University of Glasgow | School of Computing Science

Reader allocator

Eimantas Sapoka - 1106103

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

Level 4 Project — March 27, 2015

Abstract

In the School of Computing Science, final year students are assigned their course projects with the help of a web application - they use it to provide their project preferences, then an optimal student to project assignment is computed. A similar procedure is used when assigning readers - second markers - to projects. A tool is used to read the preferences and find a suitable allocation of markers to projects. This tool had been used the past few years and now is in need of replacement.
This project presents the Reader allocator - an application which will replace the current one by improving both the algorithm - with load balancing and the user interface - by making it easier to use. The application enhances the usability with the drag and drop functionality, improves visibility with colour coding and view filtering, introduces additional information available to the user and contains many other useful functions.

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: _____

Contents

Chapter 1

Introduction

> "The lurking suspicion that something could be simplified is the world's richest source of rewarding challenges."
>
> Edsger Dijkstra

All level three, level four and master students in University of Glasgow School of Computing Science are assigned a project to complete on their final year of study. These projects have their supervisor - a staff member from the department, who helps and guides the student throughout the project development process. Halfway through the course, each project is assigned a reader - another member of staff from the department who marks the project. Once the student completes the project, both the supervisor and the assigned reader review and mark it.

The current reader allocation procedure involves readers using a web portal to select their project preferences. This information is then fed into an application which generates the best solution in terms of those preferences. However, the algorithm the application uses to find the matching does not take into account load balancing - the notion that all readers should be assigned a similar amount of projects. This resulted in some readers being under-assigned or having an uneven workload in comparison to other markers and required manual modification to the resulting assignment to resolve the issue.

The aim of this project was to solve the load balancing issue and improve upon the usability of the existing application by building a new application from scratch. There were no specific requirements to the application apart from the addition of load balancing to the algorithm, so there was a lot of room for innovation and creativity. This also gave complete freedom over the design and implementation of the system.

The remainder of the document will discuss the following:

- **Background.** The second chapter will provide all relevant terms and background knowledge required to better understand the contents of the following chapters.
- **Requirements.** The chapter will present the set of requirements planned for this project and discuss how and why they were chosen.
- **Design.** Chapter focuses on the design of both the algorithm and the Graphical User Interface.
- **Implementation.** This chapter demonstrates the details of how the algorithm and the interface designs had been realized and discuss all problems that were encountered while doing so.

- **Evaluation.** Discussion on how the software was tested. Presents user evaluation test results and the approach taken to automated testing to ensure correctness.
- **Conclusion.** Reflection on the project development process, critical assessment of the final application and proposals for possible future work.

Chapter 2

Background

> Give me six hours to chop down a tree and I will spend the first four sharpening the axe.
>
> Abraham Lincoln

This chapter will provide all relevant knowledge and terms required to better understand the rest of the document. This includes the detail explanation of the underlying algorithm, an analysis of the existing application performing the algorithm and defining the aim of the project.

2.1 Motivation

A question could be raised, why does the School need a tool with an algorithm to assign readers to projects and why do they need to provide preferences? This process can surely be done manually - by readers selecting the projects themselves. Firstly, and most importantly, an automated tool is used to ensure that all projects are assigned and marked by a reader. This is also the main reason why readers need to provide a larger number of preferences than the number of projects they need to mark (see section 2.3.4) - so the algorithm is able to assign all projects a reader. Secondly, a manual allocation would be time-consuming and would lead to sub optimal assignments. But why find optimal assignments and build a complicated algorithm to do so? This is based on the assumption that a reader, who is assigned his preferred project will be better suited to assess and mark this project. The students also benefit from their projects being marked by a person who is interested in the field - he is more likely to fully understand the project material and provide an accurate assessment. Finally, every so often it happens that some projects are left unselected or unassigned, therefore a user interface is required to help modify these preferences and assignments to achieve the optimal results.

2.2 Similar problems

There are lots of similar allocation related problems. The most popular of which is the conference paper reviewer allocation [11] [3]. This is closely related to the reader project allocation - reviewers have their preferences over papers and an optimal allocation needs to be found. However, it differs slightly as papers need to be reviewed multiple times and assigned multiple reviewers, whereas a project needs to be assigned only one reader. In some instances, the reviewers do not rank the projects in a strict order of preference, but provide categorised

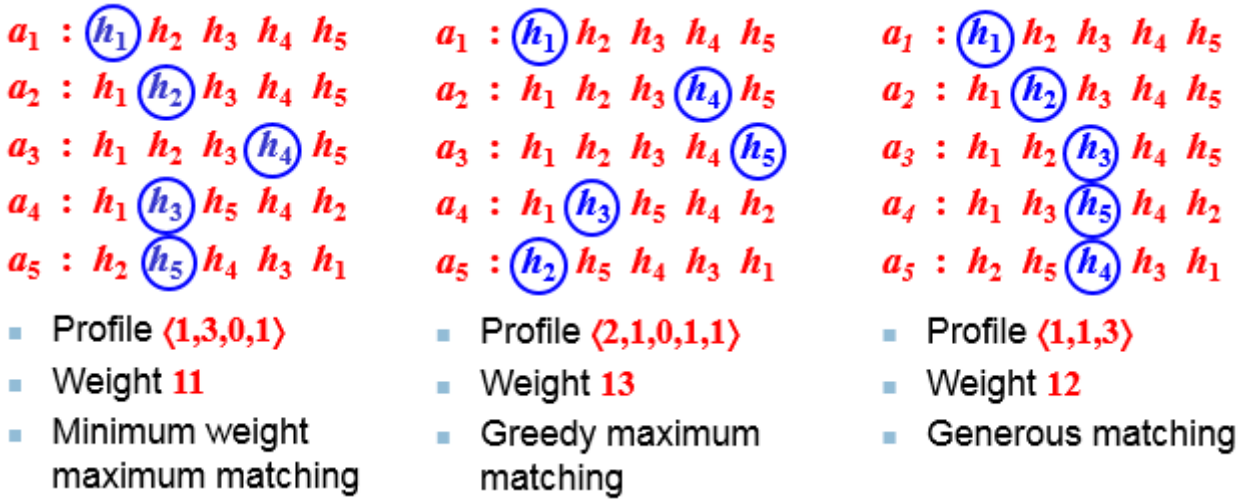


Figure 2.1: Assignment comparison of different matchings

preferences of which projects they would like to review, which ones they would not and ones they cannot - due to a conflict of interest.

2.3 The House Allocation Problem

The reader allocation problem can be modelled by a specific variant of a more general allocation problem - House Allocation (HA) problem. It is a problem concerned with allocating a set H of indivisible goods among a set A of applicants, where each applicant may have ordinal preferences over a subset of H [5, Section 1.5.1]. Formally, the problem contains a set of applicants, or in our case - readers, and a set of houses - projects. Each applicant has a preference list of houses ranked in strict order of preference. Houses do not have preference over applicants.

2.3.1 House Allocation Problem Matchings

Given an instance of a HA, it is possible to create a bipartite graph representing all acceptable applicant-house pairs. A subset of these pairs where an applicant is assigned one or less of his preferences is called a matching. This graph can be used to find various matchings depending on the optimality criteria. For example, the simplest matching is probably the naive greedy - it simply assigns applicants their first possible preference in their list with respect to some given ordering of the applicants. The greedy maximum matching finds a maximum cardinality matching and attempts to maximize the number of first preferences assigned, then maximize the number of second preferences, and so on (Seen in Figure 2.1- middle). A generous maximum matching finds the maximum cardinality matching and minimizes the number of R_{th} choices then $R_{th} - 1$ choices, and so on, where R is the size of the largest preference size (Figure 2.1- right). A combination of these two matchings is the greedy generous maximum matching, in which the generous maximum matching is performed first and once it is finished, removes all the preferences that come after the lowest ranked assigned choice and performs greedy matching on the remaining preferences. A popular matching attempts to find a matching which is the most preferred by a majority of the applicants to any other matching. Finally, a minimum weight maximum cardinality matching - an instance of maximum utility matchings in which all applicants' preferences are associated with a weight and and the resulting matching has the lowest possible total weight (Figure 2.1- left). This is the type of matching the application is performing and is implemented using Minimum cost maximum flow algorithm, presented in section 2.3.3.

2.3.2 Extensions of the House Allocation Problem

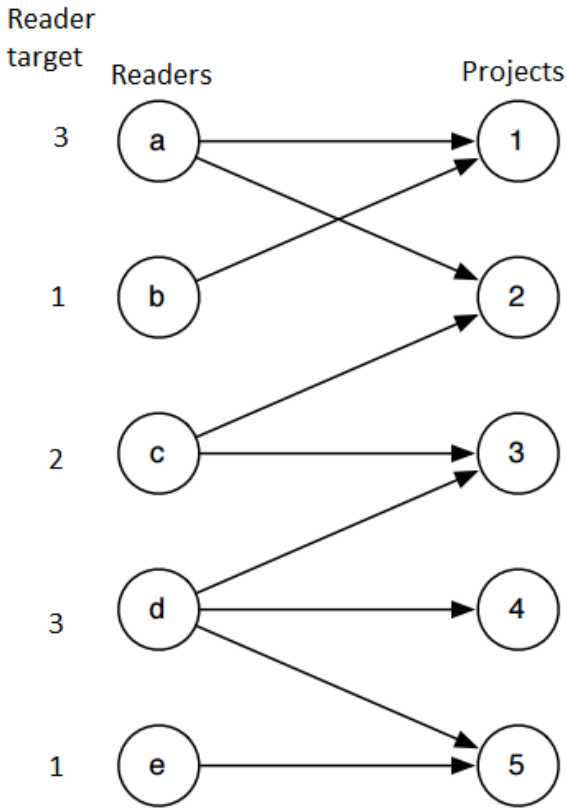
The House Allocation problem has various extensions. A variant of HA where applicants are allowed to have ties in the preferences is called House Allocation Problem with Ties (HAT). The Capacitated House Allocation (CHA) is a variant where houses can be assigned more than one applicant - a variant of which could be used to solve the reader allocation problem in the future, if school regulations change so the projects need to be marked by more than a single reader. The reader allocation problem is also a variant of the HA in the sense that each applicant can be allocated more than one house - each reader has to be assigned his reader target number (section 2.3.4) of projects. As a result, the bipartite graph no longer accurately represents the problem and regular matching algorithms no longer fit. Therefore, the problem is now presented and solved using a network by performing the Ford-Fulkerson minimum cost maximum flow algorithm.

2.3.3 Ford-Fulkerson Minimum cost maximum flow algorithm

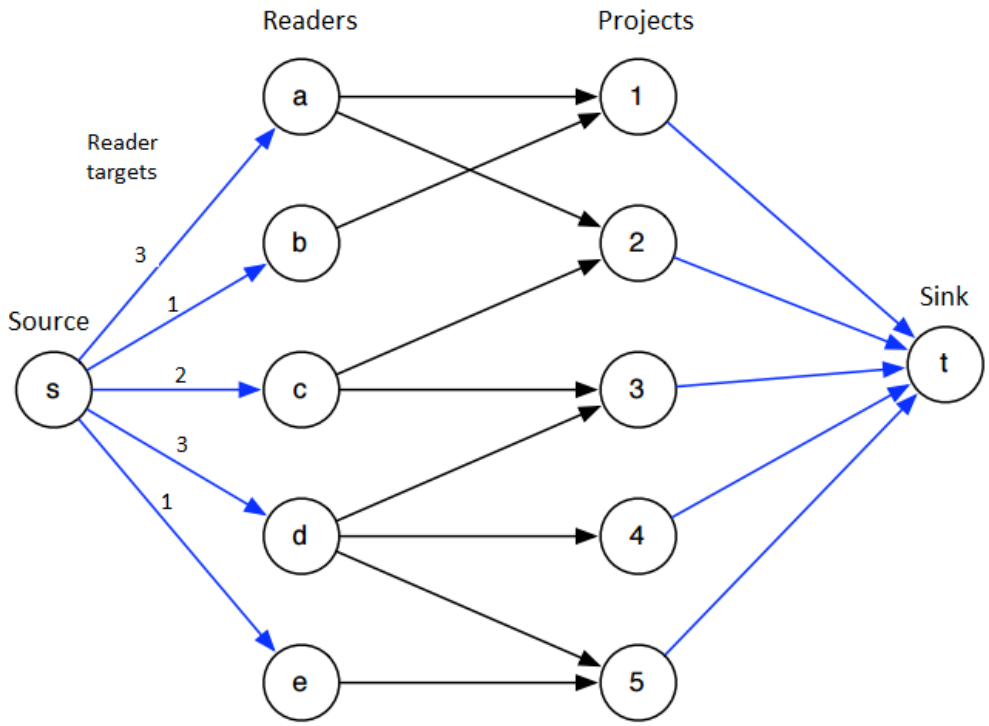
Minimum weight maximum cardinality matching where applicants can be multiply assigned is found by performing the Ford-Fulkerson min cost max flow algorithm [7, Section 8.4]. This algorithm uses a network, which can be created from the bipartite graph introduced in the house allocation problem (example in Figure 2.2a). This network is created by adding a source (a node with no incoming edges), a sink (a node with no outgoing edges), nodes representing readers and projects and edges joining all of them together (example in Figure 2.2b). Each edge has a capacity c - a non-negative real number representing the limit of flow that can pass this edge, a flow f - the current edge load $(0 \leq f \leq c)$, and a residual capacity r - the difference between edge capacity and its flow $(0 \leq r \leq c)$. Each node (apart from the source) has a distance value d $(0 \leq d \leq \infty)$ - how costly it is to reach the node from the source - computed by summing the weights of the edges taken to reach the node. Source (s) has outgoing edges to every reader with capacity equal to each reader's marking target and weight of zero. Similarly, sink (t) has incoming edges from every project with capacity of one - as each project can only be assigned once - and weight of zero. Lastly, the network contains outgoing edges from every reader to each of his preferred projects with capacities of one - a reader can select a project only once - and weights corresponding to the preference rank - the less preferred the project is, the higher the edge cost. The design of the algorithm is further discussed in the design chapter, section 4.1.

2.3.4 Reader target calculation

The reader target is specific to each reader and is computed individually beforehand. Each reader has a "delta" - a proportion of a full workload they can take. The delta is a number between zero and one, where zero might mean that the reader is on sabbatical - does need to mark any projects, a value of 0.5 means that the reader can take half the full workload - could mean that he is on probation, and a value of one means the reader needs to mark a full workload. These deltas are summed up and represent all available reader workforce (FTE). Next, the number of projects is multiplied by two (because they need to be marked twice) and divided by the sum of the deltas. The resulting value is the marking target equivalent to one unit of workload. Lastly, each reader is calculated their individual marking targets by taking the computed marking target, multiplying it by reader's delta, subtracting the number of projects he is already supervising and taking the result's ceiling. For example, if the total number of projects is 90 and the total reader workforce is 20, then we can compute a marking target per unit of workforce by dividing 2*90 by 20, which is equal to 9. Now, given a reader who supervises three projects and has a workload delta of 0.75 we can compute his individual marking target by multiplying 9 by 0.75 which is 6.75, subtract three, because he is supervising three projects, and take the result's ceiling. In the end, this reader is assigned a reader target of four. The readers are usually expected to provide a preference list which is at least twice as long as their reader target.



(a) A bipartite graph of readers and their reader targets, projects and their preferences



(b) A network created from the bipartite graph by adding source and sink vertices, and edges from source to readers, with capacities equal to reader targets, and projects to sink.

Figure 2.2: Creating a network from a bipartite graph

2.4 Current system

The application's, the school is currently using, user interface was built by Denis Sorel, an exchange student at the academic year of 2009-2010. This was his semester long project worth 10 credits. For the business logic of the application - the allocation algorithm - he used a suite of matching algorithms created by Rob Irving in 2007.

