6CCS3PRJ Final Year Project
Task Allocation System

Background Research & Specification Report

Author: Hussain Miah

Supervisor: Dr Tomasz Radzik

Student ID: 1114801

Date: 5th December 2014

Abstract

For my final year project I will be making an application to help facilitate the job of assigning employees within an organisation to tasks that the company has agreed to embark on for their projects. This task allocation system is very important not only in the real world but also in the physical hardware universe as well, where this same problem can be modelled by a load balancing system within a CPU, where a number of processes arrive and the load balancer within the computer has to determine which processes to execute next depending on various factors ranging from the priority of the processes that are arriving or getting the most performance out of the CPU as possible.

This task allocation system will be useful to any organisation using the system as it removes the manual tedious work that will be done by the manager when new projects and jobs are signed by the company. The application will offload the matching process that the manager will typically where they will look at the skills that are available within the company and trying to match them to the tasks that they have been given.

The main task of this application will be to have a graphical user interface, where the user will be able to input the organisations details (Employees, Tasks and Skills) and then see suggested allocations depending on the matching algorithm that they have chosen.

Introduction

The concept of matching is an important topic within graph theory. The matching of graphs can be divided into a number of categories, the focus of this final year project will be to find the maximum matching of employees to tasks. A maximum matching is when the matching graph has the maximum number of edges that are possible[1].

Regarding this project a matching will exist between an employee and a task when the skills of the employee match the skills of the task and the employee is available to take on the task.

The project deliverable will be a software system where the user will be able to import the organisations data and then choose a matching algorithm of their choice, and the system will compute the matching using the data that has been provided and the specific algorithm that the user has chosen. After the user has been presented with the suggested mapping they will be able to assign the specific matched task to the employee to be reflected within the database. They will also have the option to save the suggested allocation to a file to be used at a later date.

The system can also be used to predict future demands on the company in terms of the resources that are available to them and the skill set that is within. The company can then take further action where necessary whether it be recruiting more people or training the existing employees that they already have to match the skills of the tasks they are interested in.

The main difficulty with this project will be determining a suitable and efficient (running in polynomial time), maximal matching algorithm, where it will take as input the data structure retaining the employee to task matching and compute the maximal matching, which will be the output of the algorithm.

Data Structures

There are two main options when it comes to choosing the data structure that will be used in this project to hold the matching from employees to tasks.

The first option is to use an adjacency matrix,where this will be represented as a two-dimensional array defined as EMPLOYEES x TASKS. A match between an employee and task will be denoted as there being a 1 in the cell and 0 otherwise.

	Tasks					
Employees	0	1	0	1	1	1
	0	0	1	1	1	0
	1	1	0	1	0	0
	0	0	1	0	1	1
	1	0	1	1	0	0
	0	1	1	1	1	1
	1	0	1	1	1	1

ADJACENCY MATRIX TO SHOW EXAMPLE EMPLOYEE TO TASK MAPPING

The above shows an adjacency matrix for a suggested allocation. The diagram shows the drawback that associated with using this data structure. The matrix holds unnecessary information in the sense that when an employee has not been matched with a task, there is still an element within that cell which is taking up space. Using a matrix also makes it difficult all the tasks that an employee has been allocated or vice versa, for example to find all the suggested tasks that have been allocated to the employee at matrix index 3, we would have to iterate over that entire row within the matrix to find the tasks, and as the number of tasks and employees grow within the organisation the space and time complexity to query the matrix will grow and ultimately make the process of finding the matching and presenting the information to the user much longer.

A graph is the final data structure of choice within this project. This graph data structure will be one that I will implement my own within this project. The graph will have two sets of vertices, one set for the employees and final set for the tasks. The graph will then have an internal map (Java has a HashMap Object) which will hold the mapping from the employee to the task that they have been matched up with. As I will be using a map to store the matching, the act of finding the task that the employee has been assigned to will be a quick case of querying the internal hash map, which overcomes the previous performance issue that was encountered when using the matrix data structure. By using the graph data structure we are only storing the information that is required by the application and there is no excess bloat information stored.

Graphs will be discussed in more detail further in the report.

