Comparing Variants of the Stable Marriage Problem and Analysing the Number of Stable Marriages

Luke Britton

A DISSERTATION

Submitted to

The University of Liverpool

in partial fulfilment of the requirements
for the degree of

MASTER OF SCIENCE
IN
COMPUTER SCIENCE

September 21, 2017

Abstract

A classic problem in the field of combinatorics is the stable marriage problem where for a problem instance of size n we are tasked with pairing off n men and n women based on each individual's preference list over the opposite sex, in such a way that the resultant set of marriage pairs are "stable". We refer to this set of stable pairs as a stable marriage. A marriage M is said to be unstable if there exists some man m and woman w who are not married in M but both prefer each other to their current spouses in M. There always exists at least one stable marriage and we can find a stable marriage in $O(n^2)$ time using the Gale-Shapley[1962] algorithm. However this algorithm always disproportionately favours either the men or the women; simultaneously giving the best possible matches to one sex and the worst possible matches to the other. Depending on which sex is favoured, this is known as either the male or female optimal marriage.

The disproportionality in the "satisfaction" of individuals can be improved by refining our search to find the "minimum regret" stable marriage or the "egalitarian" stable marriage. We quantify the satisfaction of individuals in a marriage M in two ways; the first is to calculate the maximum regret of M, the second is to calculate the social cost of M. The regret of an individual is their preference list rank of their spouse in M. The social cost equals the sum of all regrets in M.

In this paper we determine the difference in satisfaction amongst individuals in the male optimal marriage, minimum regret marriage and egalitarian marriage by defining their average difference in maximum regret and average difference in social cost as functions of n. In addition to this we also specify a confidence interval as a function of n for the total number of stable marriages and provide insight on how one might obtain a good estimate on the upper bound for the number of stable marriages.

Contents

List of Figures

Chapter 1

Introduction

1.1 Scope

For each instance of the stable marriage problem of size n, we have n men and n women who hold preference lists over their opposite sex. A marriage is a set of n pairs of men and women. That marriage is said to be **stable** if and only if there does **not** exist some man and woman who are not currently married that prefer each other to their current spouses. In this paper when we refer to a marriage we are always referring to a stable marriage unless specified otherwise. Gale and Shapley[1962] proved that for any problem instance, a stable marriage always exists, and they provided a $O(n^2)$ algorithm for finding a stable marriage. This algorithm always outputs a marriage which disproportionately favours either the men or the women; giving one sex their best possible matches, and giving the worst possible matches to the other. Depending on which sex is favoured, this known as either the male or female optimal marriage.

We can address this disproportionality in "satisfaction" in two ways. The first is to find the marriage which minimises the maximum regret, known as the minimum regret stable marriage. The **regret** of an individual in some marriage M is equal to their preference list rank of their spouse in M. Thus clearly the **maximum regret** in M is equal to the regret of the individual with the largest regret in M. We will denote the maximum regret $\mathbf{r}(\mathbf{M})$. The second way two address this disproportionality is to instead find the marriage which minimises the social cost, known as the egalitarian stable marriage. The **social cost** of a marriage M is the sum of the regrets of all individuals in M. We will denote the social cost $\mathbf{c}(\mathbf{M})$. All stable marriages have a maximum regret and social cost which we can use to quantify the satisfaction of individuals in a marriage.

The number of stable marriages for different instances of size n varys greatly, it can be as small as 1 or exponentially large with respect to n. Knuth[1976] showed that the maximum number of stable marriages for instances of size n grows exponentially with n, but a definitive upper bound has yet to be defined.

Note *$r(M)$ could also be considered a social cost function but throughout this document when we refer to the social cost function you can assume we are referring to $c(M)$.*

1.2 Problem Statement

In this paper we will address two problems concerning the stable marriage problem.

Problem 1: Comparing the variants
The first problem is establishing the difference in satisfaction amongst individuals in the male optimal marriage M_o, minimum regret marriage M_r and egalitarian marriage M_e by specifying their average difference in maximum regret and average difference in social cost as functions of n. For some i and j, we define the average differences in maximum regret and social cost for each marriage pair as follows:

Avg max regret difference: $\overline{rd}(M_i, M_j) = \overline{r}(M_i) - \overline{r}(M_j)$

Avg social cost difference: $\overline{cd}(M_i, M_j) = \overline{c}(M_i) - \overline{c}(M_j)$

where $\mathbf{\overline{r}(M)}$ and $\mathbf{\overline{c}(M)}$ equals the average maximum regret and social cost respectively for the marriage M.
But our goal is to re-define these average difference functions as functions of n for each marriage pair $\{M_o, M_r\}$, $\{M_o, M_e\}$ and $\{M_r, M_e\}$, such that we have six new functions of the form $\mathbf{\overline{rd}_{ij}(n)}$ and $\mathbf{\overline{cd}_{ij}(n)}$. These functions can then be used to estimate the average differences in satisfaction for any value of n.

Problem 2: Analysing the number of stable marriages
Secondly, we want to build upon previous analysis on the number of stable marriages by specifying a confidence interval as a function of n for the total number of stable marriages. This will involve defining the average number of stable marriages and it's variance as functions of n. We denote these mean and variance functions $\overline{x}(n)$ and $\hat{\sigma}^2(n)$ respectively.

We also want to provide some insight on obtaining a good estimate on the upper bound for the number of stable marriages.

1.3 Approach

Problem 1 Approach
We approach the first of these problems by designing a program which finds the marriages M_o, M_r and M_e, and calculates their maximum regret $r(M)$ and social cost $c(M)$. We implement algorithms from three different sources to find these three stable marriages. We find M_o by implementing the Gale-Shapley[1962] $O(n^2)$ alogrithm, M_r by implementing Gusfield's[1987] "Algorithm B", and M_e using a combination of Gusfield's "Algorithm A" for finding all rotations and Irving, Leather and Gusfield's[1987] algorithm for finding the maximum weighted downset of the rotation poset. We will discuss these algorithms in greater detail in Chapter 2.

Upon implementation of these algorithms, we generate data to evaluate the satisfaction of individuals across the three marriages. We generate this data by running n^2 tests for different values of n using random preference lists. For each batch of n^2 tests we compute the average differences $\mathbf{\overline{rd}(M_i, M_j)}$ and $\mathbf{\overline{cd}(M_i, M_j)}$ for the marriage pairs $\{M_o, M_r\}$, $\{M_o, M_e\}$ and $\{M_r, M_e\}$. Then for each marriage pair, using quadratic regression we fit curves to our generated data to estimate the functions $\mathbf{\overline{rd}_{ij}(n)}$ and $\mathbf{\overline{cd}_{ij}(n)}$ for each marriage pair.

Problem 2 Approach
In order to build upon previous analsis of the number of stable marriages we design a program which enumerates all stable marrriages for a given problem of size n. More specifically, we implement Gusfield's[1987] algorithm for constructing an enumeration tree which contains all possible stable marriages for a given problem (see Chapter 2).

Like with problem 1, we generate data by running n^2 tests for different values of n using random preference lists, but this time for each batch of tests we are computing the average number of stable marriages, it's variance and the maximum enumeration for that batch. Again similarly to problem 1, we use quadratic regression to fit curves to our generated data to estimate the functions $\overline{x}(n)$ and $\hat{\sigma}^2(n)$. We also fit a curve to our maximum enumeration data.

1.4 Outcome

Using the approach for problem 1 from section 1.3, we obtained the following functions:

1. $\overline{rd}_{or}(n) \approx \delta n^2 + 0.41n - 7.57$
2. $\overline{cd}_{or}(n) \approx 0.07n^2 - 3.4n - 92.75$
3. $\overline{rd}_{oe}(n) \approx \epsilon n^2 + 0.36n - 7.71$
4. $\overline{cd}_{oe}(n) \approx 0.08n^2 - 3.34n + 81.33$
5. $\overline{rd}_{er}(n) \approx -(3.68E^{-5})n^2 + 0.05n + 0.14$
6. $\overline{cd}_{re}(n) \approx 0.001n^2 + 0.63n - 11.42$

and using the approach for problem 2 from section 1.3 we obtained the following confidence interval:

$$[\ (0.001n^2 + 0.55n - 7.73)\ \pm\ t_{n^2-1}(0.025)\frac{(0.23n^2 - 29.39n + 781.64)}{n}\ ]$$

See Chapter 5 for more detail of these outcomes.

Chapter 2

Background

The conduct of this project required a high level understanding of algorithms coming from various sources. In this chapter we outline those algorithms and the prerequisite knowledge needed to understand them.

2.1 Male/Female Optimal Stable Marriage

The Gale-Shapley[1962] algorithm for finding a stable marriage can be used to find the male or female optimal marriage depending on the implementation of the algorithm. The implementation we describe will be for finding the male optimal marriage, however we can easily find the female optimal marriage by simply swapping the roles of the men and women in the algorithm. Firstly, each man will propose to the woman who is first in his preference list. Then each woman who has received one or more proposals will become preliminarily engaged to the proposee highest on her preference list and will reject all other proposees. The rejected men then proceed to propose to the woman next on their preference list and the women once again consider these new proposals. If a woman who is preliminarily engaged receives a new proposal from someone she likes better than her current engagement, she will break of her current engagement and become preliminarily engaged to the new proposer. This algorithm terminates when all women have received a proposal.

2.2 Rotations

Rotations are a basic object defined by Irving and Leather[1986]. They define a rotation of a stable marriage M to be the set of pairs in M that we can swap(rotate) to obtain a new, equally valid stable marriage. If such a rotation exists in M it is said to be "exposed" in M.