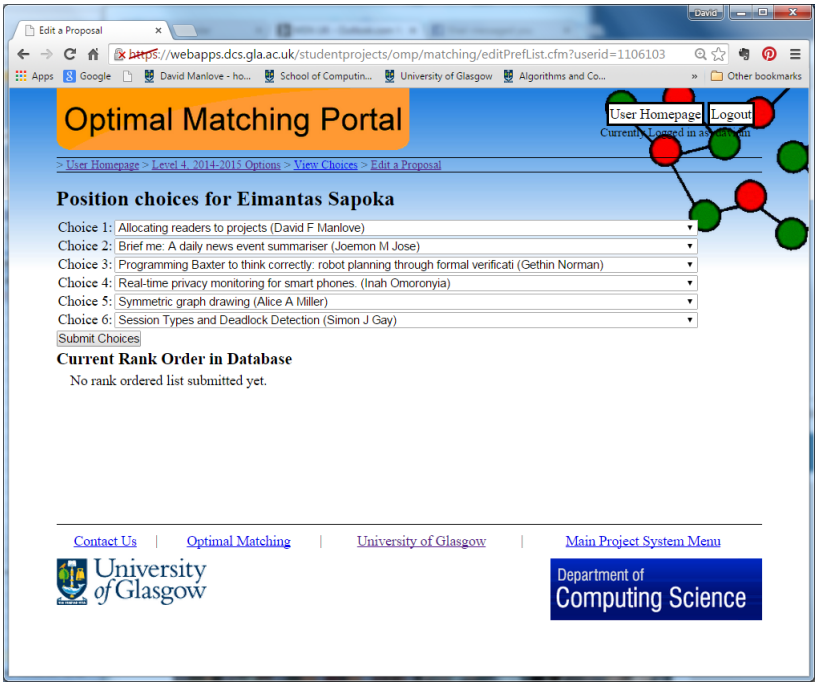


Figure 2.3: Web application students use to select project preferences. A similar page is used by readers to select their project preferences

The application first takes in reader preferences as a text file input. These preferences come from a web application readers use to select and submit their preferences (similar to the one shown in Figure 2.3). The text input consists of a number of lines containing bar separated information - reader ID, IDs of projects he is supervising, his marking target (number of projects he needs to mark) and IDs of his preferred projects in strict order of preference (Figure 2.4). This format had been chosen by the developer of the application.

```
1 |15 |1 |34 35 26 33
2 |1 16 17 18 59 66 85 61 |1 |13 20 21 22
3 |19 |8 |42 23 24 9 2 1 77 78 100 106 108 109 18 21 22 20
4 |20 21 22 59 |5 |67 68 114 34 18
5 ||0 |
```

Figure 2.4: Sample input file

Once the input file is loaded, the application main window appears (Figure 2.5). This window allows to review and manipulate reader preferences before performing the algorithm. This is mainly used to extend the lists of readers who have not provided a sufficient - twice their reader target - amount of preferences in order to achieve the best allocation - shortlists. The table on the left side of the application is showing readers with shortlists which need adjusting. The table on the right is displaying all readers with their information from the input data in separate columns. In the very center of the application lies a box with a list of project IDs which have not been selected by anyone. Below of that is a panel which allows the user to extend a specific reader's list (selected by a combo box) or all shortlists and provides a few options of doing so - with a random subset of level three and masters projects, a random subset of unselected projects, a random subset of a set of the user's choice

or a list of the user's choice. Down the bottom the user is given the choice of the matching to perform - minimum weight maximum cardinality, greedy, greedy generous or a naive greedy algorithm (described in section 2.3.1).

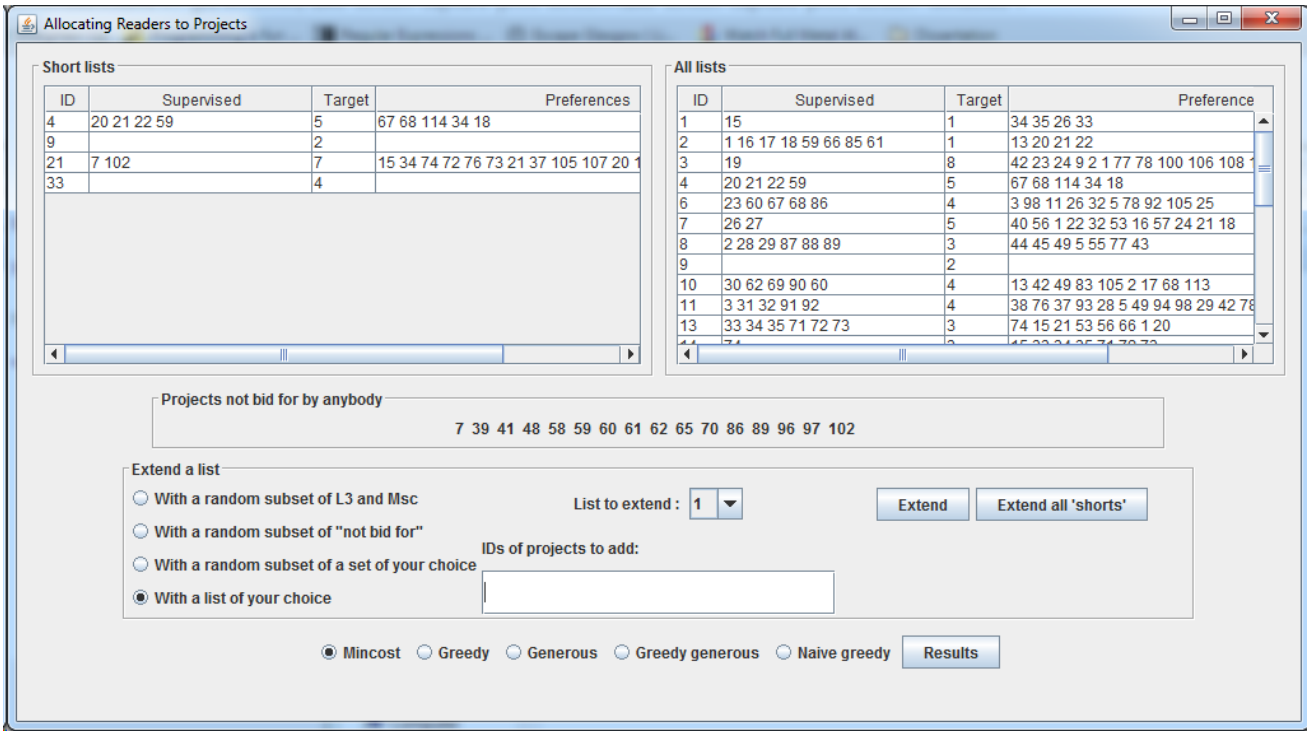


Figure 2.5: Current application assigning readers to projects - preference modification window

After the algorithm is run, a new window appears with the resulting allocation information (Figure 2.6). This window contains a huge table with reader IDs, their assigned projects, the preference ranks they were assigned and some additional information. The control buttons at the bottom of the table allow to reassign any project from any reader's list and assign it to any other reader. The project-reader view button opens up a new window showing a list of projects, the readers they were assigned to and the rank of the project in the reader's preference list.

The application has numerous user interface usability issues. Firstly, the data input the application takes is all numerical and arbitrary. Therefore it makes it confusing and difficult to make any modifications to the readers' preference lists as it does not provide any indication of reader and project names and/or identity. Secondly, the interface of the application is not user-friendly, nonintuitive and uses an old and limited java GUI framework - Swing. The tables, although scrollable, cannot be resized and, as a result, displays a fixed amount of reader information at any given time. Furthermore, they cannot be sorted by column values either, meaning the table information presentation cannot be altered. The preference modification controls are difficult - the reader must be found in the table first and his ID selected in the combo box. Next, the IDs of the projects need to be entered into the text field and once the "extend" button is clicked, the reader's preference list is extended with the project IDs from the text field. However, the text field does a poor job of validating the user's input data - the readers can be assigned non-existent project IDs, such as -10 or 9999. If this is the case, once the algorithm is run, the application crashes without any notification. Furthermore, the application does not allow to remove projects from the readers' preference lists, nor to reorder these lists for that matter, therefore the preference lists cannot modified in any other way besides extending them. This can cause an issue if a mistake is made while extending lists - the user has to relaunch the application, load the input data again and redo his modifications a second time to fix the error. Another issue is that the main window provides little feedback once an action is made. Even though the unselected project list is dynamic and changes value if a project is manually added into a reader's preference list, the application does not provide any feedback to the user if the project cannot be added to the reader. For example, if the reader already contains the project preference or is already supervising a project,

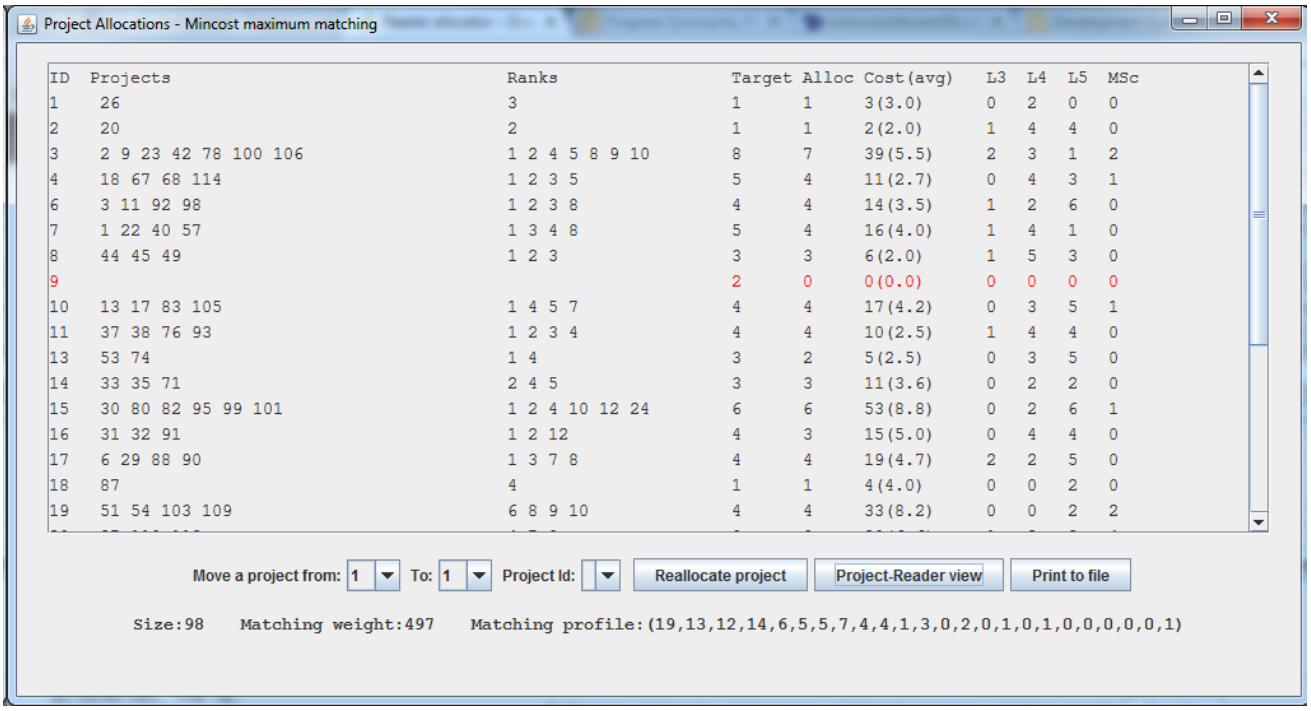


Figure 2.6: Current application assigning readers to projects - resulting assignment window

he cannot have his list extended with the project. Sensibly, the application does not allow for these actions to happen, but it does not report this information back to the user - it ignores the action entirely.

The results view is not much more functional either. Although it does present quite a lot of information on the resulting allocation, it is all presented in the table at once, overloading the user with information. However, essential information - such as projects which had not been assigned - is not shown and the controls to assign these projects are missing. This results in the user being unaware of any projects which had not been assigned by the algorithm for any reason - because of not being selected or due to the algorithm not being able to assign it to any reader. This information is also not exported to the allocation output file. Even if the user was informed about unassigned projects, the application lacks controls to assign these projects to readers. Furthermore, the remaining controls - whose purpose is to reassign projects from one reader to the other - are used similarly to controls presented in the main window. There are three combo boxes - one for selecting a reader to take an assigned project from, the second for selecting a reader to assign the project to and the last to select a project from the reader's assigned project list which will be moved. Although functional, the controls are cumbersome to operate.

Lastly, the matching algorithm the application uses to perform assignment does not implement load balancing - the notion that, if possible, all readers should be assigned a similar number of projects (see section 4.1.3). Therefore some resulting allocations are unbalanced and require manual reallocation.

However, the application does have positive features that should be retained in the new application. The project-readers view, listing all projects, IDs of readers they were assigned to and their rank in those readers' preference lists is a useful form of information. Matching profile information, network size and flow is also useful information and should be kept.

2.5 Project Aims

Because of the shortcomings of the current application it is in need of replacement. This project will be taking on the challenge of replacing this legacy system with a new one and resolving the key issues stated above. This will involve implementing the min cost max flow algorithm, extending it with load balancing and redesigning the user interface to fix the usability issues, make it more user-friendly and easier to use.

Chapter 3

Requirements

This chapter will present the requirements set up for the new reader allocation application, how they had been formed and justify their importance.

3.1 Requirement gathering

The main initial requirements were to improve the algorithm by introducing load balancing, produce correct and valid allocations and improve the usability. The first two requirements were straightforward, whereas the requirement to improve the usability of the application was left vague and required some thought and consideration to define more . Upon some discussion, we decided that in order to improve the usability, firstly I had to retain most of the key functionality from the current application. This was assigned a must-have importance, as the new application must not lack any of the useful functions of the previous one. Next, I thought on how to improve or extend this functionality to make it more convenient and easier to use. Requirements, such as exporting in different file types and allowing the user to rearrange preferences/allocations with drag and drop were created and ordered by importance. Lastly, we identified a few requirements which could be very significant, such as automatic retrieval of preferences from the web service and extending the algorithm further with fairness, but agreed would most likely not be implemented due to the limited scope and duration of the project. These were included under ”would like to have” section and would be great additions to the project in future work.

3.2 Functional requirements

The requirements are categorised using the MoSCoW principle.

3.2.1 Must Have features

- **File input.** The user must be able to select and input a data file with reader, project and their preferences information.
- **File output.** The application must be able to export the resulting allocations in to a text file.
- **Input data altering.** The user must be able to clearly see and alter the input data within the application.

- **Visual shortlists/unselected projects feedback.** The application should highlight projects which have not been selected by any reader and readers whose preference lists are too short.
- **Result altering.** The user must be able to preview and alter the results/allocations.
- **Optimal matching.** The application must find a minimum cost maximum flow matching.
- **Load balancing.** The application must be able to load balance the matching.

3.2.2 Should have features

- **Matching information.** The application should display useful profile information to the user, the number of readers assigned their first, second, third preference, total cost and flow of the underlying graph, etc.
- **Exporting as Excel.** The system would allow the user to export the results in text and excel formats.
- **Drag and drop Interface.** To make the preference and assignment data altering easier, the application should support drag and drop.

3.2.3 Could have features

- **Data modification.** The application could allow modification of input data, such as reader and project names, project supervisors, etc.
- **Graphs, visual information.** The system might have graphs or other forms of visual output to reflect and visualise the resulting allocation and/or readers' preferences data.

3.2.4 Would like to have features

These features are most likely not within the scope of the project and would only be possible within certain circumstances.

- **Automatic preferences retrieval from the web application.** The system would be able to retrieve projects, readers and their preferences from the web application.
- **Constraint programming implementation comparison.** Implement the task in the constraint programming approach and compare the two.
- **Algorithm fairness.** Using constraint programming, extend the algorithm to take fairness into account.

3.3 Non-Functional requirements

- **Usage manual.** The system would have a usage manual to help with troubleshooting and usage.
- **Change of input data format.** The application should take in input data with reader and project names and include this information in the presentation.
- **GUI.** For ease of use, the system must have an intuitive, user friendly and functional GUI. It can be either a web application or a standalone desktop application.