Background

The way in which the matching will be modelled internally will be by the use of a Bipartite graph. A bipartite graph G = (V,E), where V is the set of vertices within the graph. This vertex set is further partitioned further into two distinct sets; where V = L ∪ R. L is the left hand set and R is the right hand set of the graph. The edges only connect vertices in set L to vertices in set R. A matching M is subset of the edges that are present in the graph, M ⊆ E[2].

Bipartite Graphs

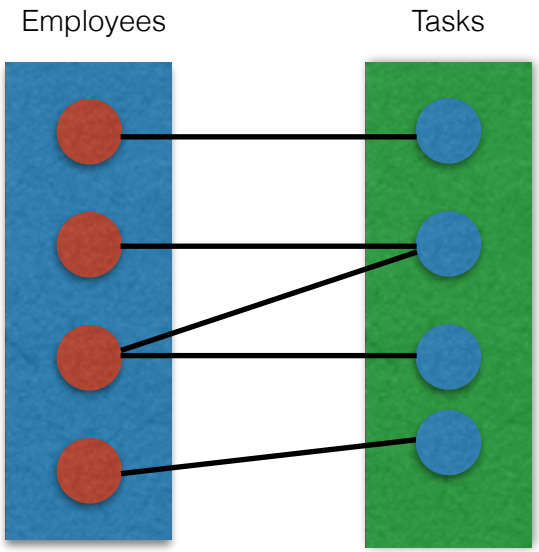

FIGURE 1: EXAMPLE BIPARTITE GRAPH

The above Figure 1 shows how the problem of matching employees to tasks can be modelled by a bipartite graph. The two distinct set of vertices is the employees set on the left and the tasks set located on the right. An edge between a employee node and task node denotes that the employee has the right skills to do the tasks.

A bipartite matching algorithm can then be executed over the above graph to find the maximum matching where there will be a one-to-one mapping from employee to task, an example can be seen below in Figure 2.

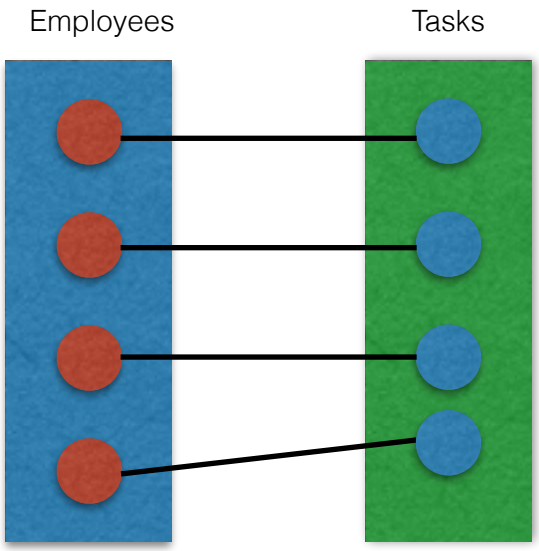

FIGURE 2: BIPARTITE GRAPH AFTER MAXIMUM BIPARTITE MATCHING ALGORITHM HAS BEEN COMPUTED

Figure 2 shows how a maximal one-to-one matching can be computed using the original matching graph in Figure 1.

Maximum Bipartite Matching and Max Flow Problem

The approach that I will take to compute the maximum bipartite matching is by transforming the original matching graph into a relaxed flow network. In graph theory a flow network is a directed graph where every edge will have a capacity in conjunction with a flow for the edge[3]. I will be using a relaxed version of a flow network because I will be adding a capacity of 1 to every edge within the matching graph but there will not be a flow associated with the edges within the graph.

The way in which the flow network differentiates its self from the matching graph is that there are two additional nodes that will be added to the graph called the source and sink nodes. The source node will map to each node within the EMPLOYEES set in the graph, while on the other hand, every node in the TASKS set will map to one single sink node.

The purpose of adding these two nodes will be to find the maximum flow between the source and sink node for every node within the EMPLOYEES set, where we will only be able to traverse edges that were in the original matching graph. Finding the max flow if a graph will identify all the different augmentation paths that exist within the network. An augmentation path is a simple path; a path that does not contain any cycles, using only edges within the graph that have a positive weighting from the source node to the sink node[4]. Finding the maximum flow will compute the maximum bipartite matching that is required. The algorithm that will be used to find the maximum flow of the network graph will be the Ford-Fulkerson Algorithm for the Maximum Flow problem[5].