A rotation π of a marriage M_a is written in the following form:

$$\pi = \{(m_1, w_1), (m_2, w_2), ..., (m_{r-1}, w_{r-1})\}$$

where each pair (m_i, w_i) is a pair in M_a and $r \leq n + 1$.
To obtain a new stable marriage M_b we "eliminate" π in M_a such that each man m_i in π is now married to woman $w_{(i+1)modr}$ in π, which also by implication means that each woman w_i in π is now married to man $m_{(i-1)modr}$ in π. All other pairs in M_a that are not in π remain unchanged in M_b. Each man m_i in π has a strictly less preferred spouse (i.e. lower on his preference list) in M_b than in M_a, and conversely each woman w_i in π has a strictly more preferred spouse in M_b than in M_a. This is true for the elimination of any rotation.

Given any stable marraige problem of size n, we can find all possible ro-

tations using Gusfield's[1987] "Algorithm A" in $O(n^2)$ time. We note that rotations can only be exposed in certain marriages. The algorithm starts at the male optimal marriage and proceeds to find and eliminate rotations to find new marriages and expose new rotations. The algorithm terminates when we find the female optimal marriage. For more indepth detail on the steps involved in Algorithm A please refer to Gusfield's concise summary in section B.1 of Appendix B.

Since rotations can only be exposed in certain marriages, and certain marriages can only be found by eliminating certain rotations, we are able to define a partial order over the rotations. Irving and Leather give the following definition of an "explicit predecessor" of a rotation:

Definition [Irving and Leather, 1986]. Let π and ρ be two distinct rotations. Rotation π is said to *explicitly precede* ρ if and only if π eliminates a pair (m, w), and ρ moves m to a woman w' such that m (strictly) prefers w to w'. The relation *precedes* is defined as the transitive closure of the relation "explicitly precedes".

A rotation that simply precedes another rotation but not explicitly can be thought of as an implicit predecessor. The "precedes" relation defines the partial order over the rotations. Rotations can only be eliminated once all predecessors of that rotation have been eliminated.

2.3 Rotations Graph

The rotations graph is simply a graph representation of the rotation poset, where each node is labelled by a unique rotation and each edge maps a rotation π_i to a rotation π_j if and only if π_i is an explicit predecessor of π_j. We implement a construction defined by Gusfield[1987].

2.4 Break Marriage

Operation breakmarriage(M, m) is a method defined by Gusfield[1987] which restarts the Gale Shapley algorithm described in section 2.1 by breaking the marriage of m and w (where w is m's spouse in the stable marriage M). Man m becomes a "free man" and proposes to the woman following w in his preference list, initiating a series of acceptances and rejections. Woman w is "semi-free" and will only accept a proposal if she prefers the proposee to m. Break marriage terminates either if some man has been rejected by all women meaning we weren't able to find a new stable marriage and thus M is returned, or if w receives a proposal from a man m' whom she prefers to m meaning we have found a new stable marriage M'.

2.5 Minimum Regret Stable Marriage

As stated previously in the introduction, if (m, w) is a pair in a stable marriage M then the regret of man m equals the position of w in his preference list (vice versa for the woman), and we define $r(M)$ as the maximum regret out of all individual in M. This idea of minimizing regret was proposed by Knuth[1976]. The minimum regret marriage is the stable marriage which has the lowest maximum regret out of all possible stable marriages for a given problem. There can exist more than one minimum regret marriage.

Two important variations of the minimum regret marriage proposed by Gusfield[1987] are the "man regret minimum stable marriage" and the

"woman regret minimum stable marriage". The man regret minimum marriage is the minimum regret stable marriage for which there exists at least one man with regret equal to the maximum regret of that marriage. Note that a man is said to have maximum regret in a stable marriage M if and only if his regret in M equals $r(M)$. We just replace man with woman to get the definition of the woman regret minimum marriage. We find these man and woman regret marriages by implementing Gusfield's "Algorithm B". The algorithm involves using breakmarriage(M, m) to find the woman regret minimum for some man m, and using breakmarriage(M, w) to find the man regret minimum for some woman w (note that breakmarriage(M, w) simply swaps the role of the men and women in the algorithm). Please refer to section B.2 of Appendix B for Gusfield's outline of algorithm B, specifically for finding the woman regret minimum. The minimum regret marriage simply equals the marriage with the smallest maximum regret out of the man and woman regret minimum marriages.

2.6 Egalitarian Stable Marriage

The objective of finding the egalitarian stable marriage is to optimise the average "satisfaction" of all individuals. This problem was first put forward by Knuth[1976] and solved by Irving, Leather and Gusfield[1987]. The focus of their solution was to minimise the social cost function $c(M)$. As we stated in the introduction, $c(M)$ is the sum of all indivdual regrets in the marriage M. Let $mr(m_i, w_i)$ and $wr(w_i, m_i)$ be the regret of man m_i and woman w_i respectively, where (m_i, w_i) is a stable pair in some stable marriage M. Thus, for a problem of size n we define the social cost function as follows:

$$c(M) = \sum_{i=1}^{n}(mr(m_i, w_i) + wr(w_i, m_i))$$

where $M = \{(m_1, w_1), (m_2, w_2), ..., (m_n, w_n)\}$.

Let's let $M_0 = M_o$ (i.e. the male optimal marriage). Irving, Leather and Gusfield redefine the social cost function,

$$c(M) = c(M_0) - \sum_{i=1}^{t} w(\pi_i)$$

where for some rotation $\pi = \{(m_1, w_1), (m_2, w_2), ..., (m_{r-1}, w_{r-1})\}$,

$$w(\pi) = \sum_{i=1}^{r-1}(mr(m_i, w_i) - mr(m_i, w_{i+1})) \ + \ \sum_{i=1}^{r-1}(wr(w_i, m_i) - wr(w_i, m_{i-1}))$$

Note: Rotations $\pi_1, \pi_2, ..., \pi_t$ must form a closed subset (downset) of the rotation poset.

We call $w(\pi)$ the weight of the rotation π. To find the egalitarian stable marriage we need to find the maximum weighted downset of the rotation poset, where the weight of a downset is simply the sum of the weights of the rotations in that downset. We find the maximum weighted downset by constructing a network flow graph and finding it's minimum cut. This network flow graph is created from the rotations graph G. We convert G to a network flow graph by adding a source with edges connected to each negatively weighted node (i.e. nodes labelled by negatively weighted rotations) and adding a sink with edges connected to each positively weighted node (i.e. nodes labelled by positively weighted rotations). The capacity of these edges is the absolute value of the rotation weight. All other edges (i.e. the edges in G) have infinite capacity. We find the minimum cut using the well know Ford-Fulkerson[1956] algorithm for finding the maximum flow of a network, which runs in $O(n^4)$ time. See fig. 4.1 for a picture of the network flow graph.

We get the maximum weighted downset by marking all nodes connected to the sink which are uncut by the mininmum cut and also marking these node's predecessors. The marked nodes make up the maximum weighted downset which we eliminate to get the egalitarian stable marriage. This whole method is by Irving, Leather and Gusfield [1987].

2.7 Enumeration of the Stable Marriages

Irving and Leather[1986] tell us that the number of stable marriages is equal to the number of downsets in the rotation poset. Gusfield[1987] uses this fact to construct an "enumeration tree" T, where each node is labeled with a unique stable marriage and each edge is labeled with a rotation, such that the rotations along the path from the root node M_0 (i.e the male optimal marriage) to any node M_i are the ones we eliminate (starting from M_0) to obtain the marriage M_i.

T contains all the possible stable marriages for a given problem, thus we simply count the number of nodes to enumerate the stable marriages. Please see section B.3 of Appendix B for Gusfield's summary of the the enumeration tree construction. See fig. 4.2 for a picture of the enumeration tree generated by our program.

Chapter 3

Design

In this chapter we outline the design of our software. We describe the main objects and alogrithms that comprise our program, and discuss the changes made from our original design plan. For a full picture of our design please also refer to the original design in the appendix as a supporting document to this chapter.

3.1 The Objects

Our program is comprised of six main objects; the "Person" object, the "Marriage" object, the "Rotation" object, the Rotations "Graph" object, the "Enumeration Tree" object and the "Marriage Problem" object. Here we will summarise the characteristics of those objects, their subclasses and their relationships with other objects.

3.1.1 Person Object

The Person object has two subclasses; the "Male" object and the "Female" object. The attributes of each instance of the Person object can be summarised by the following UML class diagram:

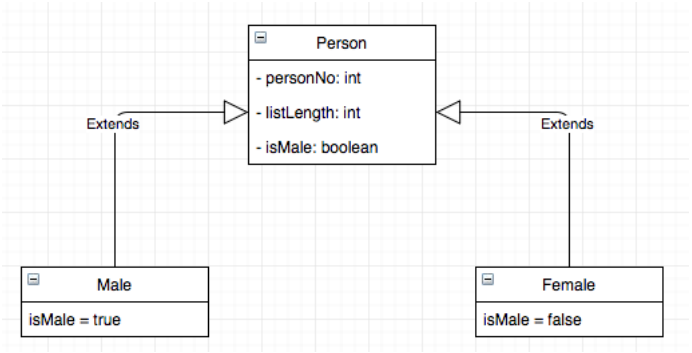


Figure 3.1: Person Object UML Class Diagram

Each person has a unique "person number" amongst individuals of the same sex as themselves. Two individuals of the opposite sex can have the same person number, as we can differentiate between them with the "isMale" variable. We initiate each individual's preference list using the "listLength" variable which is the assigned length of the person's preference list.

3.1.2 Marriage Object

For a problem of size n, each Marriage instance contains n pairs of men and women. Here is it's UML class diagram:

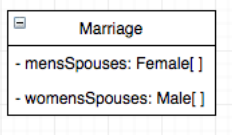


Figure 3.2: Marriage UML Class Diagram

We store each man's spouse and each woman's spouse in two separate arrays. We can find a man's spouse by simply looking at the index corresponding to his person number. This is the same for finding a woman's spouse. This allows us to quickly access any individual's spouse in constant time.