- **The system should be responsive and reliable.** The system should not load for an extensive amount of time and should not crash/fail unexpectedly.
- **The system should be easy to use** The system should be intuitive, not be too complex.
- **The process should be fast.** The algorithm assigning readers to projects should not take more than 20 seconds to run.
- **Feedback.** The application should provide useful feedback messages and essential information to the user with response to any actions or errors.

Chapter 4

Design

”Simplicity is prerequisite for reliability.”

Edsger Dijkstra

This chapter will describe the design of both the algorithm and the user interface, how they were planned and envisioned before being implemented.

4.1 Algorithm

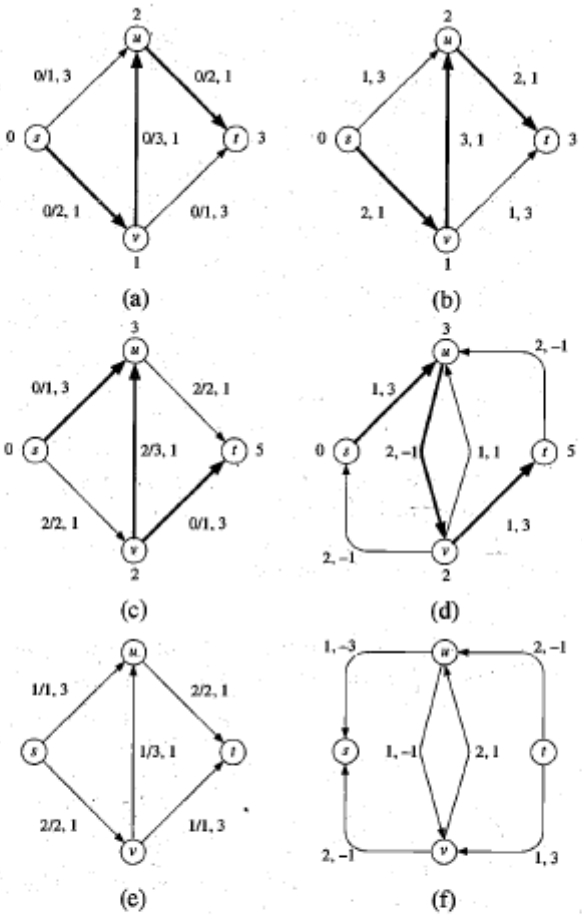
The algorithm used to match readers to projects is the Minimum cost maximum flow [7, Section 8.4]. It’s design is best explained with the help of its pseudo code seen in Figure 4.1 and by examining an example, for which a small network will be used. The example network consists of four nodes - a source (s), a sink (t) and two nodes in between (u, v) (seen in Figure 4.2). These nodes are connected with edges - two outgoing edges from the source to nodes u and v with capacities of one and two and weights three and one respectively. Another edge connecting node v to u with a capacity of three and weight of one. Lastly, there are two incoming edges to the sink from nodes u and v, one has a capacity of one and a weight of three and the other has capacity of two and weight of one.

4.1.1 Residual network

The MinCostMaxFlow algorithm uses residual networks to find augmenting paths. Given a network N, a residual network $R(N)$ is a directed network, which has all the same vertices as N, but for each edge (u, v) of N, if it has a residual capacity ($f(u, v) < c(u, v)$), make a forward edge with capacity equal to the residual capacity of e ($c_f(u, v) = c(u, v) - f(u, v)$). If edge e has flow larger than zero ($f(u, v) > 0$), make a backwards edge (v, u) with capacity $c_f = f(u, v)$. Only edges with non-zero capacity are included in residual network. The algorithm starts the search for an augmenting path by creating a residual network from the current network. Since there is no flow in the network, the residual network looks identical to the original network (Figure 4.2 b)).

```
Algorithm MinCostFlow(N):
  Input: Weighted flow network N = (G, c, w, s, t)
  Output: A maximum flow with minimum cost f for N
  for each edge e ∈ N do
    f(e) ← 0
  for each vertex v ∈ N do
    d(v) ← 0
  stop ← false
  repeat
    compute the weighted residual network R_f
    for each edge (u, v) ∈ R_f do
      w'(u, v) ← w(u, v) + d(u) − d(v)
    run Dijkstra's algorithm on R_f using the weights w'
    for each vertex v ∈ N do
      d(v) ← distance of v from s in R_f
    if d(t) < +∞ then
      {π is an augmenting path with respect to f}
      {Compute the residual capacity Δ_f(π) of π }
      Δ ← +∞
      for each edge e ∈ π do
        if Δ_f(e) < Δ then
          Δ ← Δ_f(e)
      { Push Δ = Δ_f(π) units of flow along path π }
      for each edge e ∈ π do
        if e is a forward edge then
          f(e) ← f(e) + Δ
        else
          f(e) ← f(e) − Δ          {e is a backward edge}
    else
      stop ← true          {f is a maximum flow of minimum cost}
  until stop
```

Figure 4.1: Pseudo code of the minimum cost maximum flow algorithm [7, p.404]



Example of computation of a minimum-cost flow by successive shortest path augmentations. At each step, we show the network on the left and the residual network on the right. Vertices are labeled with their distance from the source. In the network, each edge e is labeled with $f(e)/c(e), w(e)$. In the residual network, each edge is labeled with its residual capacity and cost (edges with zero residual capacity are omitted). Augmenting paths are drawn with thick lines. A minimum-cost flow is computed with two augmentations. In the first augmentation, two units of flow are pushed along path (s, v, u, t). In the second augmentation, one unit of flow is pushed along path (s, u, v, t).

Figure 4.2: Example network and the algorithm's execution on it [7, p.401]

4.1.2 Algorithm execution

Firstly, the algorithm takes a directed weighted network as input. Our directed graph is described above and illustrated in Figure 4.2. All edge flows are set to zero and all node distances from source are set to infinity before the algorithm starts. Then the algorithm creates a residual network from the original network as described in the above section (4.1.1).

Once this residual network is created, it's edge weights are modified with respect to their source and destination nodes' distances from the network's source in the previous residual graph, if one exists. The weights are modified by adding to them the difference of distances from source between edge's (u, v) outgoing (u) and destination (v) nodes. For example, the residual network seen in Figure 4.3 b) contains edges which have negative cost ($c(u, v) = -1$) and would not work with Dijkstra algorithm used to find augmenting paths. Therefore all edges have their weights modified with respect to their nodes' distances from source. The new edge weight for the edge (u, v) is computed by distance from source of node u and subtracting distance from source of the node

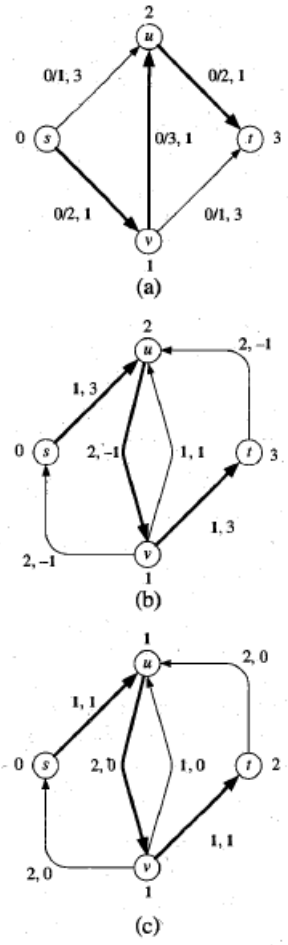


Figure 4.3: The process of adjusting residual network edge weights. Original state of the network in a), its residual network after a flow of two is augmented b) through edges $(s, v), (v, u), (u, t)$ and the same residual network after edge weights are modified c)

v. The resulting new edge weight is $w(u, v)' = -1 + 2 - 1 = 0$. Other edges have their weights modified in the exact same way. The new weight for the edge (s, u) would be $w'(s, u) = 3 + 0 - 2 = 1$. Once this is done to all edges, no negative edge weights are left and Dijkstra's algorithm can be performed on the network.

Next, the Dijkstra shortest path algorithm is run on the residual network and all nodes have their distance from source values recomputed. Once this is completed, if an augmenting path -a path in the residual network from the source to the sink - exists a (the sink vertex distance from source is less than infinity $d_t < \infty$), it finds its residual capacity $r(a)$ - the lowest residual capacity of any edge on the path. Then the algorithm augments the real network with the augmenting path. The example network contains an augmenting path (s, v, u, t) and it's residual capacity is equal to two, because even though edge (v, u) can carry three units of flow, edges (s, v) and (u, t) can only carry a maximum of two, therefore the path's maximum capacity is also two $r(s, v, u, t) = 2$. If an edge on the path is a forward one, add $r(a)$ units of flow onto the edge, if it is a backwards one, subtract the $r(a)$ of flow instead. During the first iteration, the example network does not contain a backwards edge, therefore two units of flow are added on each of the edges on the augmenting path. During the second iteration, the augmenting path $(s, u), (u, v), (v, t)$ contains a backwards edge (u, v). The augmenting path capacity is equal to one, because edges (s, u) and (v, t) can carry a maximum of one unit of flow. Therefore edges (s, u) and (v, t) have one unit of flow added to them and because of the backwards edge (u, v), edge (v, u) has one unit of flow taken away from it.

If an augmenting path cannot be found in the residual network, it means no more flow can be added onto the

network and the algorithm is complete.

4.1.3 Load balancing

Another task I had to design was the load balancing algorithm. First, to explain load balancing I will define the difference between a reader's r target t_r and the number of projects he was assigned a_r as the marking target gap $g_r = t_r - a_r$. In order for an allocation to be load balanced, the absolute difference between any two readers' marking gaps should be lower than two $\mid g_r - g_s \mid \leq 1$ for all pairs of distinct readers r, s. In order to achieve that, every time the algorithm finds an allocation, if it is unbalanced, reduce all readers' marking targets by one, reset the network to it's original state and relaunch the algorithm. Once the allocation becomes balanced (it is bound to become balanced as once all readers' have their marking targets reduced to one, the allocation cannot be unbalanced) increase the marking targets back by one and relaunch the algorithm with the same network. Since the algorithm always makes the maximum number of allocations - finds the maximum flow through the network, increasing the reader targets by one will result in everybody being assigned their maximum amount of projects they can possibly be assigned at each iteration. Once the reader targets are back to their original states, the allocation is load balanced or load balancing cannot be achieved with the given preferences

Algorithm 1 LoadBalancingMinCostMaxFlow(N)

Require: Weighted flow Network $N = (G, c, w, s, t)$
Ensure: A load balanced maximum flow with minimum cost
  $iterations = 0$
  f = MinCostMaxFlow(N)
  while f is not load balanced **do**
    $iterations = iterations + 1$
    for each Reader r **do**
      $t'_r = t_r -'$
    end for
    f = minCostMaxFlow(N)
  end while
  while $iterations \; ! = 0$ **do**
    for each Reader r **do**
      $t'_r = t_r + 1$
    end for
    $iterations = iterations - 1$
    MinCostMaxFlow(N)
  end while

The pseudo code is seen in Algorithm 1. For example, take two readers with identical preferences $p_1 = p_2 = [1, 2, 3, 4, 5]$ and capacities of $c_1 = 4, c_2 = 4$. First thing the algorithm does is run the MinCostMaxFlow algorithm with the given network. This should result in the first reader being assigned all four of his preferences and the second reader being assigned one project. Then the algorithm checks if the allocation is load balanced. It does so by comparing each readers' marking gaps with each other. Marking gap of reader one is $g_{r_1} = 4 - 4 = 0$ and the marking gap of reader two is $g_{r_2} = 4 - 1 = 3$. Since $|g_{r_1} - g_{r_2}| = |0 - 3| = 3$ and $3 > 1$ the allocation is not load balanced. Then perform the first iteration by increasing the iteration counter by one and reducing all non-zero reader targets by one, which would make the both readers marking targets to be equal to three. Perform the MinCostMaxFlow algorithm again with the modified targets. Now reader one is assigned three projects and reader two is assigned two projects. The marking gaps are $g_r1 = 3 - 3 = 0$ and $g_r2 = 3 - 2 = 1$. Now the absolute difference between those is one, therefore the network is load balanced, stop the loop and continue with the second part of the algorithm.

Once load balancing is achieved, perform $iterations$ number of loops, where during each loop increase each

reader's marking target by one and run the MinCostMaxFlow algorithm on the network with the changed reader targets, but without deleting the previous network state. Reader one was already assigned three projects and reader two was assigned two. Increase both their marking targets by one and since there are no more projects left to assign, the network stays the same. The algorithm is finished with one reader having three projects out of his marking target of four and the other having two out of his marking target of four. As already proven above, even though neither of them got their marking target of project assigned, the network state is load balanced.

To help understand the need of incremental increase of the readers' targets back up to their original values, consider the same instance with a third reader involved. The third reader has preferences of $p_3 = [6, 7, 8, 9, 10]$ and a capacity of four. Once all reader's capacities are reduced to three, readers one and three are assigned their first three capacities and reader two is only assigned his first two. Even though there are projects which are not assigned to anybody, they cannot be assigned to reader two as he does not have them as preferences. However, the allocations are still load balanced at this point. Now the reader's capacities are increased back to four and reader four is assigned project nine again. The resulting allocation is not load balanced (the difference between reader two's and reader four's marking gaps is equal to two), because it was impossible to achieve with the current preferences.

4.2 Input data format

To improve the amount of information available to the user on the graphical interface, the input data had to be changed to include more of it. Therefore it had been modified to include reader and project names and formatted in a more readable fashion. An example of the new input data format is seen in Figure 4.4.

```
5,3
1, Project One, 2
2, Project Two, 3
3, Project Three, 2
4, Project Four, 1
5, Project five, 3
1, Reader One, 3, 2 3 5 1
2, Reader Two, 2, 2 4 5
3, Reader Three, 1, 4 3 1
```

Figure 4.4: The format of the updated input data file.

The updated data format must contain two comma separated values on its first line - number of projects p and number of readers r. The next p lines must contain three comma separated values describing a project - an ID, its name and the project supervisor's ID. The last r lines contain reader information - an ID, his name, marking target and space separated list of preferences.

4.3 User Interface

> "Weeks of programming save you hours of planning."
>
> Unknown

The Graphical User Interface (GUI) was designed using the Model-View-Controller pattern (MVC). MVC

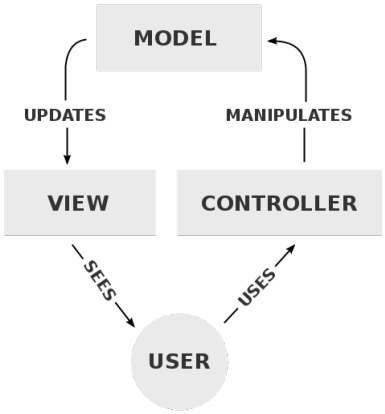


Figure 4.5: Model-View-Controller pattern

is a software architecture pattern used to implement user interfaces.The pattern separates the modeling of the domain, the presentation, and the actions based on user input into three separate classes [Beck 1992]. As the name states, the three classes are Model, View and Controller. The model is a class which holds application specific data, structures and logic. If it is changed, it notifies all its associated views - the generated representation shown to the user. If a user interacts with the application, the controller is responsible for sending commands to either the view or the model to update their state. It is most often represented by a diagram, seen in Figure 4.5. This pattern had been chosen because it is a common practice when designing applications with complex user interfaces.

