bit hesitant as I was inexperienced in this types of discussions and could only provide few comments, later I was openly able to point of error in my code, other train of thought, the conclusions the client drew from my observations and also suggest methods to collect data for the program. Because, currently enormous amount of time was spent waiting for the real data, so as to prevent this from happening every year we tried to come up with methods to collect data in a proper format so that it can directly be provided as input to the program. I do believe that the discussions with the supervisors and the client have improved my communications skills.
As said, for further work we had the following ideas:-

- As the newer batches come, the curriculum will change. This will make the current software unusable. So it is worth the effort to rewrite the program so that it can be easily modified for further use.
- Currently the program is command line based, which can be hard to use for a beginner. Hence, a GUI can be implemented which can make things easier to interact with. The code can also be modified so that it can detect errors in the input file and either do the modifications by itself or report it in an easy ti understand fashion.
- It also might be possible to generalize constraint to such a level so that constraints need not be hard coded, but rather a constraint input file can be provided so that it tells about all the constraints requirements from the clients perspective.

Appendix A

First appendix

To run the program, first need to install working version of python 2 on the platform. Second, need to install pulp package for python 2, this can be done by using the command
"pip install pulp"
There are not further technical requirements. Now to run the program you need to Students.csv and Courses.csv file in the same folder as assigner.py. To run the program change open command line, change directory into the folder with assigner.py and execute the following command
python assigner.py
The format of Course.csv should be as follows:-
Each row contains data for a separate course, the first column contains the unique course ID, the column indicates the semester in which the course is offered in, 1 for first semester, 2 for second semester, third column tells if the course is a pre-1800s course, Y for yes it is, N for no, it is not. Fourth column simply represents the capacity of the course.
The format of Students.csv should be as follows:-
Each row is for a unique student. The first column should be the unique student ID, the second column should be the the year the student is in, 3 for third year and 4 for fourth year, the next column tell the program the student is in, S for Single and J for Joint. The next column represent the number of pre-1800s courses the student has to take. The next column for third year Single students has to be -1, for third year Joint students, this should indicate the semester in which they are going to take their core course in, for fourth year single students it should indicate the preference of semester when they plan to do their dissertation, 0 for no preference, 1 for first semester, 2 for second semester and for fourth year joint students this indicates if they are going to take a dissertation and if they are, then the preference. All the columns after this indicate the preference of course from highest to lowest.
The output of the assigner program are 3 output files which are explained earlier. I have implemented some test checker which will make sure if the input is in the correct format and correct few errors.

A.1 Section of first appendix

Appendix B

Second appendix