3.1.3 Rotation Object

For each Rotation instance we input the actual rotation
$\pi = (m_1, w_1), (m_2, w_2), \ldots, (m_{r-1}, w_{r-1})$ in the form of a linked list, such that rotation[0] $= m_1$, rotation[1] $= w_1$, rotation[2] $= m_2$, rotation[3] $= w_2$, and so on. We also calculate each rotation's weight and assign each rotation with a unique rotation number. We store the rotation's explicit predecessors and successors, this is important for the construction of the Rotations Graph. The Rotation object is summarised by the UML class diagram below:

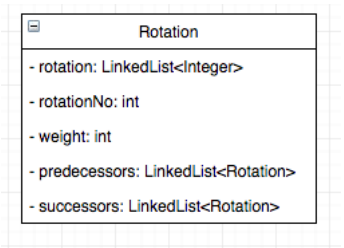


Figure 3.3: Rotation Object UML Class Diagram

3.1.4 Rotations Graph Object

Each node in the rotations graph is labelled by a unique rotation π. The parents of a node are the nodes which are labelled by the predecessors of π. The children of a node are the nodes which are labelled by the successors of π. We add source and sink nodes to convert this graph into a network flow graph, this is important for finding the egalitarian stable marriage. Here is it's UML class diagram:

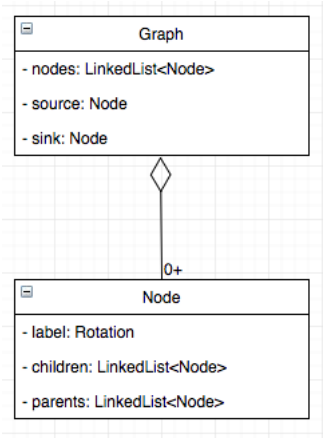


Figure 3.4: Graph Object UML Class Diagram

3.1.5 Enumeration Tree Object

For an enumeration tree instance we have a root node which we can use to access all other nodes self referentially (since each node store it's children, so we can sequentially access all nodes by starting at the root). Here is it's UML class diagram:

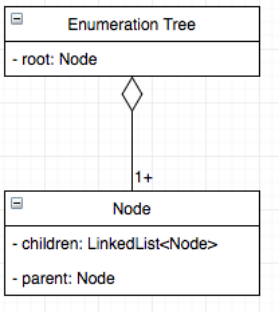


Figure 3.5: Tree Object UML Class Diagram

3.1.6 Marriage Problem Object

The marriage problem object is where we actually encapsulate the stable marriage problem. The marriage problem class is where we implement all the algorithms required for our software. Here is the Marriage Problem UML class diagram:

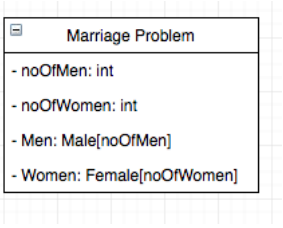


Figure 3.6: Marriage Problem Object UML Class Diagram

3.2 The Algorithms

There are five main components to implement as part of our software; finding the male/female optimal stable marriage, finding all rotations, finding the minimum regret stable marriage, finding the egalitarian stable marriage, and enumerating all stable marriages.

3.2.1 Finding the Male/Female Optimal Marriage

We find the male and female optimal marriages using the Gale-Shapley[1962] algorithm which we outlined in 2.1 of Chapter 2. See fig. D.1 in Appendix D for the psuedo-code for the alorithm taken from our original design.

3.2.2 Finding all Rotations

We find all rotations by implementing Gusfield's "Algorithm A". You can see Gusfield's outline of the algorithm in section B.1 of Appendix B. Here is our pseudo-code for the algorithm:

```
1  void gusfieldsAlgorithmA()
2  {
3      // Algorithm can only be run once for each problem instance
4      if (foundRotations)
5          throw ("Already found all rotations for this problem");
6
7      // Start with the man optimal marriage
8      Marriages currentMarriage = manOptimalMarriage;
9      Male currMan;
10
11     // Terminates when we find the woman optimal marriage
12     while (currentMarriage != womanOptimalMarriage)
13     {
14         /* Find the next man who's spouse is different in the current marriage and
15            the woman optimal marriage */
16         currMan = getNextBreakableMan(currentMarriage);
17
18         /* The adjusted break marriage method finds the next stable marriage. We call it
19            the "adjusted" break marriage method because it also stores the resultant
20            rotation of the new stable marriage in a class list */
21         currentMarriage = adjustedBreakMarriage(currentMarriage, currMan);
22
23         /* Each time a woman receives a proposal in the adjusted break marriage method
24            she is marked. This is so we know what women are involved in the rotation when
25            we find one. At the end of each loop we reset the marked women. */
26         unmarkWomen();
27
28     }
29
30     foundRotations = true;
31 }
```

Figure 3.7: Algorithm A Psuedo Code

After finding all rotations we can then construct the partial order in $O(n^2)$ time using a method by Gusfield[1987]. This can also be seen as the construction of the rotations graph. Here is our pseudo code for that method:

```
void constructPartialOrder()
{
    // labels each unstable pair by the rotation it was eliminated by
    labelEliminatedPairs();

    for each man ∈ the list of men
    {
        int currPrefRank = man.getManOptSpouseRank();

        /* Go through each man's preference list one by one starting at the position
           of that man's partner in the male optimal marriage. Stop when we reach
           the man's partner in the woman optimal marriage. */
        while (currPrefRank != man.getWomanOptSpouseRank())
        {
            currPrefRank += 1;
            Female currPref = man.getPref(currPrefRank);

            /* Get the rotation that labels the pair (man, currPref) (i.e. rot2)
               and the rotation that labels the pair (man, lastStablePref)
               (i.e. rot1), where lastStablePref is the last pref we encountered
               that was stable. */
            Rotation rot1 = man.getPairLabel(lastStablePref);
            Rotation rot2 = man.getPairLabel(currPref);

            // check that the current man has a label for the current preference
            if (man.hasLabel(currPref))
            {
                // if currPref is a stable pairing with currMan we add a rotation predecessor
                if (man.isStable(currPref))
                {
                    // if rot1 is not already a predecessor of rot2 add it
                    if (!rot2.isAPredecessor(rot1))
                    {
                        rot2.addPredecessor(rot1);
                        rot1.addSuccessor(rot2);
                    }
                }

                // else if the pair is unstable
                else
                {
                    /* Check to see if we have already encountered the label of
                       the unstable pair in the man's preference list. Also check if rot1
                       is not already a predecessor of rot2 */
                    if (!man.hasEncountered(rot2) && !rot1.isAPredecessor(rot2))
                    {
                        rot1.addPredecessor(rot2);
                        rot2.addSuccessor(rot1);
                    }
                }
            }
        }
    }
}
```

Figure 3.8: Partial Order Construction Psuedo Code

3.2.3 Finding the Minimum Regret Marriage

We find the minimum regret stable marriage by implementing Gusfield's "Algorithm B". Here is our psuedo code for Algorithm B:

```
Marriages gusfieldsAlgorithmB()
{
    /* The woman regret minimum doesn't exist if there is no woman with regret
       equal to the maximum regret of the man optimal marriage. In such cases
       the man optimal marriage is the minimum regret marriage so we return it. */
    if (!womanRegretMinExists())
        return manOptimal;

    /* The man regret minimum doesn't exist if there is no man with regret
       equal to the maximum regret of the woman optimal marriage. In such cases
       the woman optimal marriage is the minimum regret marriage so we return it.*/
    if (!manRegretMinExists())
        return womanOptimal;

    // If both the man and woman regret minimum's exist we can execute Gusfield's Alg B
    Marriages womanRegretMin = findWomanRegretMin();
    Marriages manRegretMin = findManRegretMin();
    Marriages minRegretMarriage;

    // Take the one with minimum max regret as the minimum regret marriage
    if (womanRegretMin.calculateMaxRegret() <= manRegretMin.calculateMaxRegret())
        minRegretMarriage = womanRegretMin;

    else
        minRegretMarriage = manRegretMin;

    return minRegretMarriage;
}
```

Figure 3.9: Algorithm B Psuedo Code

Finding the man and woman regret minimum is dependent on the break marriage method. The pseudo-code for this method from our original design can be seen in section D.3 of D.

3.2.4 Finding the Egalitarian Marriage

We find the egalitarian stable marriage using a method by Irving, Leather and Gusfield[1987]. You can see a description of this method in section 2.6 of Chapter 2. There are two main elements to finding the egalitarian stable marriage; constructing the network flow graph and finding the maximum flow of the network. The construction of the network flow graph is simple once we've constructed the partial order (see fig. 3.8), we simply add a source and sink. Here is our pseudo code for the Ford-Fulkerson[1956] algorithm for finding the maximum flow of a network:

```
void FordFulkerson()
{
    while (pathExists)
    {
        curPath = findPath();
        flow = getBottleneckFlow(curPath);
        pushFlowThroughPath(curPath, flow);
    }
}
```

Figure 3.10: Ford-Fulkerson Algorithm Pseudo Code

3.2.5 Finding the Number of Stable Marriages

We enumerate the stable marriages by constructing an enumeration tree using an algorithm defined by Gusfield[1987]. Here is the pseudo code for that construction:

```
// This is a recursive procedure, the initial start node is the root
void buildTree(Node startNode)
{
    // check if the start node has children
    // This is the termination condition of the recursion
    if (startNode.hasChildren())
    {
        // Get the children of the start node
        /* Note that the root node is already initialised before the start of
        this procedure */
        ArrayList<Node> children = startNode.getChildren();

        for each child node i ∈ children
        {

            Node curNode = i;
            Rotation curRotation = curNode.getLabel();
            int rotationLabel = curRotation.getRotationNo();

            /* Get the rotation labels along the path from the root node
               to the current node */
            LinkedList<Rotation> ancestorLabels = getAncestorLabels(curNode);


            /* Rotations have been labelled numerically according to the
               topological ordering of the Rotations graph. So we only need
               to check rotations with a larger label (i.e. check it's rotation no.). */
            for each rotation π with rotation number  > rotationLabel
            {
                /* if rotation  π doesn't have any predecessors when we exclude it's
                   ancestors then add it as a child of the current node */
                if (!π.hasPredecessors(ancestorLabels))
                    curNode.addChild(new Node(curNode,  π) );
            }

            // recursive statement
            if (curNode.hasChildren())
                buildTree(curNode);
        }
    }
}
```

Figure 3.11: Enumeration Tree Construction Pseudo Code

3.3 Changes to original design

There have not been any real changes from the original design apart from the omission of the additional variations which we did not end up having the time to implement and evaluate. These variations were the stable marriage with indifference, the asymmetric stable marriage problem, the stable marriage problem with weighted preference lists and the stable marriage problem with priority weighting.