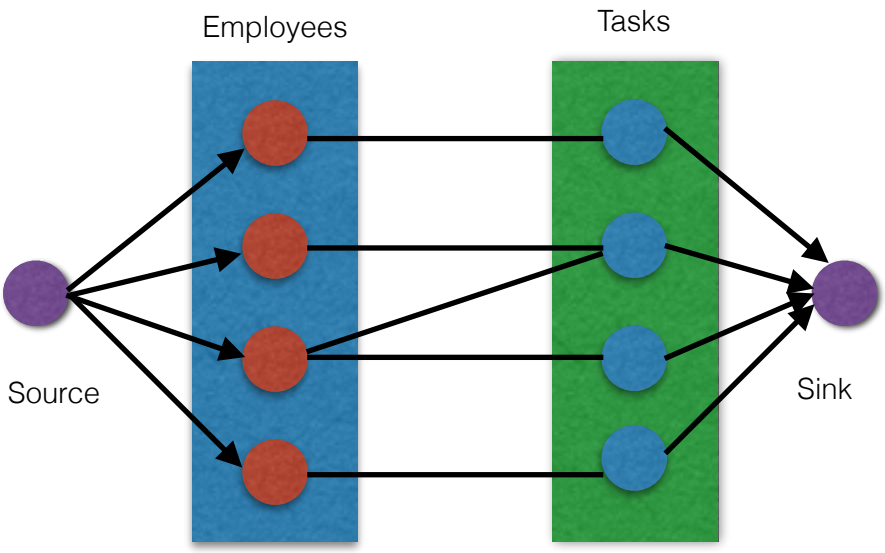

FIGURE 3: TRANSFORMATION OF ORIGINAL MATCHING GRAPH INTO FLOW NETWORK

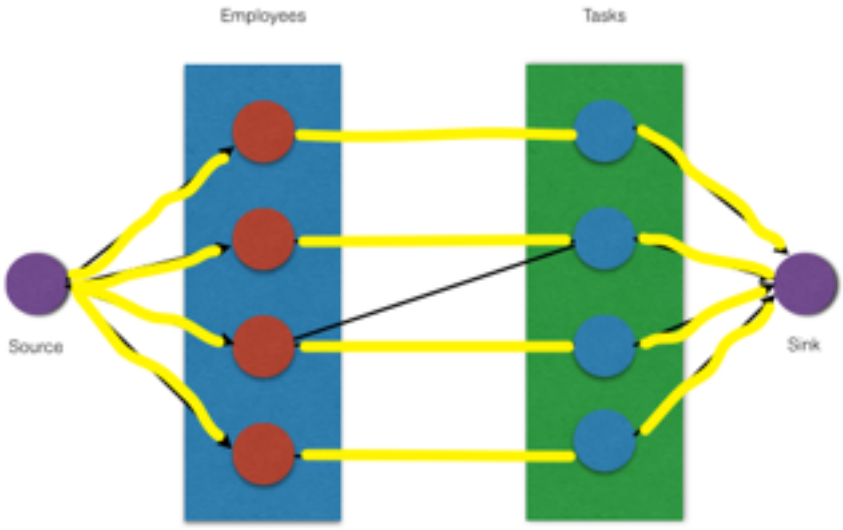

FIGURE 4: FLOW NETWORK AFTER AUGMENTATION PATHS HAVE BEEN COMPUTED USING FORD-FULKERSON ALGORITHM

Figures 3 and 4 show how a bipartite graph can be transformed into a network flow graph which will then allow us to find the maximum flow between the source and sink node to find the find the maximum bipartite matching using Ford-Fulkerson's algorithm. The output the maximum matching can be seen earlier in this report in Figure 2.

Matching Algorithms

The application will allow for two main matching algorithms that can be used to assign the employees to the tasks that they have been matched up with.

The first being a greedy approach where the algorithm will match the first employee in which their skills are equal to the skills that are required by the task.