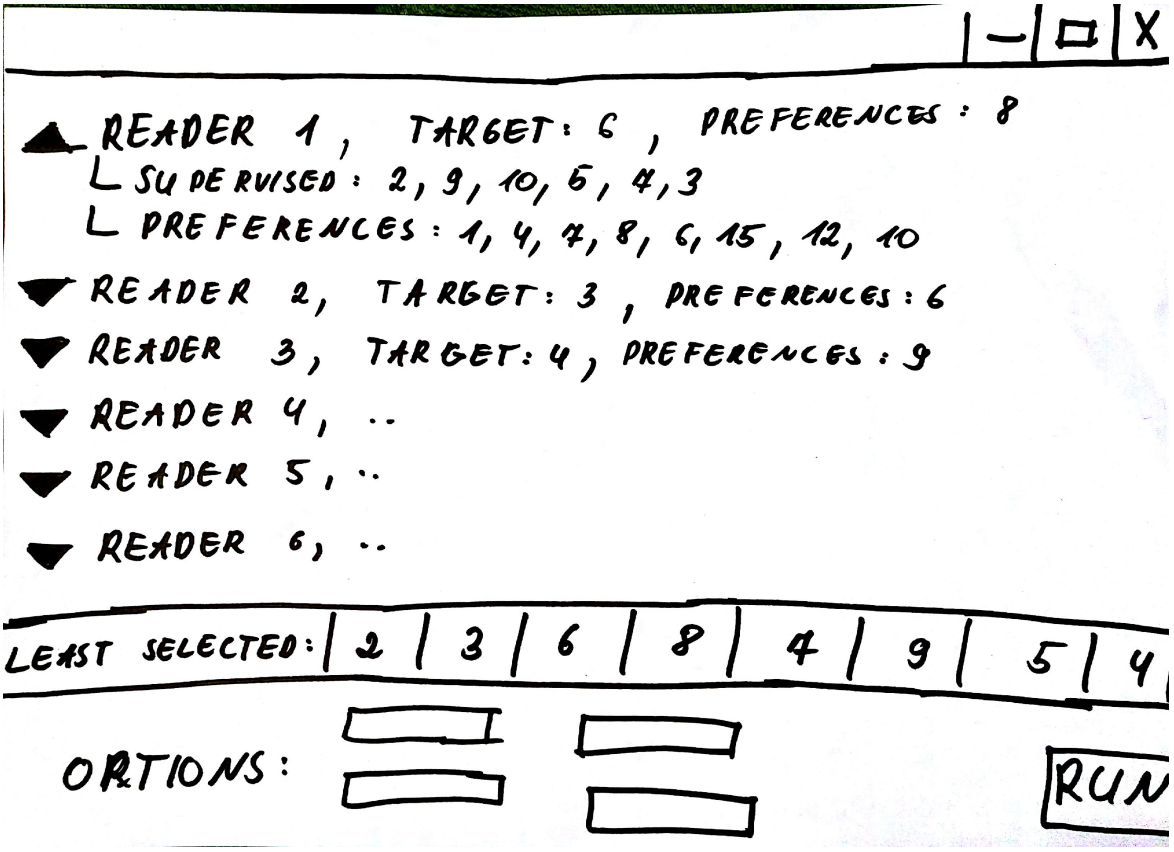
4.4 Designing the interface and prototypes

Once I had decided on the pattern to use, I had to design the application interface itself - the views. First, I identified the essential components to include in the application views. These were:

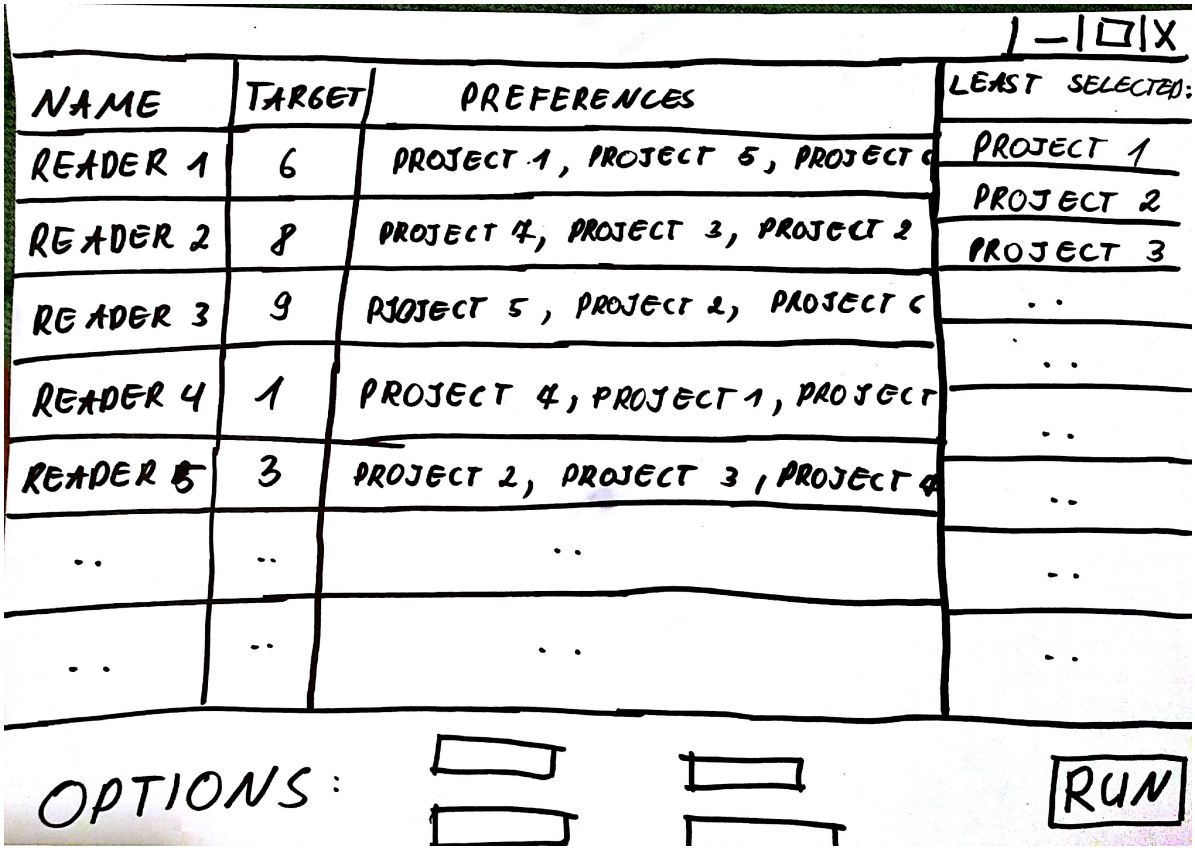
- readers' information and their preferences;
- unselected projects;
- shortlists;
- reader project assignments.

Next, I had to decide on how to visualise these components in an easy to understand and easy to view fashion. There were numerous ways on representing and laying out these components, therefore, to help visualise them and compare I created a few sketches and prototypes. The first prototype considered had the reader preferences represented by a tree structure where each node was a reader and it would contain two leaf nodes - list of supervised projects and a list of preferences as seen in Figure 4.6a. The main advantage of this design was that the presentation was compact and informative at a glance - a lot of readers could fit on the screen and their general information was presented alongside. Once this design was made into a prototype a few usability problems surfaced. The main ones were that the compact screen became packed if more than a few readers' branches were expanded and it was difficult to differentiate between the lists of each reader.

As a second attempt, I tried to visualise the readers in a table with columns for name, marking target and their preference lists. This design looked a bit crowded with information, but at the same time simple with all of the information always visible as seen in Figure 4.6b. Unfortunately it was difficult to incorporate the



(a) First sketch of an application interface design with readers represented in a tree structure.



(b) Sketch of the application with a table representation of readers and their preferences.

Figure 4.6: Initial user interface sketches

assigned projects list into the same view, therefore I decided to adopt this structure of displaying reader preferences information and create a similar, but separate view for displaying resulting allocation information. Once a prototype had been created, the design proved to be convenient with the table allowing for item ordering and column ordering and resizing helping the usability. Once the reader presentation had been decided, I attempted to visualise other essential components. Reader shortlists had an easy choice - colour coding. It highlighted shortlisted readers well enough and did not require a separate window. Adding unselected projects component was next. The first prototype had this list at the bottom of the table and it was a convenient choice, but it limited the space available for the main reader preference presentation, which was the top priority. Therefore, I decided to insert the list of unselected projects to the right side of the table vertically (4.6b). Although it limited the space available for readers' preference lists, they would rarely be large enough for it to matter, and most monitors have a widescreen aspect ratio - the application could be maximized providing for more space for these lists. Right side was preferred over the left one as it was closer to the readers' preferences lists which were the right-most column in this design. Lastly, I chose to display the reader project assignments in a separate window in a very similar table to the reader preferences one. This helped maintain consistency throughout the application and reuse the already proven convenient table design. Both of these views were accompanied by an options pane at the bottom for any controls that they may have.

4.5 Final GUI design

The final implemented design (Figure 4.7) slightly varies from the final sketch. It is because the design had been improved with respect to feedback received during the user evaluation, discussed in section 6.2.

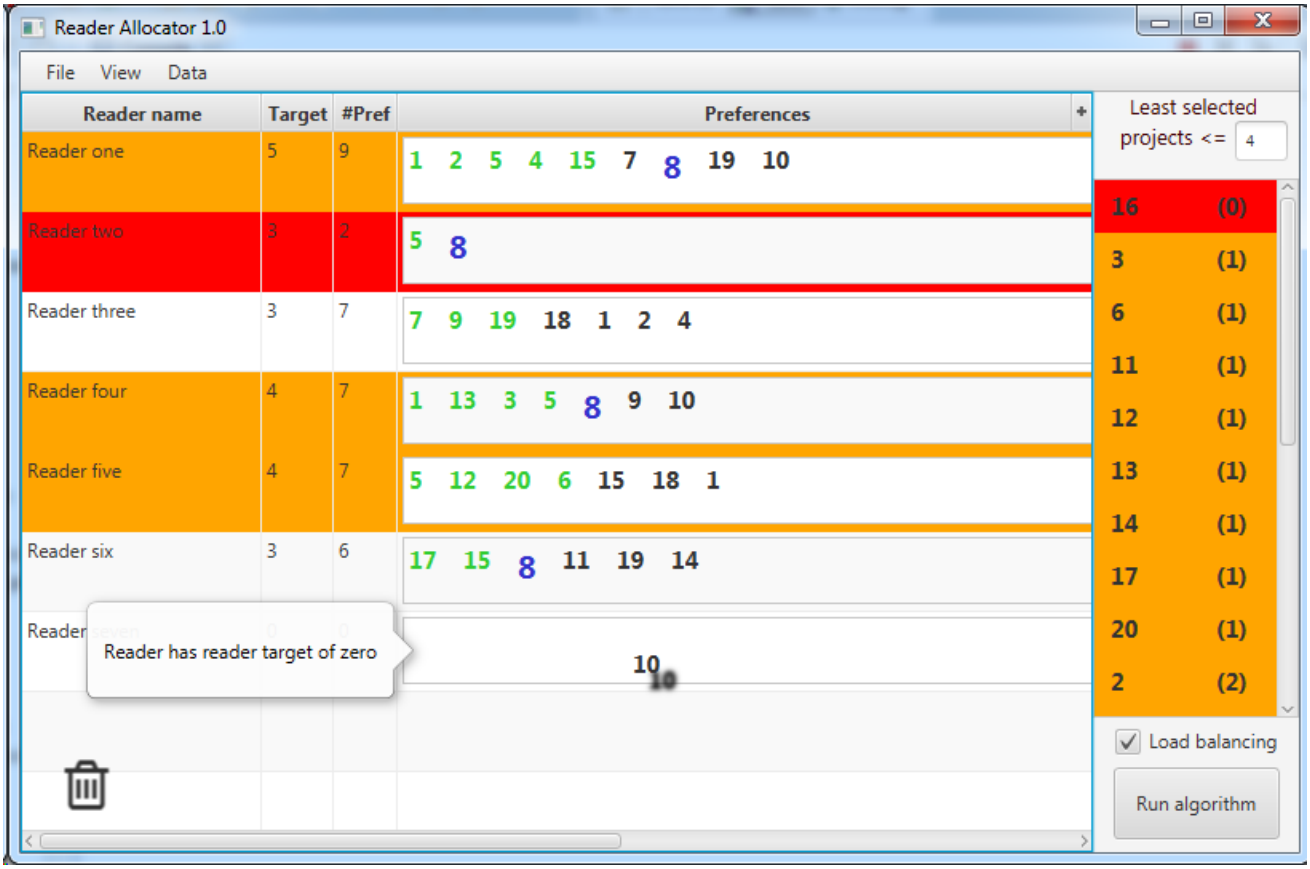


Figure 4.7: The application's preference modification window.

The final user interface is clean and simple - has only a few essential components up front - but contains

an extensive list of features. These are accessed via the menu bar or are built into the components themselves. These features are best explained individually:

4.5.1 Instance importing and generation

- **Random instance generator.** The application can create random reader allocation problem instances (further discussed in 6.1.1) from user input. The dialog window used to create a random instance is shown in Figure 4.8. The user specifies number of readers and projects and may optionally specify their upper and lower bounds for their marking targets and preference list sizes. This feature is useful for demonstration and testing purposes.
- **Reader preferences instance exporting.** The application allows to save an instance of user preferences by allowing to export them into a text file. The exported file format is identical to the one used for instance importing (Figure 4.4) and therefore can be loaded back into the application.

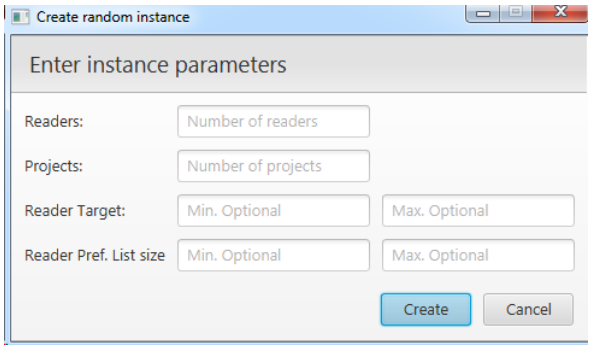
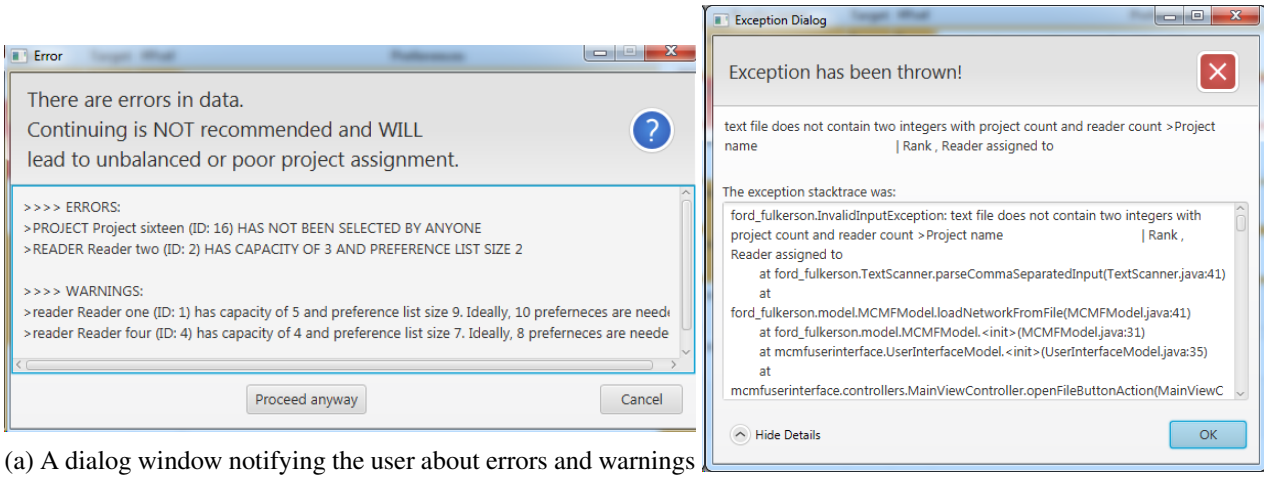


Figure 4.8: Dialog window to create a random instance.