However, the final design of our software which we have outlined in this chapter is more detailed than our original design and includes additional algorithms such as Gusfield's "Algorithm A" and the enumeration tree construction.

We also have additional objects that were not included in our original design. These additional objects are the rotation, rotations graph and enumertaion tree objects. See D.2 of Appendix D for the UML class diagram of our original design.

Chapter 4

Implementation

In this chapter we will showcase the implementation of our software with code listings and outputs for different examples. We will be heavily referring to the example in C.1 of Appendix C throughout this chapter. You should also see the software report document in the appendix folder of the electronic submission.

4.1 Finding the Male/Female Optimal Marriage

Our implementation of the Gale-Shapley algorithm was similar to our original design in fig D.1 of Appendix D. You can see our output in fig C.2 of Appendix C.

4.2 Finding all Rotations

We implemented Gusfield's "Algorithm A", largely by following the psuedo code given in fig 3.7.

Algorithm A is dependent on a method which we call the "adjusted break marriage" method. We call it the "adjusted" break marriage because Gusfield also defines a method called "break marriage" which is slightly different (see in section D.3 of Appendix D for details of the break marriage method). You can see the psuedo code for our implementation of the adjusted break marriage method in section A.2 of Appendix A which is a condensed version of our actual implementation.

Our implementation of the construction of the partial order is the same as our design in fig. 3.8. The resultant outputted rotations graph can be seen in fig. 4.1 by ignoring the blue edges.

The outputted rotations can be seen in fig. C.3 of Appendix C.

4.3 Finding the Minimum Regret Marriage

We implement Gusfield's Algorithm B in the same way as the outlined design in fig. 3.9 of chapter 3.

We find the woman regret minimum by following the steps given by Gusfield seen in section B.2 of Appendix B. Our implementation of breakMarriage(M, m) (which is a component of the steps given by Gusfield) is similar to our outline in D.3 from our original design.

We find the man regret minimum by also following the steps in section B.2 of Appendix B, but we instead start the alogrithm at the woman optimal marriage and move towards the man optimal marriage using breakMarriageWomen(M, w) (which simply swaps the role of the men and women in the

break marriage method).

The outputted minimum regret stable marriage can be seen in fig. C.4 of Appendix C.

4.4 Finding the Egalitarian Marriage

We implement the Ford-Fulkerson alogrithm in the same way as the outlined design given in the psuedo code in fig. 3.10.

To see our implementation for finding a path/augmented path in our Ford Fulkerson alogrithm and subsequently findin the minimum cut please see our code listing in section A.3 of Appendix A.

Below is the outputted rotations graph from our program. The rotations graph is outputted each time we run our program.

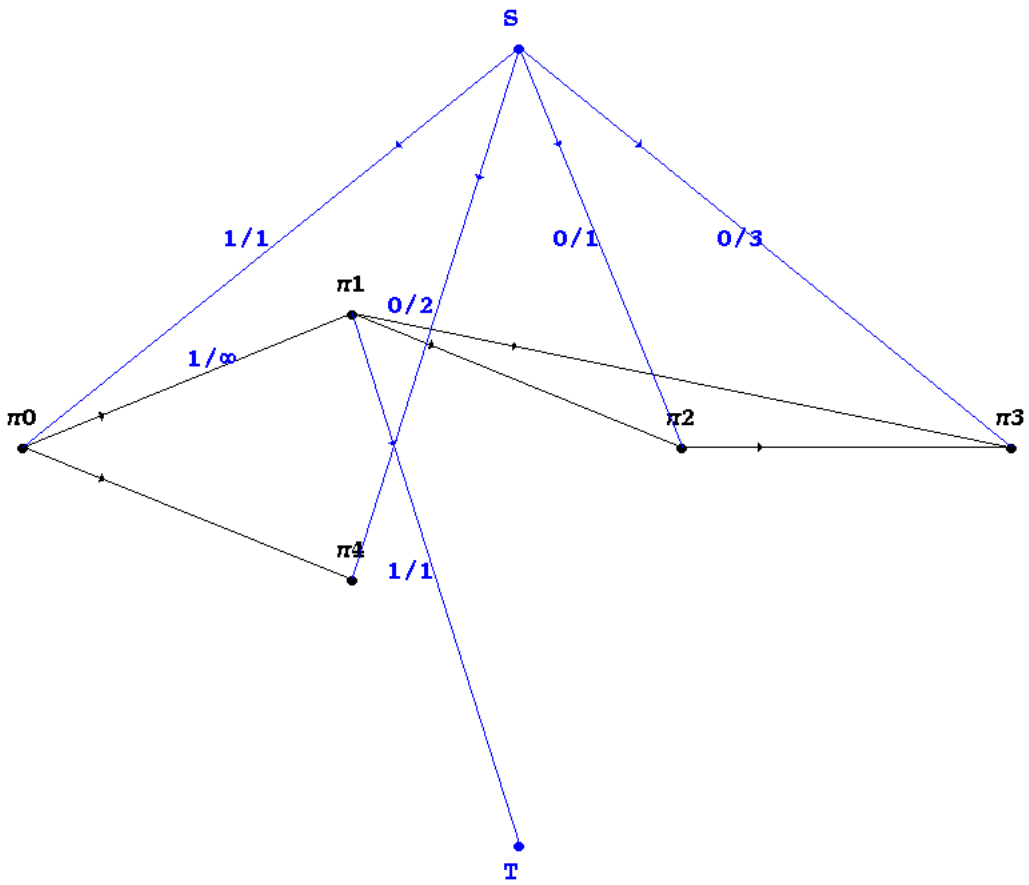


Figure 4.1: Network Flow Graph Output

The minimum cut of this graph is just the edge from the source S to π_0. We mark the nodes that have uncut edges connected to the sink T and we also mark their predecessors. So in this case we would mark π_1 and π_0. This is the maximum weighted downset of this example. See fig. C.5 for the ouputted egalitarian stable marriage.

4.5 Finding the Number of Stable Marriages

We construct this enumeration tree in the same way that we outlined in the design in fig.3.11. Below is the outputted enumeration tree from our program which is outputted each time we run our program.

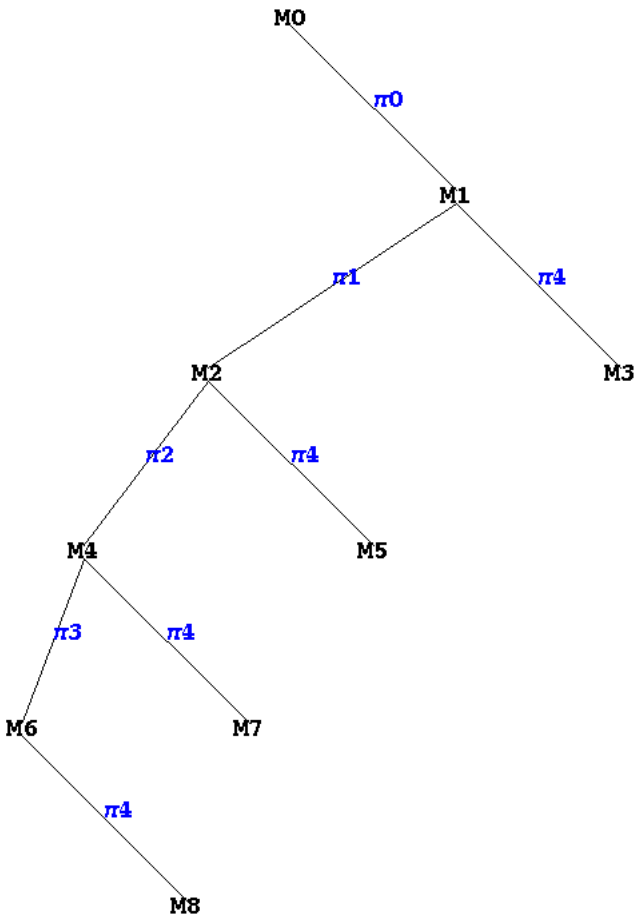


Figure 4.2: Enumeration Tree Output

We count the number of nodes to get the number of stable marriages, so in this example we have 9 stable marriages.

Chapter 5

Evaluation

In this chapter we will discuss the outcomes of our project and the significance of these outcomes.

We split our evaluation into two segments; Comparing variants of the stable marriage problem and analysing the number of stable marriages.

Note *before reading this chapter you should read section 1.2 of Chapter 1 as we will be using functions and terminology defined in that section. Also, a reminder that when we refer to the social cost we are always referring to* $c(M)$.