The final matching algorithm will be to find the maximum number of tasks that can be completed taking into consideration all the employees that are suitable for the tasks that they have been matched up with.

A polynomial time algorithm has been found to find the maximal matching of a bipartite graph ,the pseudocode for the algorithm is detailed below:

```
Algorithm HOPCROFT-KARP
B ← ∅
PUSH s onto STACK
While STACK is not empty do
begin
        While LIST(TOP) is not empty do
        begin
                FIRST=first element from LIST(TOP)
                If FIRST ∉ B then
                begin
                        PUSH FIRST onto stack and delete it from LIST
                        If TOP ≠ t
                                then B ← B ∪ {TOP}
                                else
                                        begin
                                                empty the stack and print the path
                                                PUSH s
                                        end
                end
                POP STACK
        end
end
```

HOPCROFT-KARP ALGORITHM TO FIND THE MAXIMUM MATCHING OF A BIPARTITE GRAPH

The algorithm follows the same lines as the Ford-Fulkerson's algorithm where by it is computed over a network flow graph. The algorithm uses a stack data structure keep note of the nodes that have been visited and the aim of the algorithm is to find a path from the source node (denoted as S in the above pseudocode) to the sink node (denoted as T in the pseudocode).

The running complexity of the algorithm is a polynomial of $O(N^{5/2})$.

Requirements Analysis

In order to help guide me in the developing the solution for this final year project, it will be beneficial to have a set of requirements that will need to met by the end of the project. Having these requirements listed defines goals and milestones to be reached within the project and there is an end goal to work towards and achieve.

- **A database schema will exist to retain the organisations information**
  The application will be using a MySQL database in the backend to store information relating the organisation. A script will be provided as a deliverable for this project in order to create the database and the tables within it.
- **The user will be able to import the organisations information using a CSV file.**
  To overcome the tedious task of entering the organisations information into the database manually using a form within the application, the user will be able to a CSV file into the application which will handle the task inserting the rows specified in the file into the database. Each table will have its own corresponding file that will need to be used when importing data into the table. See below for an example of a file, as the file has to be in a certain format for it to be accepted by the application and inserted into the database.
- **There will be a greedy alphabetical matching algorithm**
  A greedy approach can be taken where the name of the employees are retrieved from the database ordered by their name alphabetical. This greedy approach will take the employees in the order that they are retrieved and assign the first employee who is suitable for the task.
- **There will be a greedy cost matching algorithm**
  This greedy approach will try and maximise the profit of the organisation by assigning tasks to employees that have a lower daily cost compared to other employees with the same skills or better.
- **A maximum matching**
  This matching algorithm will take all the employees and tasks that are within the database and internally assign the employees to all the tasks that they are suitable for. After this initial matching is completed an algorithm will be used to compute the maximum number of tasks that can be completed using the initial matching.
- **The application will take into account employees that have already been assigned to tasks**
  There is a very high likely hood that there will exist a set of employees that have been assigned to tasks, the user will have the option to tick a check box to add this extra constraint in the matching process. So when a matching is being computed it will have to look at the start and end date of the task that has been assigned to the employee and see if there is a clash between this date and the start and end date of the new proposed task to be matched with the employee.
- **The user will be able to save the suggested allocation of employees to tasks to a file**
  After the user is presented with suggested allocation will have the opportunity to the save the allocation to a file, to be used at a later date if so required. They will also be able to open this file within the application again as well to start assigning tasks to employees. See below for an explanation of the file.
- **The user will be able to assign the tasks that have been suggested by the application**
  The information that is presented to the user can then be updated and reflected in the database to say that a specific employee has been assigned to a task. There will be a

checkbox in every row of the allocation table for the user to choose which employees the user would like to assign to tasks.

CSV File Format

A CSV file will be used to import the organisations data into the database, the CSV file does not need to have a special file name, but the extension must end in **.csv**.

Below is an example of a CSV file to import into the **EMPLOYEES** table.