4.5.2 General features

- **Drag and drop support.** By far, the biggest feature of the user interface must be the drag and drop. This allows for a natural and easy modification of reader preference data or the resulting allocations. By using drag and drop, the application disposes of the need for multiple complex controls that would be required to precisely operate large amounts of such complex data. It allows to reorder preferences by dragging a project and dropping it before another, to move projects between readers' lists, to move projects from the side list into the table and to remove projects from a reader's list by dragging them onto the trash bin icon. The UI is also cleaner and simpler with no visual components required to implement the feature.
- **Side list.** Both the preferences view (Figure 4.7) and the results view (Figure 4.11b) contain a list of projects on the right side of the table. In the preferences view it displays the least selected projects, ordering of least selected project first. This list also has an adjustable threshold - an upper limit which dictates up to how many times does a project need to be selected to be included in the list. This feature's purpose is to display unselected projects as well as projects which should be selected more to allow the algorithm to find the best allocation.
- **Project removal from a list.** Once the user drags an item in the preferences view, a trash bin icon appears on the bottom left of the application screen (seen in Figure 4.7). Dropping a project on top of the icon will remove the project from a reader's list. In the allocation view dropping a project on top of the unassigned project list removes the project from the reader's assigned projects list. This essential feature is missing in the predecessor application.



(a) A dialog window notifying the user about errors and warnings in the preference data.

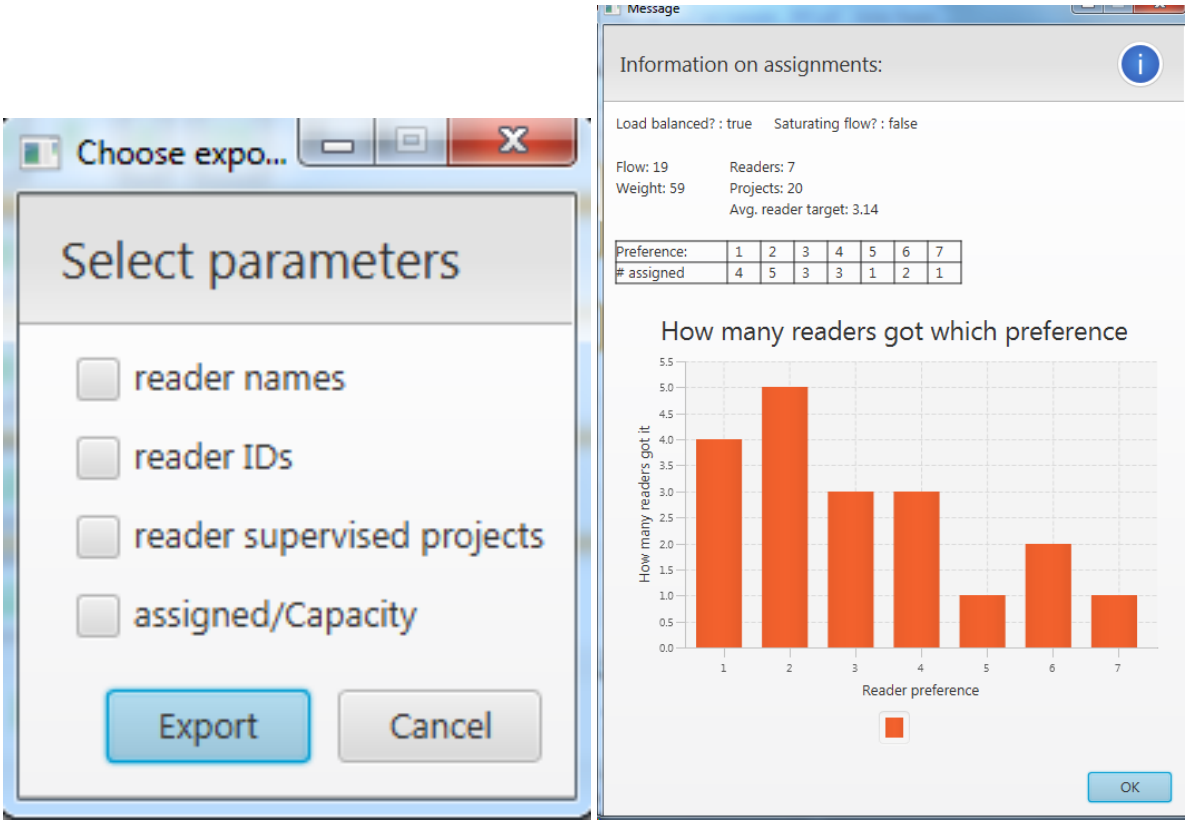
(b) An expanded exception dialog window with an exception message and stacktrace.

Figure 4.9: Exception and error dialogs

- **Colour coding.** The application uses colour coding, seen in Figures 4.7, 4.11b to highlight errors, warnings and useful information as discussed in user section 4.4
- **Information popups and feedback.** Right-clicking on any project shows a popup with the project's information, such as its name,ID and either how many times it is selected or the name of the reader it had been assigned to - depending on which view the action was performed in. The same style of popups is used to report a disallowed project drop once a project is dragged over a reader's list (Figure 4.7). Furthermore, before the algorithm is run, if any errors or warnings remain in the preference data, a dialog window is shown notifying the user about them and asking weather to proceed with the action (Figure 4.9a). Lastly, the application reports any exceptions thrown by showing an exception dialog window, containing the exception message and a stacktrace. This is most commonly observed if the input file is malformed (Figure 4.9b).
- **Reader information modification.** The table allows reader names and marking targets to be modified. Double clicking an a reader's name will show an input box allowing to enter any value. Similarly for the reader's marking target, only an integer $0 \leq N \leq 99$ is allowed.
- **View filtering.** The application allows to change the data the table is presenting by showing/hiding readers with a marking target of zero or readers with complete lists - opposite of a shortlist. The table by default supports column reordering and sorting the table by column value. Furthermore, columns can be shown/hidden by selecting which columns to display by clicking on a plus icon on the top right of the table (Figure 4.7).

4.5.3 Reader preferences window specific features

- **Project highlighting.** The table allows any project to be highlighted - make it bigger and add some weight to it. This is applied to all occurrences of this project in all readers' lists. This allows the user to quickly find the project in other readers' lists (Figure 4.7).
- **Automatic shortlist extension.** The shortlists can be automatically extended to become complete lists by clicking "Fix shortlists" button found under Data menu.



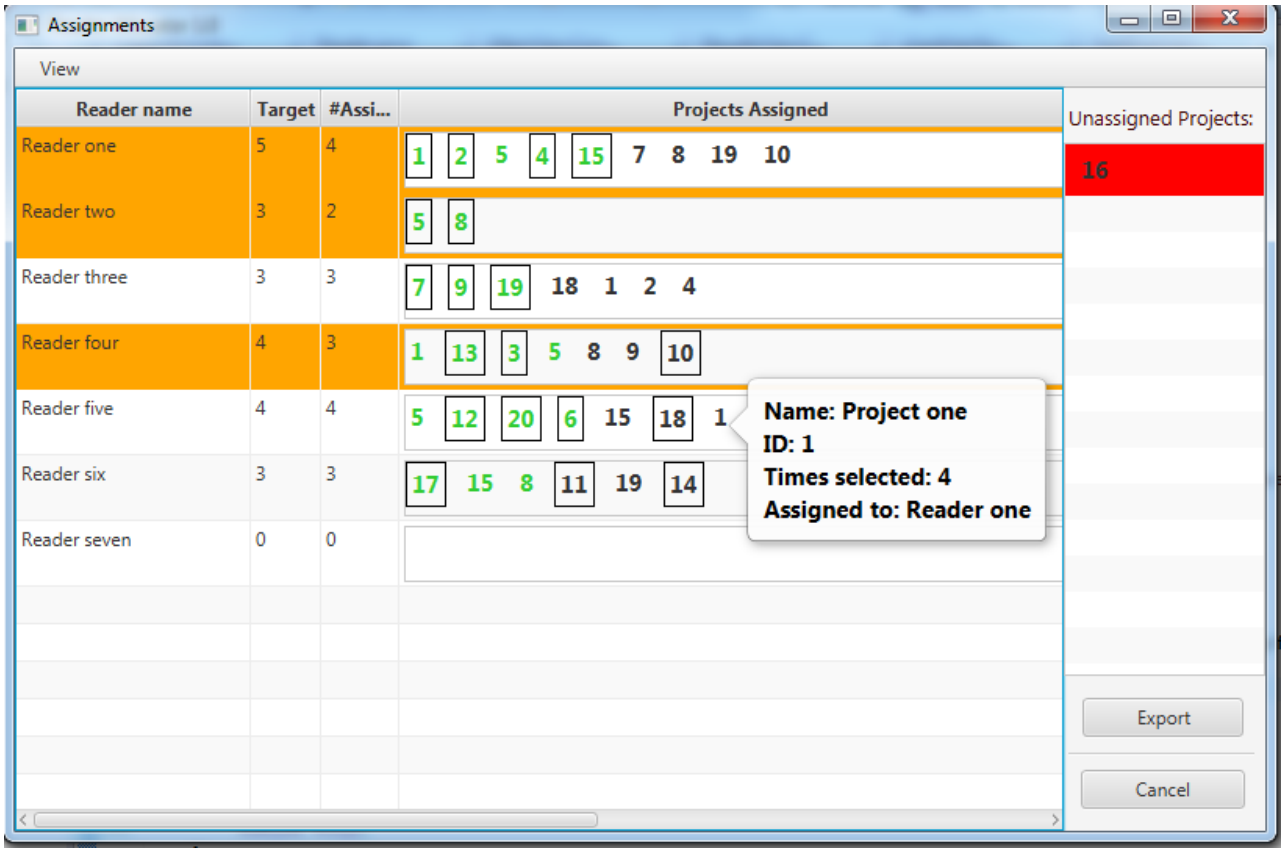
(a) A dialog window allowing the user to select what information to include in the output file.

(b) A dialog window showing matching profile and network information.

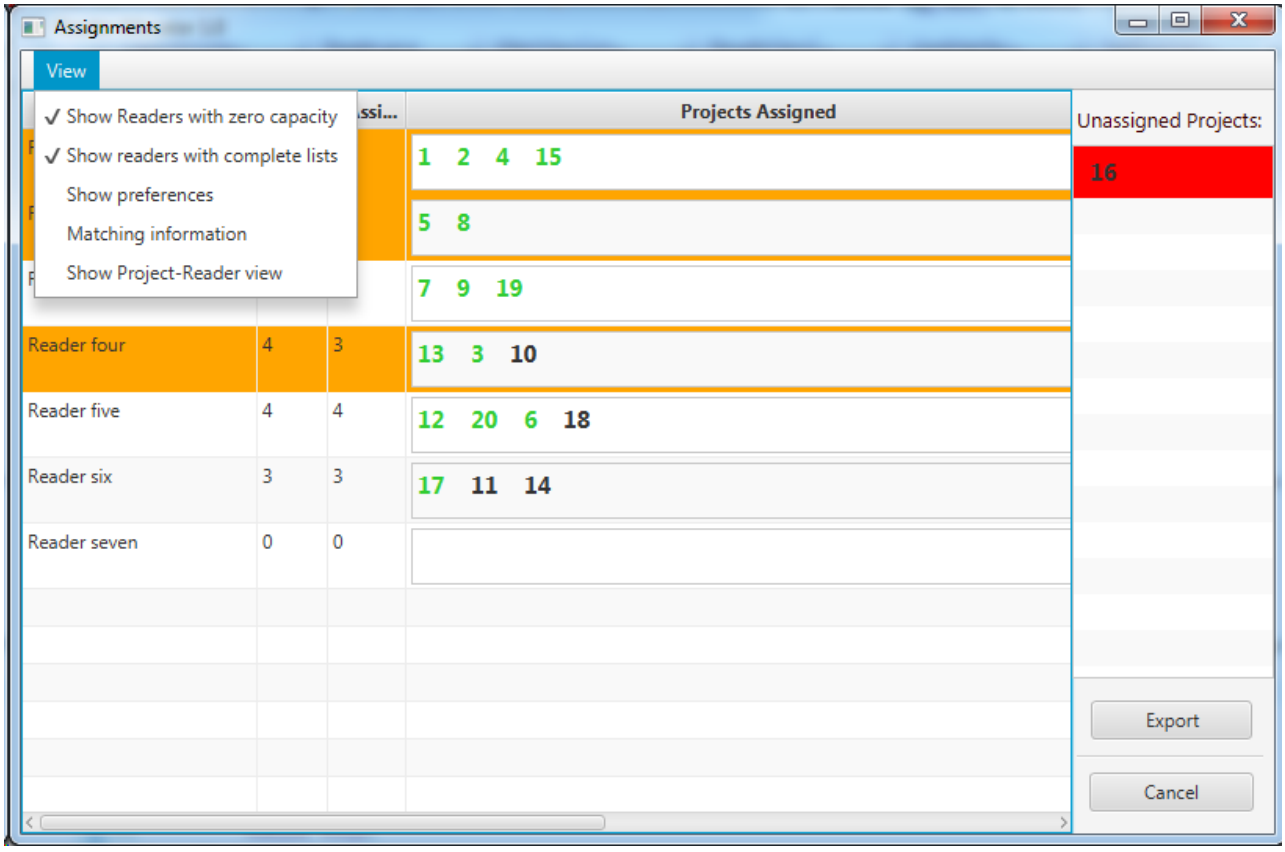
Figure 4.10: Results window dialogs

4.5.4 Reader allocations window specific features

- **Preference and assigned project comparison.** In the results view, an additional filter option is available which extends each reader's assigned projects list with his preferences and puts a border on the assigned projects to make them stand out. This allows to visually compare a reader's preference list with the projects he had been assigned. This feature is seen in Figure 4.11a.
- **Project-Reader view.** An very similar dialog box is shown to the predecessor application's one, showing a list of projects, readers they had been assigned to and which rank the project was in the reader's preference list. However, the view had been updated to show project and reader names, not their IDs. The information can be exported into a text file.
- **Matching profile and network information.** This information is available to the user by clicking "Matching information" in the View menu of the results window. This includes the matching profile - how many readers where assigned each rank preference rank - displayed as a table and a graph. This can be seen in Figure 4.10b.
- **Allocation export format parameterising.** Once the user clicks on export in the results window, a dialog window is shown allowing the user to select which information to include in the output file. Readers' allocation lists are included by default. This dialog box is seen in Figure 4.10a.



(a) Results view with preference lists filter applied. The bordered projects had been assigned to the reader.



(b) The application's resulting allocation view which opens in a separate window.

4.6 Examples of the results window

Chapter 5

Implementation

”Talk is cheap. Show me the code.”

Linus Torvalds

This chapter will explain how the application design was realised and implemented, justify the selection of application type and frameworks and present the problems faced during the process of completing the application.

5.1 Choice of application type

There were significant decisions to be made before the implementation could begin. The first and the most significant one being the type of application to build - web or standalone desktop. This decision also included choice of the programming language. If it were a desktop application, I had no doubts I would make it in Java, as I knew it best and had most experience with it. If I were to choose to build a web application, I would definitely had made it in html with javascript as it is the language of the web. Since both were acceptable choices to my supervisor, I had to decide on one by myself. To help me make a choice, I had compared the two against a set of criteria:

- **Portability.** Both application types would be portable. A desktop application coded in Java would be platform independent and would work on well on both linux and windows OS. A web application would be highly portable, not only supporting linux and windows, but, if correctly configured, mobile as well.
- **Deployability.** A web application poses difficulties of deployment - the university would have to approve the application before it could be deployed on its domain. If undeployed, it can still be used - either the server has to be run locally, which is very inconvenient to use, or have the whole implementation in javascript and preview the application on the browser. A desktop application would be a compact runnable jar file.
- **User interface development complexity.** Lately javascript based web applications have gained popularity. As a result there had been frameworks developed which greatly reduce the complexity in developing user interfaces. JQueryUI [4] is one of the most popular user interface libraries, which offers most of the functionality needed to build a usable user interface and makes for easy development. Java, on the other hand, does not offer great user interface building frameworks.