5.1 Comparing the Variants

5.1.1 Method

For the first segment of our project we wanted to compare the satisfaction of individuals across three stable marriages; the male optimal marriage M_o, the minimum regret marriage M_r and the egalitarian marriage M_e. In order to do this we first took each marriage pair $\{M_o, M_r\}$, $\{M_o, M_e\}$ and $\{M_r, M_e\}$, and calculated their average difference in maximum regret $\overline{rd}(M_i, M_j)$ and average difference in social cost $\overline{cd}(M_i, M_j)$ for different problem sizes n. For each problem size n we would run n^2 tests using random preference lists to compute the averages. We did this for $n = 10, 20, 30, ..., 100$ and then $n = 150, 200, 250..., 450$. Please see the SPSS file in the appendix attached as part of the electronic submission which contains all our generated data. Also see fig. A.5 in Appendix A to see the code listing of how we generated the random preference lists for each person.

Having generated our data we then wanted to define the average differences in maximum regret and social cost as functions of n, $\overline{rd}_{ij}(n)$ and $\overline{cd}_{ij}(n)$. We did this by fitting a curve to our data for each of the six averages $\overline{rd}(M_o, M_r)$, $\overline{rd}(M_o, M_e)$, $\overline{rd}(M_r, M_e)$, $\overline{cd}(M_o, M_r)$, $\overline{cd}(M_o, M_e)$, and $\overline{cd}(M_r, M_e)$ using quadratic regression. We used SPSS for this fitting.

5.1.2 Results

We will now go through our results for each marriage pair.

Note *we round to 2.d.p. for the coefficients of each function.*

Difference in Satisfaction between M_o and M_r

Below we have the fitted curve of the average difference in maximum regret of M_o and M_r for different values of n. The **x-axis** is n and the **y-axis** is $\overline{rd}(M_o, M_r)$.

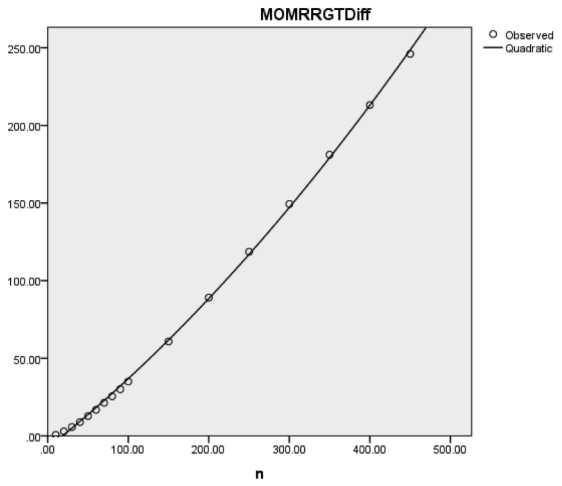


Figure 5.1: Average difference in **Maximum Regret** of M_o and M_r

The function of this curve is $\delta n^2 + 0.41n - 7.57$. Thus, approximatley

$$\overline{rd}_{or}(n) \approx \delta n^2 + 0.41n - 7.57.$$

where $0 \leq \delta < 0.01E^{-8}$.
The quadratic coefficient returned by SPSS was 0.000. This occurs in SPSS when the coefficient is less than $0.01E^{-8}$. Therefore, this means that the quadratic coefficient δ is less than $0.01E^{-8}$.

Below we have the fitted curve of the average difference in social cost of M_o and M_r for different values of n. The **x-axis** is n and the **y-axis** is $\overline{cd}(M_o, M_r)$.

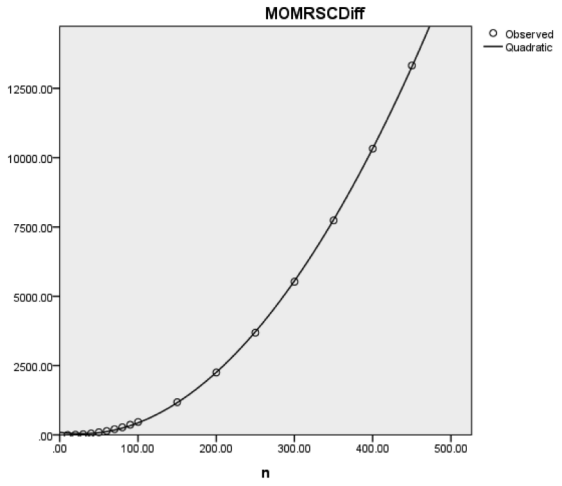


Figure 5.2: Average difference in **Social Cost** of M_o and M_r

The function of this curve is $0.07n^2 - 3.4n - 92.75$. Thus, approximatley

$$\overline{cd}_{or}(n) \approx 0.07n^2 - 3.4n - 92.75.$$

Difference in Satisfaction between M_o and M_e

Below we have the fitted curve of the average difference in maximum regret of M_o and M_e for different values of n. The **x-axis** is n and the **y-axis** is $\overline{rd}(M_o, M_e)$.

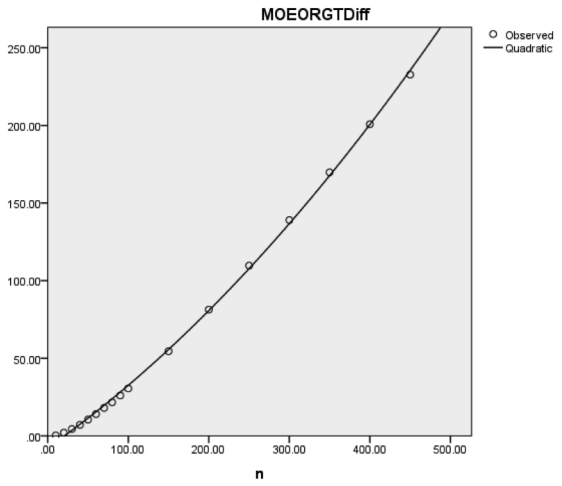


Figure 5.3: Average difference in **Maximum Regret** of M_o and M_e

The function of this curve is $\epsilon n^2 + 0.36n - 7.71$. Thus, approximatley

$$\overline{rd}_{oe}(n) \approx \epsilon n^2 + 0.36n - 7.71.$$

where $\epsilon < 0.01E^{-8}$
Once again the quadratic coefficient returned by SPSS was 0.000, which again means that the quadratic coefficient ϵ was less than $0.01E^{-8}$ and thus wasn't returned by SPSS.

Below we have the fitted curve of the average difference in social cost of M_o and M_e for different values of n. The **x-axis** is n and the **y-axis** is $\overline{cd}(M_o, M_e)$.

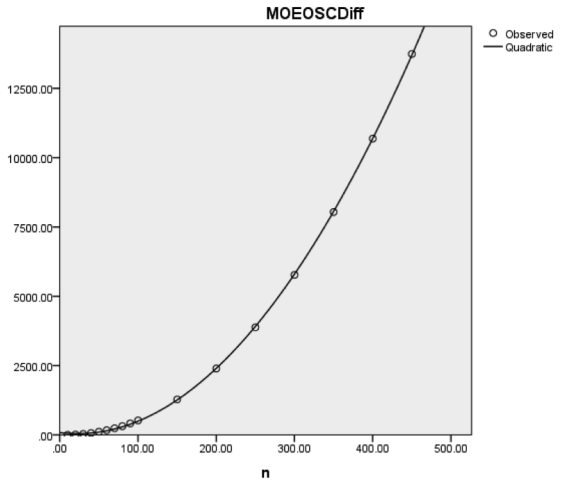


Figure 5.4: Average difference in **Social Cost** of M_o and M_e

The function of this curve is $0.08n^2 - 3.34n + 81.33$. Thus, approximatley

$$\overline{cd}_{oe}(n) \approx 0.08n^2 - 3.34n + 81.33.$$

Difference in Satisfaction between M_r and M_e

Below we have the fitted curve of the average difference in maximum regret of M_r and M_e for different values of n. The **x-axis** is n and the **y-axis** is $\overline{rd}(M_e, M_r)$.

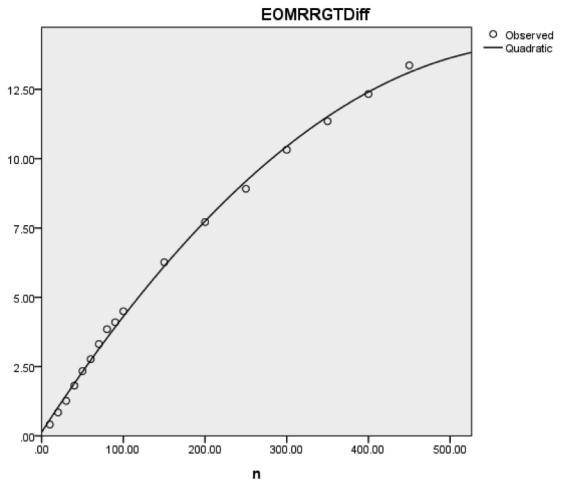


Figure 5.5: Average difference in **Maximum Regret** of M_r and M_e

The function of this curve is $-(3.68E^{-5})n^2 + 0.05n + 0.14$. Thus, approximatley

$$\overline{rd}_{er}(n) \approx -(3.68E^{-5})n^2 + 0.05n + 0.14.$$

Below we have the fitted curve of the average difference in social cost of M_r and M_e for different values of n. The **x-axis** is n and the **y-axis** is $\overline{cd}(M_r, M_e)$.