ID,FIRST_NAME,LAST_NAME,SEX,COST

1,Ben,Jones,M,1000

2,Sophie,Baxter,F,500

Task Allocation File Format

One of the requirements of the application is the save the suggested allocation to a file to be used at a later date. The file extension that I decided to use to denote this file will be a file with the extension **.ta**. The Task Allocation File is based off an XML file, where its contents will contain a number of tags describing the various properties and attributes relating to the employee.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<taskAllocation>
    <employee id="0">
        <name>James Foster</name>
        <taskAssigned>t1</taskAssigned>
        <taskID>012</taskID>
        <assigned>false</assigned>
    </employee>
    <employee id="1">
        <name>Amy Pointer</name>
        <taskAssigned>t2</taskAssigned>
        <taskID>014</taskID>
        <assigned>true</assigned>
    </employee>
</taskAllocation>
```

EXTRACT 1: EXAMPLE OF A TASK ALLOCATION FILE, SPECIFYING TWO EMPLOYEES AND THEIR SUGGESTED ALLOCATED TASK

System Architecture

I will divide my system into three distinct modules which will hold implementation details relating to that module. The three modules that I will define are: Front end, Functional Core and Data.

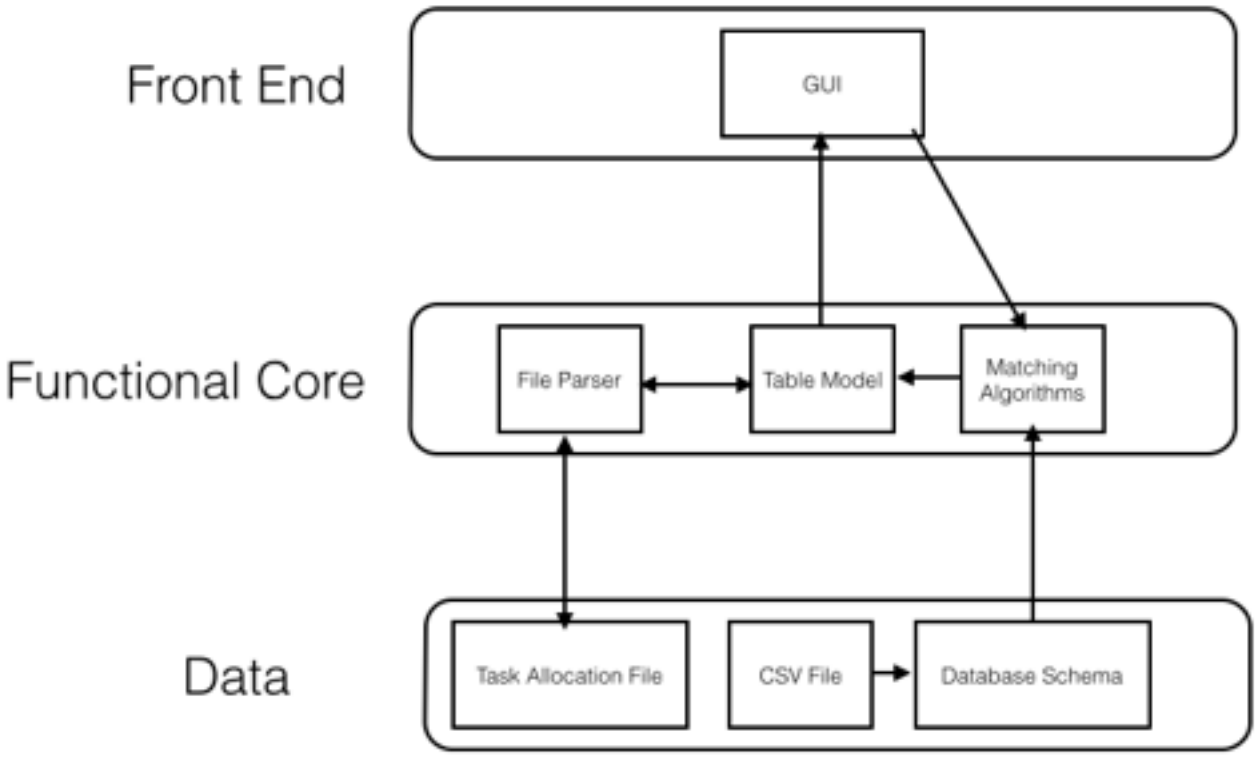

FIGURE 5: SYSTEM ARCHITECTURE FOR TASK ALLOCATION SYSTEM

Front end: The front end will house a graphical user interface which will have various screens. There will be a user interface for the to browse their file system and select an appropriate CSV file to be imported into the database. There will also be a user interface, where the user will be able to choose the matching algorithm of their choice and it will present the outcome to the user within the same instance in a table.

Functional Core: The functional core will contain the logic to save the table presented to the user as a task allocation file, this is also where the logic will be contained to read the task allocation file stored on the users machine and transform the file into a model that can be used to populate the allocation table.

The table model will define the columns and behaviours of the Task Allocation table.

The matching algorithm will take the desired approach selected by the user and the data that is stored with the database to produce a suggested allocation and present that within the Task Allocation table.

Data: The Task Allocation file is a file representation of the Task Allocation table. The database and the CSV file is also stored within this module.