- **Algorithm development complexity.** Developing the algorithm in java would pose little to no difficulties. If the web application had a server, the algorithm could also be implemented in Java. However, if the application were to be server independant, the algorithm would have to be implemented in javascript, which would make it difficult to test. However, there are countless javascript frameworks which extend the language to follow an object programming paradigm and offer testing support, but this raises another issue discussed in the following point.
- **My coding ability.** During the computing science course we had been taught multiple programming languages, mainly Java. Therefore I was confident in my Java programming skills and that I could produce both the algorithm and a functional user interface. I did not have too much experience in building web applications or working with html and javascript for that matter, therefore I was not sure if I could successfully produce a correct algorithm or an up to par user interface. However, learning and practicing web development was a significant factor as well.

Although it would have been a great learning experience to attempt and build a web application, it would not have been such a great idea with a project of such significance. Deployability was another issue casting a shadow over the fate of the project, therefore I decided not to risk it and develop a desktop application with Java.

5.2 Algorithm

The development of the application started with the implementation of the Min cost max flow algorithm. As Java is an object oriented language, it meant creating a class for each entity involved. The main ones are:

- **The model.** Class holding a list of readers, a list of projects and a network.
- **Reader.** Holds information about a single reader - id, name, marking target, list of supervised projects, list of preferences, list of assigned projects and it's network node reference.
- **Project.** Similarly to reader, holds information about a project instance - id, name, supervisor id, number of times selected as preference and it's network node reference.
- **Network.** Contains a source and sink nodes and lists of nodes and edges.
- **Vertex.** Class representing a node in the network. Has an id, an object reference it is representing, distance from source and a list of outgoing edges.
- **Edge.** A class representing an edge within a network. Has a capacity, weight, flow, and source and destination vertices.
- **Residual network.** An extension of the Network class. Has all the same fields, but operates with residual vertices and residual edges - extension of the Edge class with a field stating weather it is forward or backwards.
- **The algorithm.** This is a static class containing methods which take the model or its network as a parameter and perform on them the Min cost max flow algorithm or it's load balanced version.

This structure can visualised with a class diagram in Figure 5.1

Once the text input file is read, a new model instance is created. For each line in the input file a new project or reader class instance is created and populated with information (preferences, supervisorId, etc.) and added into the model. Project and Reader classes create their own vertex instance and keep a reference of it. Once the

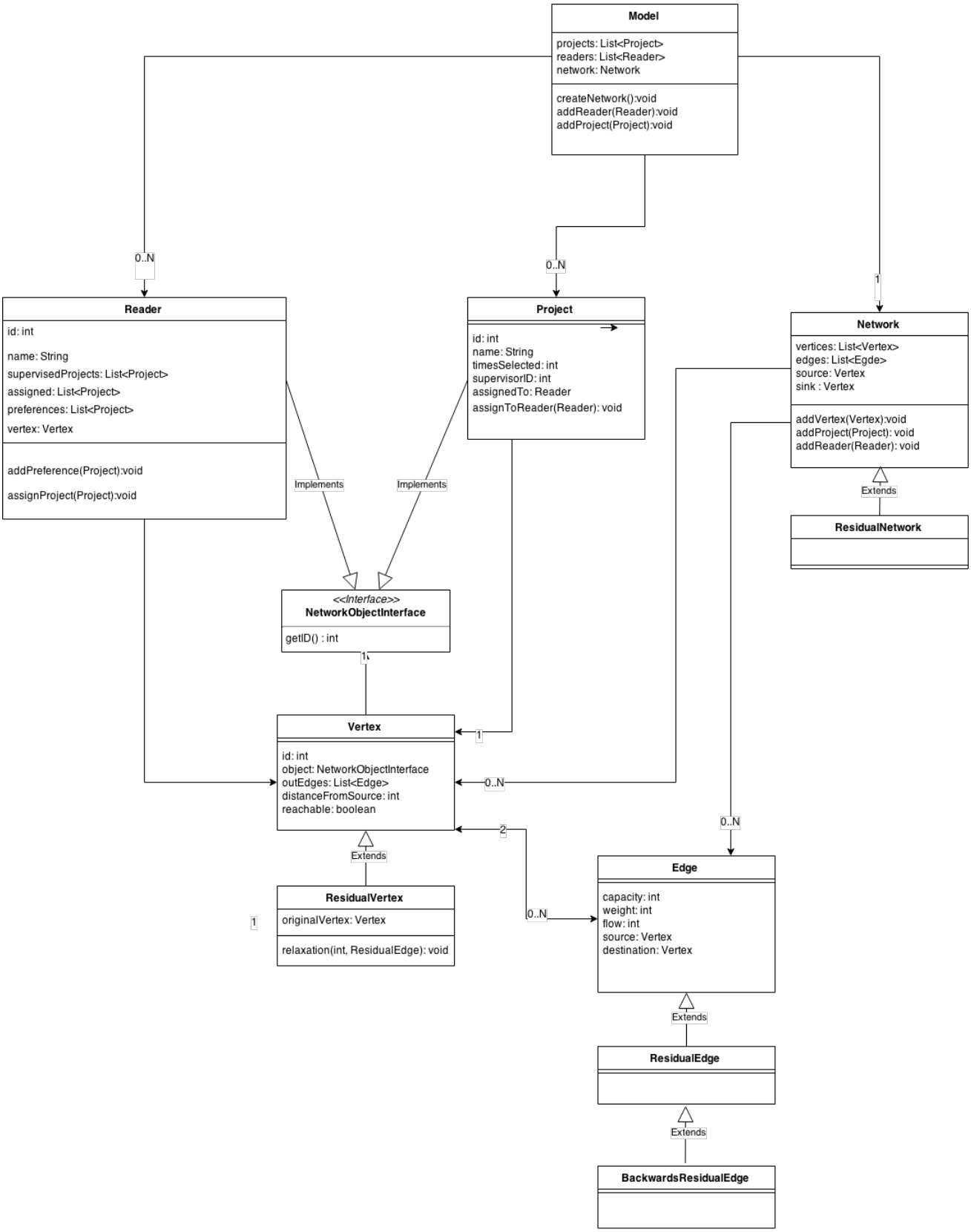


Figure 5.1: Algorithm implementation UML diagram

model calls a method called createNetwork, it creates a new Network class instance, iterates over all readers and projects, adds their vertices to the network and generates edges between them. Once this is done, the model is ready and may be passed to a static class MinCostMaxFlowAlgorithm where it's network is solved and the model may retrieve allocation information from it.

5.2.1 Minimum Cost Maximum Flow Algorithm

The algorithm is implemented in accordance to the its design in section 4.1. However, some modifications were made to due to the object oriented nature of Java language in which it was implemented. Firstly, a residual network is created. The ResidualNetwork class is an extension of the Network class and has a constructor which takes in a network and copies over all vertices from it. Next, the residual network creates residual edges connecting these vertices as explained in section 4.1.1. A residual vertex class copies all the information of the original vertex and retains a reference to it. Residual edge and Backwards residual edge classes are created by providing a source and destination vertices, a capacity and the original edge reference. Lastly, the residual edge weights are computed using the distance from source values from the previous iteration. Once the residual network is set up, the algorithm iterates over all edges and computes distances to each reachable vertex in the graph using a simple version of the Dijkstra's shortest path algorithm [2]. While doing so, each vertex keeps a reference of a residual edge the algorithm took to reach it. Once the distances are calculated, if the sink vertex is reachable (all vertices have a boolean value isReachable which is set to true on visit), an augmenting path is extracted as a list of residual edges taken to reach the sink. Since all vertices have a reference of a residual edge taken to reach them and all edges have their source and destination vertex references, it is not difficult to extract the augmenting path by backtracking edges from the sink to the source, taking the edge's source vertex as the next hop until network source is reached.

This path is then used to update the edge weights on the real network. Since all residual edges have a reference to the real network edge they were created from, it can be used to easily update them while traversing the augmenting path.

5.2.2 Load balancing

Load balancing did not require a lot of implementation. There were a couple of essential additions to the existing algorithm. One of them is a method which would compare each reader to all other readers to find weather the current allocation is load balanced. The other was an addition of an offset variable - representing reduction of reader targets - to the model class along with methods in to increase/reduce the offset by one. The implementation of min cost max flow algorithm was slightly altered to take this new offset into account when creating networks. Once these were in place, it only took a couple of loops - according to load balancing pseudo code in section 1 - to achieve load balanced allocations.

5.2.3 Problems encountered

I had faced a few issues when implementing the algorithm. In the textbook I used as a reference, the pseudo code and the algorithm description did not make it clear that the formula modifying weights takes the edge weight from of the residual graph of the previous iteration and not the original edge weight from the real network. This had prevented me from progressing for a couple of weeks. Only after the supervisor inquired about the problem with another staff member, who had previously implemented the algorithm, he provided us with the correct way of modifying edge weights.

Another problem I faced was the algorithm pseudo code suggesting that the network should be augmented with flow equal to the minimum residual capacity of all edges on the augmenting path. In certain networks this resulted in sub optimal allocations. Only after extensive research into this problem I learned that always augmenting the flow by one unit is the reliable solution.

5.3 User Interface

Once the algorithm was in place I started working on the Graphical User Interface of the application. First, I had to pick a Java framework to use to build the GUI. The main options were using Swing and JavaFX [9]. Since Swing toolkit is outdated and is visually unappealing, I decided to use JavaFX.

5.3.1 JavaFX

JavaFX is a very recent UI framework, still in development and included in the Java 8 package. Although it is still missing some advanced functionality for it to be complete, it has more than enough required for my application. As with Swing, JavaFX is a MVC based framework and offers data structures which are equipped with change listeners and automatically update the views. The main advantages of using this framework for this application are:

- **FXML.** JavaFX is an XML-based markup language that enables a declarative creation of a UI. It separates the presentation from the application logic and helps maintain a complex UI. Component injection using the tag @FXML allows for easy retrieval and use of any declared visual components and method injection eliminates the need of overriding each component's listeners to add interactivity to them.
- **NetBeans and Scene Builder.** Scene Builder [10] is a visual layout tool for creating JavaFX applications. It offers a drag-and-drop interface allowing for quick and easy creation of a UI. The application is integrated with the NetBeans IDE [8] and automatically generates a FXML document.
- **Visual presentation and CSS.** The visuals JavaFX offers are new and have a professional style. On top of that, they are easily customizable with the help of CSS. Again, this helps separate the presentation from the business logic and completely changing the appearance of specific group of components is as easy as adding a style class to them.

5.3.2 High level description

The high level description of the implementation is fairly simple, therefore it will be presented first. Later, individual essential component's implementation will be discussed.

As designed, the application contains two views - main reader preferences view and resulting allocation view. These two views have their own controllers to handle actions within them - MainViewController and ResultsViewController. Both these views had almost identical layout, functionality and components - main table, vertical list on the side - therefore an abstract superclass was extracted which contained most of the common code. Methods which had common functions, but needed to be implemented differently were made abstract and ensured that both controllers will have this done their own way. Furthermore, the abstracted ViewController superclass implements a ControllerInterface - a set of methods a controller is expected to contain to be able to communicate to the model class. The use of this interface us justified in table component implementation 5.3.3. The class diagram accurately represents the overall application design implementation 5.2. The views call their respective controller's methods, which in turn call appropriate model methods and perform changes to the model.

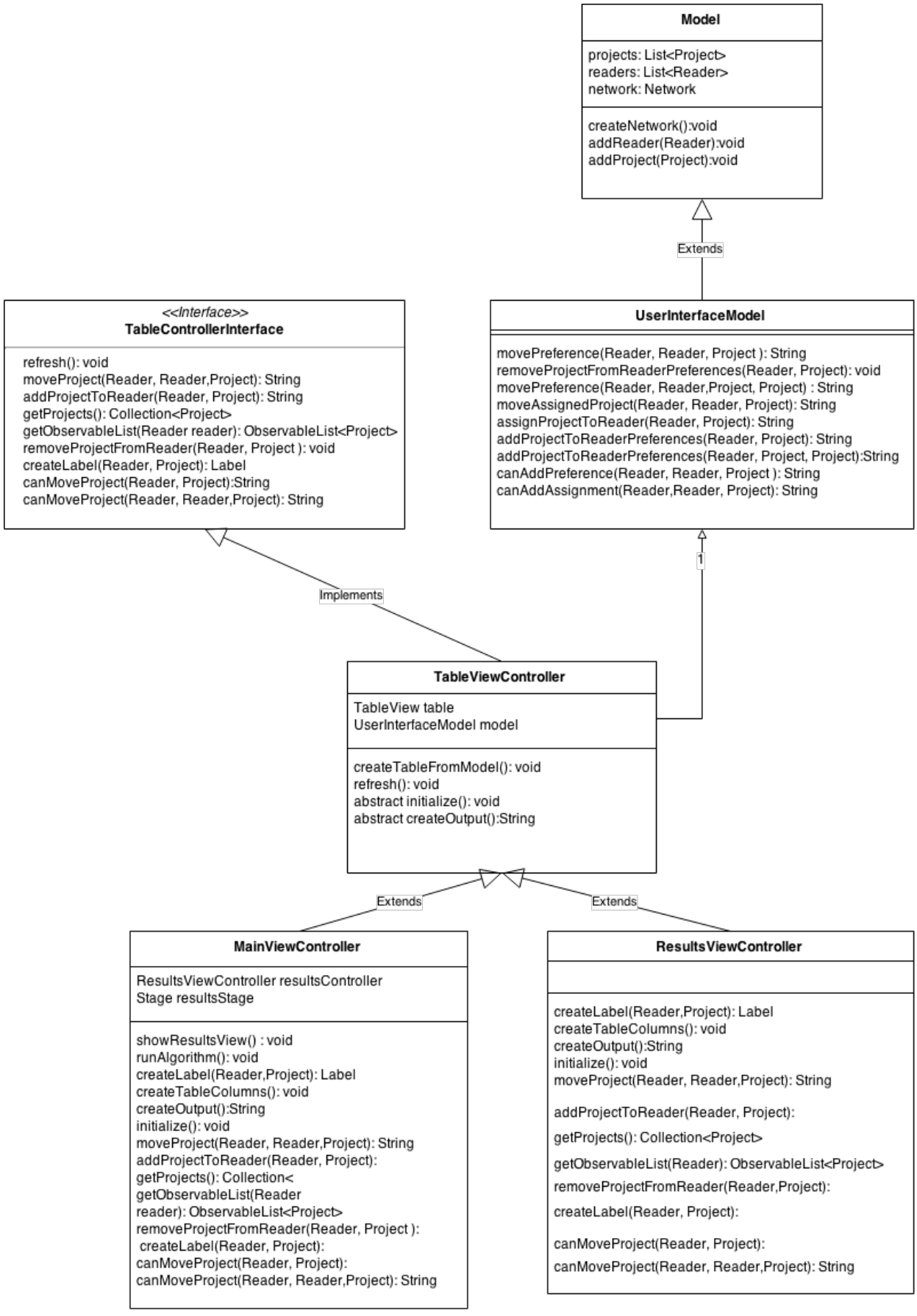


Figure 5.2: High level class diagram of the application interface

5.3.3 Component implementation

The two main components - table and side list - make up the majority of the application's complexity. Therefore it is important to explain these both in detail.

- **Table.** To represent both the reader preferences and the resulting reader allocations tables I used the TableView component from the JavaFX framework. The component itself is highly customizable and by overriding row, column or even cell rendering factories it is possible to add any other component into the table content. I had used this to my advantage by adding a list of nodes into the "Preferences" and "Assigned Projects" columns in the main view and the results view respectively. This allowed me to equip these nodes with various listeners and make them interactive. Furthermore, since the tables were almost identical, their functionality was the same, only the underlying data was different - the first table used readers' preferences, whereas the results table used their assigned projects - I was able to abstract this and reuse the table implementation. This resulted in the same table class being created in both views, only the controllers providing them with the appropriate data set.
- **Side list.** This list is a generic implementation of ListView component found in the JavaFX framework. Similarly to the table, both views had a side list performing the same functionality, but with a different data set, therefore I also abstracted the list class and reused most of its code with the data set changing depending on the controller in use.