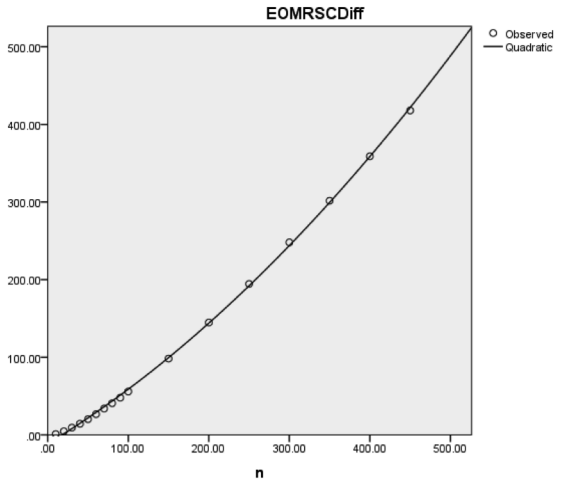


Figure 5.6: Average difference in **Social Cost** of M_r and M_e

The function of this curve is $0.001n^2 + 0.63n - 11.42$. Thus, approximatley

$$\overline{cd}_{re}(n) \approx 0.001n^2 + 0.63n - 11.42.$$

5.1.3 Analysis

Difference in Satisfaction between M_o and M_r

Maximum Regret

The average difference in maximum regret between the marriages M_o and M_r is almost linear, with only a very small quadratic term. As we can see from fig. 5.1, the curve fits the data very well suggesting that the corresponding function $\overline{rd}_{or}(n)$ gives an accurate prediction of $\overline{rd}(M_o, M_r)$ for any value of n. Since the quadratic term is so small it will take fairly large n before we see a significant difference in the speed at which $\overline{rd}(M_o, M_r)$ increases.

From this result we can conclude that there is a definite relationship between the problem size n and $\overline{rd}(M_o, M_r)$ which could be the subject of further research. We can also conclude that on average the maximum regret of individuals in M_o is consistently greater than that of individuals in M_r.

Social Cost

As we can see from fig. 5.2, the curve fits the data very well suggesting that the corresponding function $\overline{cd}_{or}(n)$ gives an accurate prediction of $\overline{cd}(M_o, M_r)$ for any value of n. The fact that the relationship is quadratic means that as n increases, $\overline{cd}(M_o, M_r)$ will begin to increase faster and faster.

From this result we can conclude that there is a definite relationship between the n and $\overline{cd}(M_o, M_r)$ which could be the subject of further research. We can also conclude that on average the social cost of individuals in M_o is consistently greater than that of individuals in M_r.

Difference in Satisfaction between M_o and M_e

Maximum Regret

The average difference in maximum regret between the marriages M_o and M_e is almost linear, with only a very small quadratic term. As we can see from fig. 5.3, the curve fits the data very well suggesting that the corresponding function $\overline{rd}_{oe}(n)$ gives an accurate prediction of $\overline{rd}(M_o, M_e)$ for any value of n. Since the quadratic term is so small it will take fairly large n before we see a significant difference in the speed at which $\overline{rd}(M_o, M_e)$ increases.

From this result we can conclude that there is a definite relationship between the n and $\overline{rd}(M_o, M_e)$ which could be the subject of further research. We can also conclude that on average the maximum regret of individuals in M_o is consistently greater than that of individuals in M_e.

Social Cost

As we can see from fig. 5.4, the curve fits the data very well suggesting that the corresponding function $\overline{cd}_{oe}(n)$ gives an accurate prediction of $\overline{cd}(M_o, M_e)$ for any value of n. The fact that the relationship is quadratic means that as n increases, $\overline{cd}(M_o, M_e)$ will begin to increase faster and faster.

From this result we can conclude that there is a definite relationship between the n and $\overline{cd}(M_o, M_e)$ which could be the subject of further research. We can also conclude that on average the social cost of individuals in M_o is consistently greater than that of individuals in M_e.

Difference in Satisfaction between M_r and M_e

Maximum Regret

As we can see from fig. 5.5, the curve fits the data very well suggesting that the corresponding function $\overline{rd}_{er}(n)$ gives an accurate prediction of $\overline{rd}(M_e, M_r)$ for any value of n. Unlike all the other curves, this one is concave. This means that as n increases, $\overline{rd}(M_e, M_r)$ will begin to increase slower and slower.

From this result we can conclude that there is a definite relationship between the n and $\overline{rd}(M_e, M_r)$ which could be the subject of further research. We can also conclude that on average the maximum regret of individuals in M_e is consistently greater than that of individuals in M_r. However this average difference is very small even for large n.

Social Cost

As we can see from fig. 5.6, the curve fits the data very well suggesting that the corresponding function $\overline{cd}_{re}(n)$ gives an accurate prediction of $\overline{cd}(M_r, M_e)$ for any value of n. The fact that the relationship is quadratic means that as n increases, $\overline{cd}(M_r, M_e)$ will begin to increase faster and faster.

From this result we can conclude that there is a definite relationship between the n and $\overline{cd}(M_r, M_e)$ which could be the subject of further research. We can also conclude that on average the social cost of individuals in M_r is consistently greater than that of individuals in M_e.

5.2 Analysing the Number of Stable Marriages

5.2.1 Method

For the second segment of our project we wanted to analyse the number of stable marriages by producing a confidence interval as a function of n. We also wanted to perform analysis on the maximum enumeration for different problem sizes and comment on the upper bound.

We did this by once again generating data by running a batch of n^2 tests for different values of n using random preference lists. We run these tests for $n = 10, 20, 30, ..., 100$ and then $n = 150, 200, 250..., 450$. For each batch of tests we computed the average number of stable marriages $\overline{x}$, it's variance σ and the maximum enumertation for that batch. We used quadratic regression to fit curves to our generated data to define the mean and variance as functions of n, $\overline{x}(n)$ and $\hat{\sigma(n)}$. We also fitted a curve to the maximum enumeration data using quadratic regression.

5.2.2 Results

We will now go through each of the resultant functions we obtained from our curve estimations and analyse the significance of each result.

The Average Number of Stable Marriages

Below we have the fitted curve for the average number of stable marriages $\overline{x}$ for different values of n. The **x-axis** is n and the **y-axis** is $\overline{x}$.

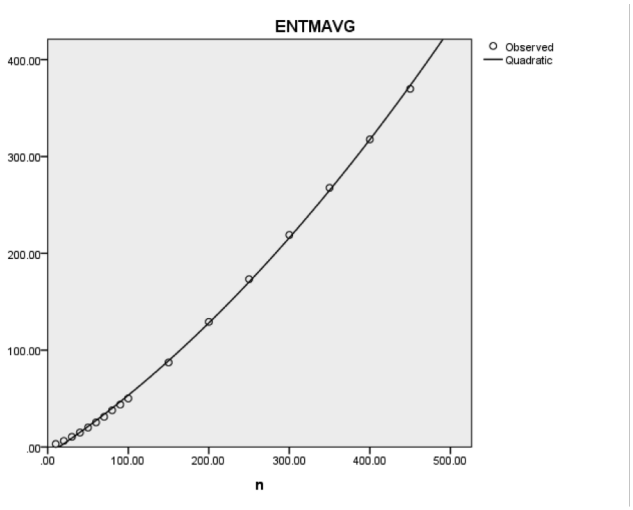


Figure 5.7: The Average Number of Stable Marriages

The function of this curve is $0.001n^2 + 0.55n - 7.73$. Thus, approximatley

$$\overline{x}(n) \approx 0.001n^2 + 0.55n - 7.73.$$

Below we have the fitted curve for the variance $\hat{\sigma}$ for different values of n. The **x-axis** is n and the **y-axis** is $\hat{\sigma}$.

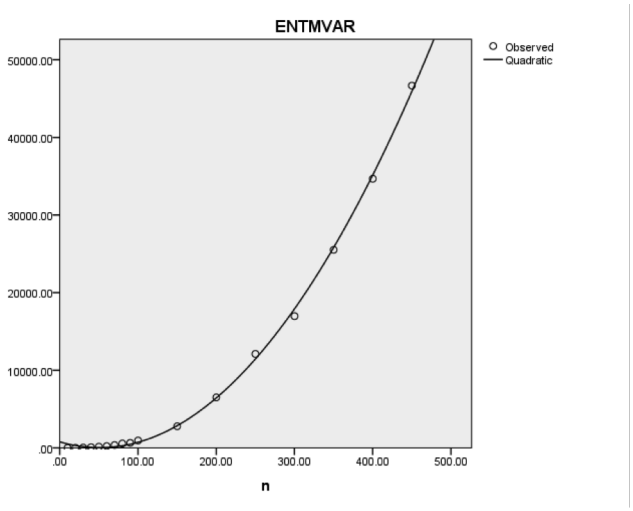


Figure 5.8: The Variance for the Number of Stable Marriages

The function of this curve is $0.23n^2 - 29.39n + 781.64$. Thus, approximatley

$$\hat{\sigma}(n) \approx 0.23n^2 - 29.39n + 781.64.$$

Confidence Interval

Let m be the number of tests we perform. In our case always $m = n^2$. We can produce a 95% confidence interval as function of n using the Central Limit Theorem[Pólya, 1920] for unknown variances. Here is that confidence interval:

$$[\ \overline{x}(n)\ \pm\ t_{m-1}(0.025)\tfrac{\hat{\sigma}(n)}{\sqrt{m}}\ ]\ \ =\ \ [\ \overline{x}(n)\ \pm\ t_{n^2-1}(0.025)\tfrac{\hat{\sigma}(n)}{n}\ ]$$

We simply substitute the functions for $\overline{x}(n)$ and $\hat{\sigma}(n)$ into our confidence interval to get:

$$[\ (0.001n^2 + 0.55n - 7.73)\ \pm\ t_{n^2-1}(0.025)\frac{(0.23n^2 - 29.39n + 781.64)}{n}\ ]$$

Note t_{m-1} *denotes the t-distribution*

Maximum Enumeration

Below we have the fitted curve for the maximum enumeration for different values of n. The **x-axis** is n and the **y-axis** is the max enumeration.