Database Design

The below figure shows the entity relationship diagram that will be used to design the database for the Task Allocation system.

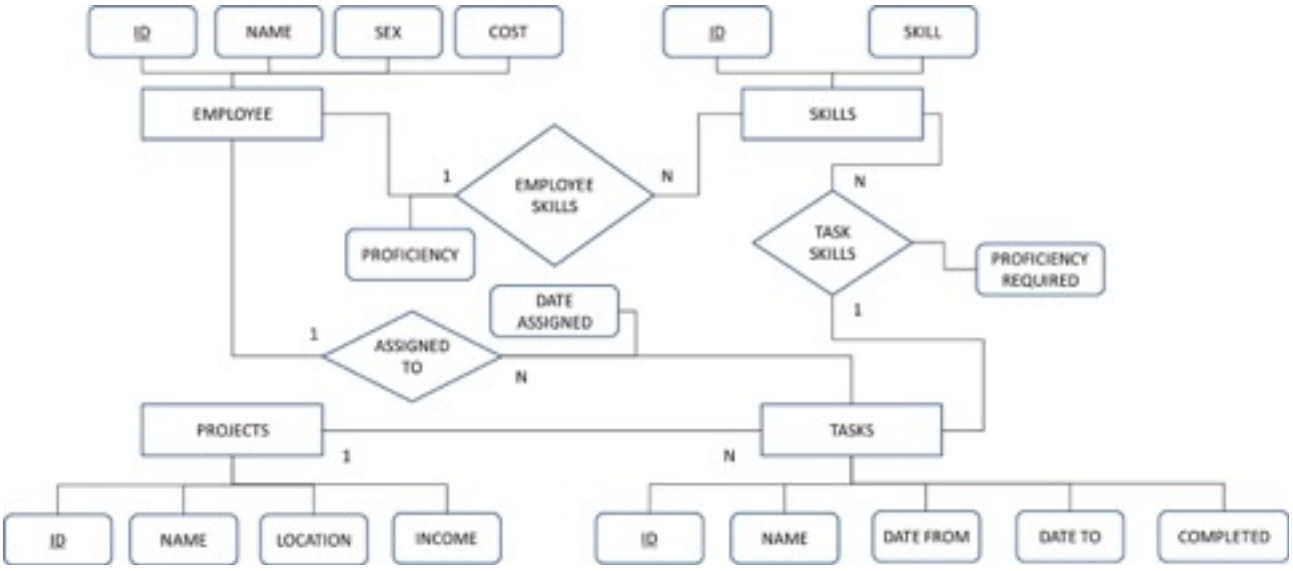

FIGURE 6: ENTITY RELATIONSHIP DIAGRAM OF THE TASK ALLOCATION SYSTEM

Application and Allocation Design

The design of the application will exploit the use of inheritance within the Java Application and I will also be using Java Packages to my advantage as well which will allow me to group similar classes and methods within packages to make the layout of the project easier to follow for a person following the code.

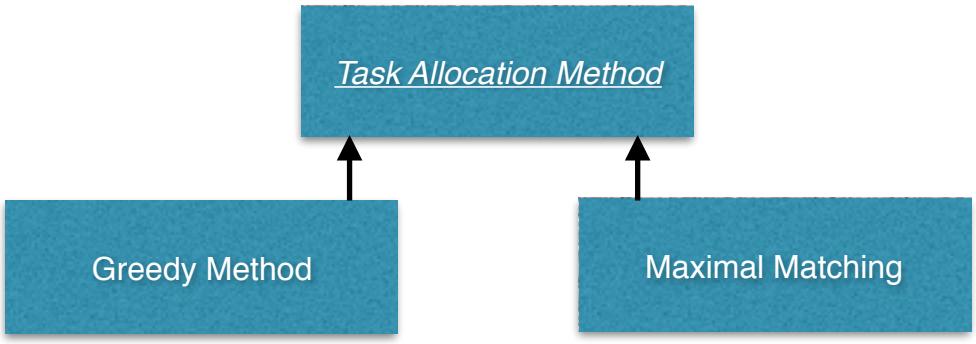

FIGURE 7: CLASS DIAGRAM OF THE TASK ALLOCATION METHOD ALGORITHM

The above class diagram shows the design of the Task Allocation Algorithm, where a super base class (Task Allocation Method) will hold all the common properties and methods that will be used by both subclasses. For example the purpose of the Task Allocation Method class will be to query the database and to return the employees and tasks within the database and all the other information relating to the employees and tasks like the skills that the employees know and the skills that are required by the tasks in order for them to be completed. The other purpose of the super class Task Allocation Method is to build the partial graph that will be used to store the employee to task mapping. The Task Allocation Method class will not assign or in other words map any of the employees to the tasks but will build the two distinct sets that will be used by the two

subclasses, so the Task Allocation Method will build the employee set of the graph and tasks set of the graph. This Task Allocation Method class will also have the extra property of being an Abstract class, which in Java means that you will not be able to instantiate an instance of this class, within this abstract class there will be an abstract method; **allocateTasks,** the reason why I have chosen to make this method an abstract method is to make sure that every subclass of this class will implement this method as every new Allocation Method will have a different way of allocating the tasks to the employees.

The subclasses of the super class will need to implement the **allocateTasks** method that they will inherit and within this method the class will be able to utilise the employees and task nodes that have been created before during the query stage of the Task Allocation method to actually allocate the employees that they have been matched up with, this is where the edge between the employee node and the task node will be added. In other words the mapping will be added to the internal hash map that is used by the graph object.