5.3.4 Maintainability and code quality

The implementation of the user interface is maintainable and reusable. The source code is well commented and structured. Useful design patterns, such as the template pattern [1], had been used to reduce code duplication, especially when implementing the two very similar interface controllers. Both the controllers extend the same parent class (Figure 5.2), which provides a lot of common functionality and attempts to abstract a lot of it. For example, the abstract parent controller class has an initialize method, which initializes all components which are common to both the views. It also calls an abstract initialize() method, which must be implemented in the extending class and it will initialize any view specific components. Furthermore, some common components might have different properties depending on the view - e.g. the side list in the preferences view contains a a list of least selected projects, whereas the same side list in the results view contains unassigned projects. Instead of having two very similar implementations of the list, the parent controller class has an abstract method - getSideListProjects(), which is implemented individually in the view controllers and returns a different project list to use in the side list. The table columns are created similarly - the abstract controller class contains an abstract method createColumns(table) which populates the table with columns, depending on which view the table is created in. All of these are examples of the template pattern. Furthermore, the code is highly abstracted to make it reusable. For example, the column displaying the preferences list and the assigned projects list is reused. When creating the list items, the column requires the controller to provide a list that needs to be displayed. The which list is created in the column depends on which controller instance is passed into the column constructor. Furthermore, a clear separation of concerns is present in the design. The components (including the table column in the previous example) receive a TableControllerInterface instance, which only provides them with the methods they need, rather than passing a concrete class, exposing the components to a lot of methods they will never use. As a result of all of these good code practices, the application can be easily extended, modified or maintained.

5.3.5 Problems encountered

There had been some major problems encountered while implementing the user interface. The most serious of which turned out to be a feature of the framework. When implementing the table, I had placed a list component

into the preferences column rows. This list would be filled with reader preference interactive nodes. What I was unaware of is that, since creating list cells is an expensive task, the framework only creates a number of cells currently visible on the screen. Once the visible screen changes (the window is scrolled for example), instead of creating new list cells, the framework reuses no longer visible ones and updates their values to match the visible cells. This feature works very well with simple structures inside the lists - text - and increases performance significantly - a list containing a million items would not create a million list cells, only enough to fit the screen and reuse them as the list is viewed. Unfortunately, if a list contains complex data structures, such as an interactive node, the framework struggles to replace the old value of a cell with a new one quickly enough. As a result, once the table had been scrolled quickly, some reader preference list cells would not update - rendering the old value. This meant that some readers appeared to have incorrect and changing preference lists. To solve this visual bug, instead of using a list, I used horizontally laid out nodes. This achieved the same result of preferences appearing in an ordered fashion, but without the caching feature of list cells.

Another problem I faced is the lack of dialog implementation in the JavaFX framework. Dialog windows will only be included to the framework in the March 2015 java update, therefore I had to use third party libraries to be able to use dialogs.

Besides these two main issues, I came across a number of framework bugs I had to find workarounds for. Most of them are due to the framework being new and still in development. Fortunately, I had been able to overcome all of them without any significant impact to the final application.

Chapter 6

Evaluation

This chapter will discuss the testing methods used to ensure the application is producing correct results and user evaluation results on the user interface.

6.1 Unit testing

> ”Program testing can be used to show the presence of bugs, but never to show their absence!”
>
> Edsger Dijkstra

The main function of the application is to produce a reader to project assignment from the given preferences. Therefore it is a highly significant task to ensure those allocations are correct and, most importantly, optimal. The only way to ensure the application's algorithm is producing positive results is to compare them with results that are known to be correct. It can be easily achieved by creating a small instance of reader preferences, computing the optimal allocations manually and comparing them with the output of the application's algorithm. Unfortunately, with this approach it is possible to test only a small subset of possible instances and does not provide a conclusive proof that the algorithm is implemented correctly. A better approach is to use an alternative implementation of the same algorithm, which is known to produce correct allocations, and compare the results of the two. To do this, I used an existing implementation of the Min cost max flow algorithm implemented by Augustine Kwanashie. This allowed me to use more complex instances of reader allocation problem and compare the outputs from the two implementations. However, even though the instances compared were more complex, it still tested only a small selective set of instances. Therefore, to help test more instances, two random instance generators were created.

6.1.1 Random instance generators

To help me test my implemented min cost max flow algorithm, I created two problem instance generators - a random network generator one and a random reader preferences instance generator. The random network instance generator creates a network with a specified amount of nodes and generates edges between those nodes with a specified probability. The networks generated are very different from the reader preference networks that the application is designed to solve, but it provides unusual and extreme cases for the algorithm to ensure it works

correctly on all types of networks. Contrary, the reader allocation instance generator creates networks which are of similar structure to the ones the application is designed to solve. It starts with creating a specified amount of readers and a specified amount of projects. Optionally, upper and lower bounds for reader marking target and preference list size can be set and a random value between those limits is assigned, otherwise, the generator assigns them a random reasonable value instead. Next, those readers randomly select a number of projects as their preferences, mimicking a real case scenario.

Once a random network is generated, it is ported over into an instance of the alternative algorithm implementation and both the algorithms are run and compared. The two results are compared with respect to overall network cost and flow - the actual project to reader assignments may vary, but the overall result quality is the same. Lastly, the resulting assignment is asserted for basic constraints - weather no reader was assigned more than their marking target, weather all assigned projects were present in those readers' preference lists, etc. This process is repeated a number of times (5000 times for each type of instance generator) with different input values - varying number of readers, projects and preferences - to ensure an extensive coverage of various instances. The number of readers N_r and number of projects N_p in an instance are determined using formulas $N_r = i \; mod \; 100$ and $N_p = i \; mod \; 500$ where i is the current iteration $(0 \leq i \leq 5000)$. As a result, extreme case instances are also generated and tested - some with very high or low reader count to project count ratios. Out of the total 10000 tests none failed, providing high confidence in the correctness of the results.

6.1.2 User interface

Although it is difficult to test the correct performance of the user interface, I had implemented over 40 tests to ensure that the methods used by UI controllers to modify the model are functioning as expected. Therefore, even if a bug is found in the visual part of the interface, it should not affect the underlying model.

6.2 User evaluation

> "We all need people who will give us feedback. That's how we improve."
>
> Bill Gates

To test the user interface I conducted usability testing with a group of selected users. This was done to ensure the user interface met the requirements of being intuitive and user friendly.

6.2.1 Evaluation methodology

For the user evaluation tests a group of computing science students were selected. This reflected the application's target user base - a user with background in the field of computing science. These students were presented with the application, sample input data and a set of instructions they needed to perform using the application (Found in appendix A). All the participants were briefly introduced to the problem the application is solving - just enough for them to understand the application's purpose and how it is intended to be used. The main method used to conduct the feedback was the think aloud protocol. The applicants were encouraged to voice any of their thoughts, comments and struggles with each of their actions during the course of the evaluation and were informed that notes were taken. As the users were performing these tasks, the application was judged on their ability and time taken to perform the task and any successes or failures were noted. After the users were finished

with the tasks, a series of general feedback questions were asked regarding the intuitiveness, usability and the overall design of the application and how it could be improved.

The user evaluation was conducted over a period of three weeks and involved a total of 11 users. Due to the extended period of time it took to conduct these tests, the application evolved and the initial user feedback was taken into account (The effect of the feedback on the interface can be seen in Figure 6.1) . As a result, not all users performed the evaluation on the same user interface - five users had tested the first iteration of the interface (6.1a), two subjects tested the improved interface (6.1b) and the last four users performed the usability test on the final design of the application's interface (6.1c).

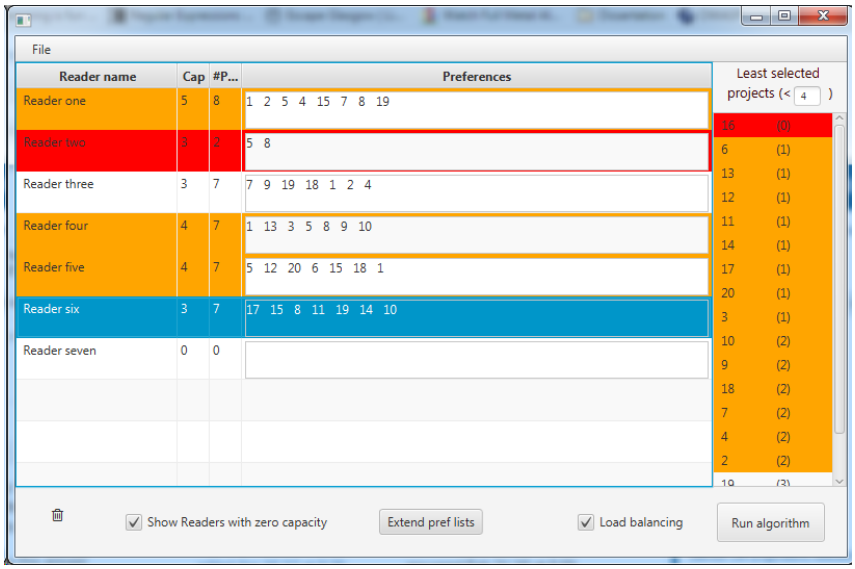
6.2.2 User evaluation process and feedback

The majority of results are of qualitative nature. These include notes taken while the user was performing tasks and their answers to the feedback questions after they had used the application. The user evaluation feedback is best presented with respect to the application version the users tested:

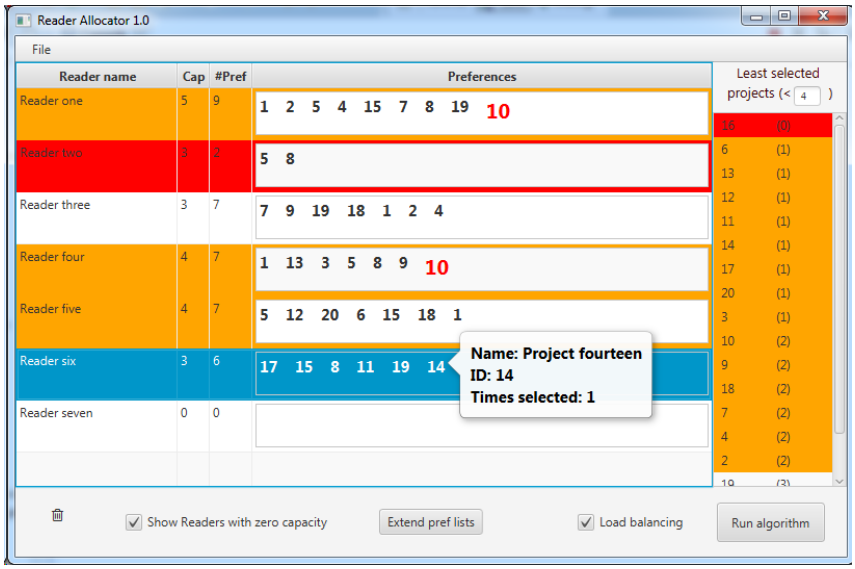
- **First version.** There were five users who tested the initial application interface. All of them had a few problems performing the tasks and commented that the interface was unclear and confusing at first. The application did not make it visually clear on how to move and remove preferences - the preference lists were seen as static and had no indication that the items were draggable. Furthermore, the trash bin feature had been underused. Some users said they had not noticed the small icon in the corner and were not aware projects could be dropped on it. Some tried clicking it with no success. The initial interface would allow any drag and drop action within the readers' preference lists, but would throw an error message if the action were invalid. This was seen as useful, but annoying by some users. Others got frustrated if their action was refused too many times. The cause of this inconvenience was the table allowing invalid actions in the first place. Furthermore, three of the five users gave a good guess on the meaning of colours, which were described as good in attracting attention and all five of the users understood the purpose of the least selected projects list on the side. A strong suggestion was made to include a tutorial on drag and drop functionality upon opening the application or make the items look more like ”objects” rather than text. OVerall, all five users struggled to use the application at the beginning, but once they figured out the main functionality, they quickly grasped the usage of drag and drop and had little trouble in completing the remainder of the tasks.

- **Second version.** There were only two users who were able to test the second version of the interface design, improved with regard to feedback received from the first usability tests. Both of them did realise that the projects are draggable much quicker than the previous test users. However, they had similar comments about the interface being not intuitive at a first glance and disliked the options pane at the bottom, containing various options, which seemed ”out of place”. This also obstructed their view of the trash bin icon, resulting in one of the users using a context menu to remove a project from the list, rather than dragging it to the trash bin, even though the user was fully aware of drag and drop functionality. Unfortunately, neither of the two users had a more specific idea on what colour coding meant besides red meaning ”something bad”, orange ”less bad” and transparent ”most likely good”.

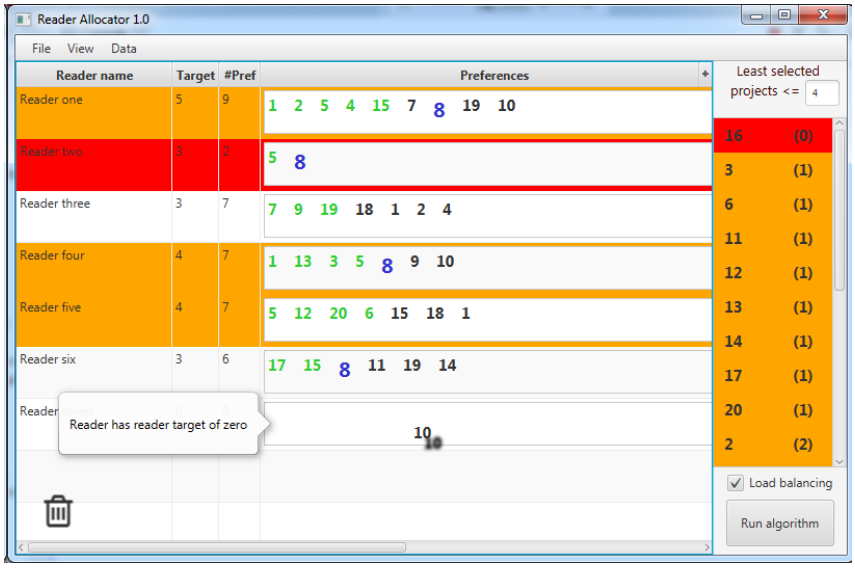
- **Final version.** The final version had quite a few visual and feedback related changes. The most noticeable is the removal of the bottom options pane and moving of those options into the menu bar. As a result, the trash bin icon was changed to appear once a project is dragged. Contrary to initial expectations, the filtering feature was less noticeable in the initial version, where it was placed at the bottom of the window, than in the final version, where it was placed under the ”View” menu. Users responded to this by reasoning that the clean window design led them to search for this option in the menus - ”the only other place it made sense” - and the ”View” menu seemed an intuitive place to find it. Another change was the feedback to the user - previous versions would allow a project to be dropped into any preference list and would show



(a) Initial version



(b) Second version



(c) Final version

Figure 6.1: Improvement of the user interface

an error dialog if it could not be done. It was changed so once a project is dragged over a list, a message would appear on the left side of the list showing an error message and the drop action would be disallowed. This final version of the interface was tested by four users and all four of the users tried to drag a project as their first attempt of reassigning it. Once asked to explain why they thought the items were draggable, the users listed a few main reasons - the larger font size and weight made the project list items stand out, the standard mouse cursor changing into a hand cursor once the mouse was dragged over a project indicated that the object could be interacted with, and finally, the lack of any other buttons or components led the users to believe that the lists could be modified directly. However, two out of four users could not identify the meaning behind colour coding. The remaining two did guess the meaning of red highlighted rows, but could not identify why some rows were orange. All four users used the appearing trash bin icon to remove a project from the list with one user mentioning that it was ”an obvious task”. Another user had noticed the trash bin appear after he started dragging a project for the first time and noted that this would most likely be a way to remove the project.