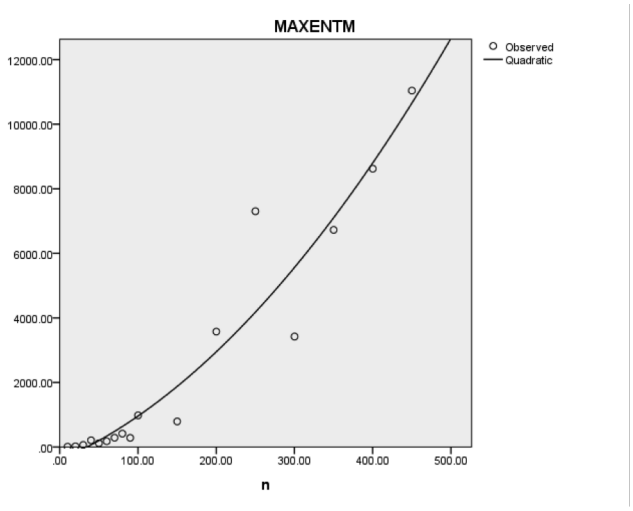


Figure 5.9: The Maximum Enumeration Plot

The function of the fitted curve is $0.03n^2 + 10.54n - 406.34$.

5.2.3 Analysis

The Average Number of Stable Marriages

Given how well the curves fit the data for the mean number of stable marriages and it's variance seen in fig. 5.7 and fig. 5.8 respectively, we can be fairly confident of the accuracy of our confidence interval in section 5.2.2.

Also given the accuracy of our curve estimation for the mean number of stable marriages, we conclude that for any problem size n the average number of stable marriages will be approximatley quadratic.

Maximum Enumeration

The data for the maximum enumertation is quite sporadic, for example the maximum enumeration for $n = 250$ is larger than the maximum enumeration for $n = 300$ and $n = 350$. Obviously the larger n is, the larger the maximum enumeration should be, so this result suggests we are not performing enough tests for each problem size n(currently performing n^2 tests) to find an enumeration close to the maximum.

Chapter 6

Conclusion

6.1 Learning Points

Skills Gained

Programming Skills
This project required me to produce a significant piece of software where I had to implement complex algorithms. This was the first large piece of software I've ever produced. As a result my skills in programming (in JAVA) have increased ten-fold. I now have much better idea on how to approach designing a large piece of software.
Time Management Skills
In this project it was vitally important that I managed my time efficiently to meet deadlines. This involved setting personal goals to complete certain aspects of the project and establishing adjustments I would make in the event I could not implement some aspect of my software.
Presentation Skills
My ability to deliver presentations has improved as a result of this project. Particularly I learned the importance of being able to deliver simple, well thought out explanations of the material I'm presenting.

Milestones

Here we list the milestones vital to the completion of the project.

- Implementation of Gale-Shapley Algorithm (June 15th)
- Implementation of Gusfield's Algorithm A (July 31st)
- Implementation of Gusfield's Enumeration Tree Construction (August 11th)
- Implementation of Irving, Leather and Gusfield's algorithm for finding the Egalitarian Marriage (August 17th)
- Implementation of Gusfield's Algorithm B (August 20th)

Improvements

I think overall my project has gone quite well. I've completed the majority of the goals I set out at the start of the project. However, there are some improvements that I think could have been made.

1. **More flexible code.** Due to the rigidity of my code, if I wanted to implement additional variations of the stable marriage problem (as I had planned to do) I would have to make a lot of adjustments to my current program.

2. **Better design plan.** On multiple occassions I had to almost completely restart writing my program, which wasted a lot of time. This was due lack of detail in the initial design plan.

6.2 Professional Issues

Here we will discuss how our project has related to relevant aspects of the BCS code of conduct.

Public Interest

My project acts in the public interest by respecting the rights of Third Parties. In this case those Third Parties are the authors of the literature that I used to complete the project. I showed due regard to those authors by referencing any work of their's I used in my project.

Professional Competence and Integrity

I undertook this project with confidence that I was competent to complete it, which I now have. I respected and listened to the critiques of my supervisors and responded by making adjustments to rectify these issues.

Duty to Relevant Authority and the Profession

I showed duty to the University of Liverpool and the Profession by ensuring all of the work I completed was my own and not misrepresenting any work done by others as my own.

6.3 Project Summary

In our project we have been able to write a substantial piece of software which implements various algorithms. As a result we were able to perform analysis that produced interesting findings in the form of curve estimations and a confidence interval seen in Chapter 5. Excluding the maximum enumeration fitting, our curve estimations fit the data extremely well which warrants further investigation to figure out the underlying reasons for this accuracy.

Directions for Further Work

- Investigate the underlying reasons for the accuracy of our curve estimations.

- Consider additional stable marriage variants such as the asymmetric or the weighted stable marriage problems.

- Perform a larger number of tests to obtain a more accurate prediction of the stable marriage enumeration. May require finding a more compact way of representing the number of stable marriages.

Bibliography

[1] Ford, L. R., & Fulkerson, D. R. (1956). Maximal flow through a network. *Canadian Journal of Mathematics, 8*(0), 399-404. https://doi.org/10.4153/CJM-1956-045-5

[2] Gale, D., & Shapley, L. S. (1962). College admissions and the stability of marriage. *The American Mathematical Monthly, 69*(1), 9-15.

[3] Gusfield, D. (1987). Three Fast Algorithms for Four Problems in Stable Marriage. *SIAM J. Comput., 16*(1), 111-128. https://doi.org/10.1137/0216010

[4] Irving, R. W., & Leather, P. (1986). The Complexity of Counting Stable Marriages. *SIAM J. Comput., 15*(3), 655-667. https://doi.org/10.1137/0215048

[5] Irving, R. W., Leather, P., & Gusfield, D. (1987). An Efficient Algorithm for the Optimal Stable Marriage. *J. ACM, 34*(3), 532-543. https://doi.org/10.1145/28869.28871

[6] Knuth, D. E. (1976). *Mariages stables et leurs relations avec d&autres problmes combinatoires.* Presses de lUniversit de Montral.

[7] Pólya, G. (1920). *ber den zentralen Grenzwertsatz der Wahrscheinlichkeitsrechnung und das Momentenproblem. Mathematische Zeitschrift, 8*, 171181.

Appendix A

Some Interesting Bits of Code

A.1 Enumeration Tree Construction

Below we have the source code for our enumeration tree construction algorithm:

```
57      // This is a recursive procedure, the initial start node is the root
58      private void buildTree(Node<Rotation> startNode)
59      {
60          // check if the start node has children
61          // This is the termination condition of the recursion
62          if (startNode.hasChildren())
63          {
64
65              // Get the children of the start node
66              /* Note that the root node is already initialised before the start of
67              this procedure */
68              ArrayList<Node<Rotation>> children = startNode.getChildren();
69              int noOfChildren = children.size();
70
71              for (int i = 0; i < noOfChildren; i++)
72              {
73                  Node<Rotation> curNode = children.get(i);
74                  Rotation curLabel = curNode.getLabel();
75                  int curLabelNo = curLabel.getRotationNo();
76
77                  /* Get the rotation labels along the path from the root node
78                     to the current node */
79                  LinkedList<Rotation> ancestorLabels = getAncestorLabels(curNode);
80
81                  /* Rotations have been labelled numerically according to the
82                     topological ordering of the Rotations graph. So we only need
83                     to check rotations with a larger label. */
84                  for (int j = curLabelNo+1; j < noOfRotations; j++)
85                  {
86                      /* if rotation j doesn't have any predecessors excluding it's
87                         ancestors add it to the current node */
88                      if (!rotations.get(j).hasPredecessors(ancestorLabels))
89                          curNode.addChild(new Node<Rotation>(curNode,rotations.get(j),curNode.getHeight() + 1));
90                  }
91
92                  // recursive statement
93                  if (curNode.hasChildren())
94                      buildTree(curNode);
95              }
96          }
97      }
```

Figure A.1: Enumeration Tree Construction Code

A.2 Adjusted Break Marriage Psuedo Code

Below we have our adjusted break pseudo code explaining our implementation:

```
/*
This method finds new marriages by breaking man m from his spouse w in M similar to
the break marriage method. However the difference is the inclusion of
pauseBreakMarriage which is used to store the rotations we find.
*/
Marriages adjustedBreakMarriage(Marriages M, Male m)
{
    // curMarriages corresponds to the last stable marriage found
    // mens/womens fiances holds the current state of the proposal sequence
    Marriages curMarriages = M;
    Female[] mensFiances = M.getMensSpouses();
    Male[] womensFiances = M.getWomensSpouses();
    Male freeMan = m;
    Female w = M.getSpouse(m);
    mark(w);
    while (freeMan.hasNextPref() && (womensFinaces[w.personNo] == m))
    {
        Female pref = freeMan.getNextPref();
        Male prefFiance = womensFiances[pref.personNo];

        /* If woman's proposee (i.e. pref) is marked and prefers the proposer
           (i.e. the free man) to her spouse in the curMarriage */
        // This means we've found a new stable marriage and subsequently a new rotation
        if (pref.isMarked() && pref.prefersTo(freeMan, pref.getSpouse(curMarriages)))
        {
            /* We call pause break marriage to make note of the rotation. It does this
               by writing down all marked women and their partners from the last stable
               marriage found. It then unmarks all these women */
            curMarriages = pauseBreakMarriage(curMarriages, mensFiances, womensFiances, freeMan);

            /* Check if pref prefers freeMan to her current
               fiance. If yes replace fiance with freeMan. Old fiance in now the free man.
               In the cases where we pause breakMarriage we treat the marking
               of women differently.  */
            if (pref.prefersTo(freeMan, prefFiance))
            {
                womensFiances[pref.personNo] = freeMan;
                mensFiances[freeMan.personNo] = pref;
                freeMan = prefFiance;
            }
            else
                mark(pref);
        }
        // if woman pref accepts proposal from the free man
        else if (pref.prefersTo(freeMan, prefFiance))
        {
            if (!pref.isMarked())
                mark(pref);
            womensFiances[pref.personNo] = freeMan;
            mensFiances[freeMan.personNo] = pref;
            freeMan = prefFiance;
        }
    }
    if (womensFiances[w.getPersonNo() - 1] == m)
        return M;
    return new Marriages(mensFiances, womensFiances);
}
```

Figure A.2: Adjusted Break Marriage Psuedo Code

This pseudo code is basically just a condensed version of our actual implementation. Please also see B.1 in appendix B as supporting material to this pseudo code.