Development Analysis

The Task Allocation System will be written using the Java programming language and I will be using version 7 of Java on my development machine. I will be using the Java Swing GUI framework to create the user interface of the application. I will also be using the Apache Log4J[6] library to both write logs to a file stored on the machine and to the user interface of the application so the user will be able to see the application log its progress as it is executing, this overcomes the issue of the user closing the application and then navigating to the folder where the logs are being stored and open them to read their contents to see if there are any errors or not.

This Swing Log4J appender is an open source piece of software that I am including in my final year project[7].

The database management system that I will be using in this project will be the free and open source DBMS MySQL. I will be providing a script that will create the database and the user that will be used to query the database within the Task Allocation System. I will not be providing the MySQL executable within my project, I will assume that MySQL is already installed on the machine that the application will be running on and I will not be providing instructions on how to install MySQL on a machine.

To connect my programming logic to the database I will be using the JDBC library[8] which will allow me to the connect to the database and once a connection has a been established, the library provides functions to query and manipulate the database by the way of SELECT and INSERT and UPDATE statements.

Bibliography

[1]: http://www.geeksforgeeks.org/maximum-bipartite-matching/

[2]: John E. Hopcroft and Richard M. Karp
Reviewed by David Jackson
“An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs”

[3]: http://en.wikipedia.org/wiki/Flow_network

[4]: http://stackoverflow.com/questions/10397118/graph-what-exactly-is-augmenting-paths

[5]: Charles E. Leiserson, Thomas H. Cormen, Ronald L. Rivest and Clifford Stein
"INTRODUCTION TO ALGORITHMS 3rd Edition”

[6]: http://www.slf4j.org/

[7]: https://code.google.com/p/log4j-swing-appender/

HOPCROFT-KARP Algorithm: John E. Hopcroft and Richard M. Karp
Reviewed by David Jackson
“An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs”

[8]: http://www.oracle.com/technetwork/java/javase/jdbc/index.html