6.2.3 Qualitative results

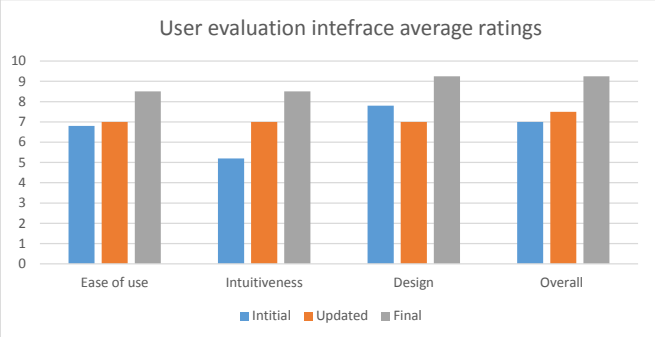
The user evaluation qualitative results are best summed up by providing the most common user opinion on application's functions.

- **File input/output.** None of the users had any problems performing tasks related to loading/saving an instance or exporting output. It was seen as an ordinary function, which need not explaining or training.
- **Drag and drop.** Initial interface users struggled to discover the feature, but once they did, they had little to no trouble using it. Final version users had no trouble discovering this function. Overall, the drag and drop feature had been praised to be an easy way to modify lists.
- **Least selected projects list.** Some users had doubts on the purpose of such a list, others recognised the potential straight away. Overall, it was seen as good to have.
- **Trash bin.** The appearing trash bin icon attracts attention straight after the first item drag, therefore users were instantly aware of it and understood how and when to use it. Overall it has been rated as a more intuitive and a bit more convenient alternative to using the list's context menu to remove items.
- **List context menu.** Only 4 out of 11 users discovered that lists have a context menu. Those who did had an easier time completing the task of adding a project to a preferences list. Overall, it is there as an alternative option to drag and drop and, although not noticed by users, is just a nice feature to have.
- **Colour coding.** The most controversial feature. The majority of the users had a very vague idea of what colours represented, especially in the results view table. However, users who had Algorithms 4 course come close to guessing the meaning of colour coding, presumably due to more profound knowledge of allocation problems. Therefore an assumption is made that the users had too little knowledge of the problem the application is solving to make use of and understand this feature.
- **Feedback.** The change to report disallowed actions while the user is still dragging the project resulted in final version readers making significantly less drag actions and completed the tasks quicker (some of which were intentionally required the user to attempt to complete disallowed actions). Overall, it was rated helpful and convenient.
- **Table view filtering.** View filtering had received mostly positive feedback. However, some users were unclear of the purpose of the ”Show preferences” feature - which allows to visually compare the readers' preference lists to their assigned projects lists. In the end, most of the users marked this feature as ”nice to have” and noted that it only adds to the functionality of the application and does not interfere with it.

- **Overall design.** Apart from the cluttered options pane at the bottom of the window in the first two versions of the application design, the overall interface quality had been ranked highly by most users. Most users thought the application was well presented and looked like a professional product. The final design was praised for having a clean and simple look and feel.
- **Random instance generator.** This feature is not directly related to the application's purpose, therefore was not evaluated.

6.2.4 Quantitative results

Although the main focus of user evaluation was to receive feedback on the usability and design of the interface, some quantitative data was collected in order to compare and visualise the results. After the users had finished the tasks, they were asked to rate the application's ease of use, intuitiveness, design and overall satisfaction with the application on a scale from one to ten, one being the lowest score and ten being the highest. The results of which can be seen at Figure 6.2. Time it took each user to complete all tasks was also recorded, but due to character biases the data was highly biased and incomparable, therefore it was omitted from the results.



	Ease of use	Intuitiveness	Design	Overall
User 1	5	3	8	6
User 2	7	5	8	7
User 3	7	4	6	6
User 4	6	6	8	7
User 5	9	8	9	9
User 6	7	7	8	8
User 7	7	7	6	7
User 8	8	8	9	9
User 9	10	9	10	10
User 10	7	8	9	9
User 11	9	9	9	9

Averages

	Ease of use	Intuitiveness	Design	Overall
Intitial	6.8	5.2	7.8	7
Updated	7	7	7	7.5
Final	8.5	8.5	9.25	9.25

Figure 6.2: Quantitative data received from the users and a graph comparing the averages of these in between interface versions.

Chapter 7

Conclusion

This final chapter will reflect on the work done, quality of the application and suggest further possible development. Also it will consider a different approach that could have been taken to implement the algorithm the application uses.

7.1 Further development

Although in a working and complete state, the application can be further extended to contain more advanced features. These could include automatic reader preference list retrieval from the web portal and extending the algorithm with fairness - both in the requirements under the ”Would like to have” section (3.2.4). The automatic reader preferences retrieval would require the web portal to provide a web API which would offer reader information and their preferences. The application could then send HTTP requests to this API and retrieve this information - in either XML or JSON and parse it into the algorithm model. This would render input files obsolete and remove a large step from the current reader allocation process - retrieving the input file from the web portal. Furthermore, this would allow the application to retrieve more information, such as each projects' categories, readers' specialisations and use these to perform a more precise and parameterised allocation.

Another feature which could drastically improve the application is to extend the current algorithm with fairness - the notion that all readers should be assigned similar preferences mean rank. The current algorithm may sometimes compute unfair allocations, for example, given two readers A and B who have identical preferences - projects one, two, three and four in this strict order. If both of these readers have a capacity of two, the current algorithm would most likely assign one of the readers projects one and two and the other reader projects three and four. This allocation is unfair, as one of the readers was allocated his top two preferences and the other was allocated his last two. To make it fair, the algorithm would have to allocate one reader his first and last preference and the other reader his third and fourth. Both these allocations would have an equal total weight and flow, but for the individual readers' the second allocation is better. Unfortunately, the fairness algorithm has a non-polynomial time complexity [6], therefore, realistically, could only be implemented in a constraint programming approach.

7.2 Alternative implementations

The algorithm could have been implemented in a constraint programming approach. Arguably, it would have been an easier and a quicker approach than constructing the networks and solving them in java. The implementation would not be too complicated - all it takes is to set up a number of readers, projects and a few constraints.

Each reader would have a constraint defining that the reader cannot be assigned more projects than his marking target and another saying that he can be assigned a project only if it is in his preference list. The projects would have a constraint that they can only be assigned to one reader. By maximizing the total flow and minimizing the total weight the algorithm would compute the min cost max flow allocation. Additionally, minimizing the marking target gap between each reader would produce a load balanced allocation. This does have it's own pros and cons. The main pro is that it would be much easier to extend the algorithm with fairness. However, this would have taken away much of the learning experience from the project, as implementing the algorithm was a challenge and an achievement. Furthermore, the maximizing and minimizing on so many variables would result in the algorithm being run high number of times, which could have a significant impact on performance once the problem instance gets large.

7.3 Reflection

7.3.1 Personal development

A lot has been learned during the development of this project. I had the opportunity to implement a complicated algorithm from scratch, extend it with load balancing and build a functional user interface to support it's use. In the process, I developed both my back-end and front-end development skills, learned in great depth a recent MVC based user interface framework for Java - JavaFX. There were a lot of problems I had to overcome - both in the design and the implementation of the application. Lastly, I experienced a full project development cycle - from requirements gathering, to design and implementation, evaluation and release.

7.3.2 Current state of the application

There is no doubt the application improves upon it's predecessor. It meets all of the project's essential requirements - load balancing, data modification, etc. - and surpasses them with additional features, such the dynamic least selected projects list, view filtering, preference instance saving, useful feedback, preference list overlay with assigned projects and etc. These features significantly improve the usability of the application. Furthermore, as the user evaluation tests have shown, the majority of the users admired the user interface and rated it highly. The unit tests ensured that the resulting algorithm is producing correct results and, in fact, the load balanced version of the algorithm was used to find the reader allocations to fourth year computing science student projects during the year it was developed (2014-2015). However, there are still requirements left which were not implemented. A should have requirement to allow export data into excel had not been achieved. This decision was made, because the implementation complexity of the requirement greatly outweighs it's benefits. Although it would be nice to export the data into an excel spreadsheet, it would require significant effort and third party library support to achieve and the benefits are minor - a comma separated input can easily be imported into an excel spreadsheet. Therefore other features gained priority over this requirement. In the end, the final application is functional, in a complete state, maintainable, well documented and ready for further development. Therefore I consider the project to be a definite success.

Appendix A

User Evaluation Instructions Document

Before you begin

Are You familiar with reader allocation algorithms? Y/N

This is a think-aloud type of user evaluation test. It consists of a few simple tasks You will perform using an application. It is highly encouraged to express any opinions regarding the user interface and speak out everything You are thinking while performing these tasks. The test should not take longer than 5-10 minutes. The test is anonymous, but your attempt will be recorded.

The purpose of the application is to help the user with the process of assigning projects to readers. The reader preferences are provided as a text input. The application allows the user to modify these preferences, if needed, and run the algorithm, which finds the best allocation of projects to readers, based on their preferences.

The tasks

1. Import the sample data file called "sample data.txt" from the desktop. (Note: please do not delete my C: drive)
2. Describe to your best understanding what the table and window as a whole are presenting.
3. State if you think the sample data is fine to run the reader allocation algorithm. If not, why?
4. Move project 7 from "reader one" to "reader four". Place it into any position in the list. Explain what happened in terms of colour change.
5. Add project 20 to "reader three" as its first preference.
6. Remove project 20 from "reader three".
7. How many times has project 9 been selected? what is this project's name?
8. Attempt to run the algorithm. If there are any errors being reported (warnings are still fine to run with), cancel the algorithm and fix those errors. Repeat until there are no errors and then run the algorithm.
9. Describe to your best understanding what the new results window is presenting.
10. Assign all unassigned projects to any readers you seem fit.
11. Reassign project 15 to "reader four"
12. Un-assign project 13. Make it not be assigned to any reader.
13. Using the application, find out how many readers got their first choice assigned to them. Try to find out as much about the resulting assignment as possible and state your conclusions about the assignment.
14. Export the data in the format of your choice.
15. Go nuts. Attempt to break the application in any way you seem fit. Each unique uncaught exception gains one candy! (Out of ordinary behaviour, such as projects not getting reassigned or assigned wrong - bugs - count) (Note: use random instance generator for better testing)

Figure A.1: User evaluation instructions first sheet

Some feedback questions

1) Did You find the application easy to use, intuitive? How easy/difficult was it to perform the tasks?
2) Was the application design good? Would You have designed it in any other way?
3) Did the application perform well, was it slow? Do you feel the performance should be improved for better use?
4) How could the application be improved, made easier to use?
5) Any general feedback, questions?

Thank You!

Figure A.2: User evaluation instructions second sheet

Appendix B

User Manual

This chapter will document the usage and explanations of all the functionality within the new Reader Allocator application.

B.1 Loading an instance

The application takes a text file as an input. The format of the text file is described in section 4.2 and presented in figure 4.4. Only .txt file extension is allowed to be selected. If the file input is malformed and the application is unable to parse it, an exception dialog is thrown with an error describing message and a stacktrace.

Alternatively, the user may generate a random instance. This is done by clicking ”Create random instance” in the File menu. A dialog window appears with options allowing the user to configure the random instance. The mandatory fields are reader count and project count. There are additional optional fields to specify each reader's capacity and preference list size lower and higher bounds. These fields can be left blank and a default random reasonable value will be assigned to each reader. If any of these optional fields are not input correctly, e.g. a higher bound is set to be lower than the lower bound, they are ignored. If either reader count or project count field is missing a value, the instance is not generated and the dialog is closed.

B.2 Exporting

The user can choose to export three things: the reader preferences instance, the resulting allocation or the project-reader view. The reader preferences instance can be exported by selecting ”Export instance” from the ”File” menu in the main application window. To export the resulting allocation, the user should click ”Export” button in the results window and select any parameters that should be included in the exported file. The reader allocation lists are included by default. To export the project-reader view, first open this view by selecting ”Project-Reader view” from ”View” menu in the results window and then pressing export in the opened dialog box. All of these export actions open a file chooser which allows the user to select the location and the name of the exported file. Only .txt file extension is allowed.

B.3 Manipulating the preferences

The user can change the readers' preferences in whichever way he seems fit simply by dragging and dropping the projects to and from these lists. To move a preference from one reader to another, simply drag the project from one reader's list and drop it into the other's. To add a project to a reader's list, the user can drag it from a least selected projects list on the right side of the application and drop it into an appropriate place in the desired reader's list. If for any reason a project cannot be added to the reader's preferences, an error popup will appear by the list with the error message. Alternatively, the user can right click on the empty space in the reader's list and select "Add.." from the context menu. This will open a dialog box providing a combo box of all the projects the reader can have added to his preference list and clicking "Ok" will add it as his last preference. To remove a project from a reader's list, start dragging the project, a trash bin icon will appear on the bottom left of the application screen and dropping the project on top of it will remove this project from his preference list. Alternatively, this can be done by right-clicking on an empty space in the reader's list and selecting "Remove..". This will open a similar dialog window to the adding option, also containing a combo box, only now it allows to select projects which are already in the reader's list and can be removed. Clicking "Ok" will remove this project from the reader's preference list.

B.4 Manipulating the assignments

The user may manipulate readers' assigned projects list in a very similar way to their preference lists. This is done using drag and drop. To reassign a project, simply drag it from one reader's list into another's. To allocate an unassigned project, drag it from the unassigned projects list and drop it into any valid reader's list. To unassign a project, drag it from a reader's list to the unassigned project's list and drop it there. If for any reason a project cannot be assigned to the reader, an error popup will appear by the list with the error message.

B.5 Colour coding

The application uses a lot of colour coding. In a general sense, red means error, orange means warning.

More specifically, in the reader preferences table a reader highlighted in red means his preference list contains less projects than his marking target. A reader highlighted in orange means his preference list size is less than twice his marking target. In preference lists, the first marking target number of projects (the ideal reader's allocations) are highlighted in green. If a project is left-mouse-clicked, all occurrences of this project are highlighted in blue.

In the results window, the table has same colours used, bit with a bit different meanings. Red reader rows mean that this reader had been assigned less than his marking capacity minus one and orange rows mean the reader had been assigned his marking capacity minus one projects. The green projects in the assigned projects list mean the project was within the first marking target number of projects in the preferences list.

On the right least selected projects list, projects highlighted in red have not been selected by any reader and orange ones have been selected by less than four readers.

B.6 Filtering and information

There are multiple ways in which the user might filter the table views or find information. Firstly, in both the main view and the results view, the table can be filtered to show/hide readers with zero capacity and show/hide readers with complete lists.

Next, in the results window the user may click on show preferences found under ”View” menu to expand the assigned projects list with the projects the reader had as preferences. This also highlights the projects the reader had been assigned with a black border and allows to see where in the preferences list the assigned projects were located.

Furthermore, in the results window, clicking on ”show matching information” in the ”View” menu will display a dialog window with the information about the resulting allocation - such as the network size, cost, flow and the matching profile displayed as a table and a graph.

By clicking on ”show Project-Reader view” found under ”View” in the results window, another dialog window will be displayed with all project names and reader names to which they were assigned to.

Bibliography

[1] Template pattern description. http://www.oodesign.com/template-method-pattern.html.

[2] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[3] EasyChair. Easy chair conference management system. http://www.easychair.org/.

[4] JQuery. http://jqueryui.com/.

[5] D.F. Manlove. *Algorithmics of Matching Under Preferences*. World Scientific, 2013.

[6] Naveen Garg, Telikepalli Kavitha, Amit Kumar, Kurt Mehlhorn, Julin Mestre. Assigning papers to referees. 2010.

[7] Roberto Tamassia Michael T. Goodrich. *Algorithm Design. Foundations, Analysis, and Internet Examples*. 2001.

[8] NetBeans. Netbeans ide - the smarter and faster way to code. https://netbeans.org/.

[9] Oracle. Javafx - the rich client platform. http://www.oracle.com/technetwork/java/javase/overview/javafx-overview-2158620.html.

[10] Oracle. Javafx scene builder - a visual layout tool for javafx applications. http://www.oracle.com/technetwork/java/javase/downloads/javafxscenebuilder-info-2157684.html.

[11] Don Conry,Yehuda Koren,Naren Ramakrishnan. Recommender systems for the conference paper assignment problem. labs.yahoo.com/files/p357.pdf.