A.3 Finding a Path in the Network Flow Graph

Below we have the source code for finding a path in the network flow graph as part of the our implementation of the Ford-Fulkerson algorithm:

```
433     private  LinkedList<Integer> findPath(boolean findingMinCut){
434         /* path holds the child index (parent index if augmented) of each node
435            in the path starting at the source */
436         // pathNodes holds the actual nodes involved in the path
437         Node<Rotation> curNode = source;
438         LinkedList<Integer> path = new LinkedList<Integer>();
439         LinkedList<Integer> pathNodes = new LinkedList<Integer>();
440         int num;
441 
442         /* terminates either when we reach our destination (i.e. the sink) or
443            when we can't find a path */
444         while(!visited[destination] && pathExists){
445 
446             // find next node finds a node we can push flow through and returns it's index
447             /* Each time a node is found it is marked as visited. This is so we do not
448                end up in a loop where we are repeatedly visiting the same node. We reset
449                all visited each time we find a path. */
450             num = findNextNode(curNode, findingMinCut);
451 
452             // if no node is found we move backwards to the previous node in the path.
453             // not found is an unreachable number > than destination
454             if (num == notFound){
455 
456                 if (pathNodes.size() != 0){
457                     path.remove(path.size() - 1);
458                     pathNodes.remove(pathNodes.size() - 1);
459                     if (pathNodes.size() != 0){
460 
461                         num = pathNodes.get(pathNodes.size() - 1);
462                         curNode = nodes.get(num);
463                     }
464                     else
465                         curNode = source;
466                 }
467                 else
468                     curNode = source;
469             }
470             else if (!visited[destination]){
471                 // if num is negative it means we move backwards along the edge (augmented)
472                 // Thus we store the parent index as opposed to the child index
473                 if (num < 0){
474                     int k = Math.abs(num) - 1;
475                     curNode = curNode.getParent(k);
476                     pathNodes.add(curNode.getLabel().getRotationNo());
477                     path.add(num);
478                 }
479                 // if num is positive we move forwards along
480                 else{
481                     curNode = curNode.getChild(num - 1);
482                     pathNodes.add(curNode.getLabel().getRotationNo());
483                     path.add(num);
484                 }
485             }
486             else
487                 path.add(num);
488         }
489         return path;
    }
```

Figure A.3: Code Listing for Finding a Path in a Network Flow

We find the next node in a path with the following code:

```
    /* Note if findingMinCut=true this method will find the minimum cut
       given we have already found the maximum flow */
    /* An edge is in the min cut iff it goes from a node that can have flow pushed
       through it to one that can't (again only given we've found the max flow) */
    private int findNextNode(Node<Rotation> curNode, boolean findingMinCut)
    {
        int index = 0;
        int num;
        /* if curNode has a child node that is the sink we want to check it first since
           if we can push flow through it then we have found a path */
        if (curNode.hasSinkChild())
        {
            /* if we can push flow through the sink, mark destination as reached and
               return the sink's child index for curNode */
            if (curNode.canPushFlow(curNode.getSinkChildIndex()))
            {
                index = curNode.getSinkChildIndex();
                visited[destination] = true;
                return (index + 1);
            }

            /* else if we can't push flow through the sink and we are finding the min cut
               then this edge is in the minimum cut */
            else if (findingMinCut)
            {
                int i = curNode.getLabel().getRotationNo();
                int j = curNode.getSinkChildIndex();
                minCut.add(i);
                minCut.add(j);
            }
        }
        // Go through each child node to check if we can push flow through it
        while (curNode.hasChild(index))
        {
            /* Already dealt with the sink above and the sink will be the last child alway
               so we break from the loop */
            if (curNode.getChild(index).isSink())
                break;

            else
            {
                // get the label of the current child
                num = curNode.getChild(index).getLabel().getRotationNo();
                // if can push flow through current child return it's index and mark as visited
                if (curNode.canPushFlow(index) && !visited[num])
                {
                    visited[num] = true;
                    return (index + 1);
                }
            }
            index++;
        }
        index = 0;
        /* If we come up empty with the above loop then we look for an augmented path where we
           move backwards along edges. Check each parent of the curNode to see if we can retract
           flow */
        while (curNode.hasParent(index) && !findingMinCut)
        {
            // If curNode has a parent which is the source we skip that parent
            if (curNode.getParent(index).isSource())
            {
                index++;
                continue;
            }

            else
            {
                // get the label of the current parent
                num = curNode.getParent(index).getLabel().getRotationNo();
                /* if can retract flow through current parent return the negation of it's
                   index and mark as visited. We return the negation of the index to
                   indicate that we need to move backwards along the edge */
                if (curNode.canRetractFlow(index) && !visited[num])
                {
                    visited[num] = true;
                    return (-(index + 1));
                }
            }

            index++;
        }
        // Otherwise there are no more paths
        if (curNode.isSource())
            pathExists = false;
        return notFound;

    }
```

Figure A.4: Code Listing for Finding Next Node

A.4 Generating a Random Preference List

Below is the code listing showing how we generated a random preference list for a person in our program:

```
// generate a random preference list given a list of people
public void generatePrefList(Person[] personList)
{

    int listLength = personList.length;
    int[] randList = new int[listLength];
    int randomNo;
    int temp;

    for (int i = 0; i < listLength; i++)
        randList[i] = i;


    for (int i = 0; i < listLength; i++)
    {
        temp = randList[i];
        randomNo = (int)(Math.random()*listLength);
        randList[i] = randList[randomNo];
        randList[randomNo] = temp;
    }

    // We generate a random order and then add it to our preference list
    for (int i = 0; i < listLength; i++)
        prefList[i] = personList[randList[i]];

}
```

Figure A.5: Code Listing for Generating a Random Preference List

Appendix B

Important Alogrithms

B.1 Algorithm A

Below is Gusfield's[1987] description of Algorithm A taken from his "Three Fast Algorithms for Four Problems in Stable Marriage" paper.

ALGORITHM A

0. Set $i = 0$; find and output the man optimal marriage M_0; find the woman optimal marriage M_t. All women are *unmarked* at this point.

1. If $M_i = M_t$ then stop. Else unmark any marked women and let m be the first man (in the fixed ordering of men) whose mate in M_i is different from his mate in M_t.

2. Set M to M_i. Let w be m's mate in M. Mark w and initiate breakmarriage (M, m); carry out (or continue, if returning from 3d) the sequence of proposals, acceptances and rejections as determined by breakmarriage (M, m), with the following modifications:

(a) When any unmarked woman *accepts a* proposal, mark that woman.

(b) Whenever a *marked* woman, say w' (which could be w), *receives* a proposal from a man, say m', whom she prefers to her mate in M_i, go to PAUSE breakmarriage (M, m).

Note that the comparison here is to the mate of w' in M_i, not to whom she is presently engaged to. Note also that the pause comes before w' decides to accept or reject the proposal from m'.

3. PAUSE: When breakmarriage (M, m) pauses do:

3a. Let $R(W)$ be the set containing w' and all the women who were marked since the most recent time that w' became marked; let $R(M)$ be the mates of $R(W)$ in M_i; and let *p-cycle* R_i be the ordered set of pairs consisting of $R(W)$ and their respective mates in $R(M)$, where the pairs are ordered in the order that the women in $R(W)$ were most recently marked.

3b. Let M_{i+1} be the marriage where w' is paired with m', where every other woman in $R(W)$ is paired to the man she is currently engaged to (as given in the current status of breakmarriage (M, m)), and all other women are paired to their mates in M_i. Output the p-cycle R_i and marriage M_{i+1}; set $i = i + 1$ and unmark all women in $R(W)$.

3c. If $w = w'$, then let w' accept the proposal from m' (which completes breakmarriage (M, m)), and go to step 1.

3d. If $w \neq w'$, then let w' accept or reject the proposal from m' (by comparing m' to the man she is currently engaged to in the current status of breakmarriage (M, m)).

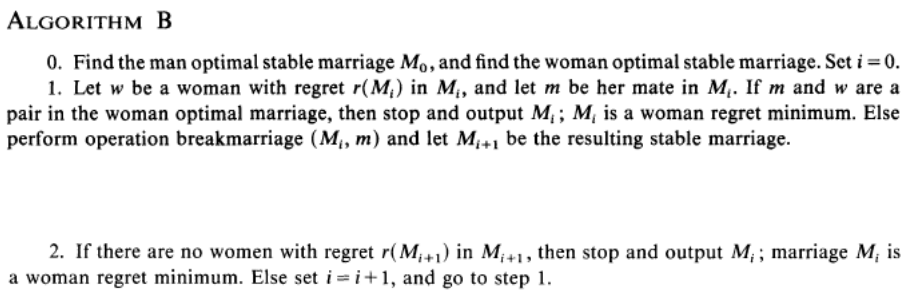
If w' rejects the proposal from m', then mark w'; return to step 2 and continue with breakmarriage (M, m) (i.e. m' next proposes to the woman on his list below w').

If w' accepts the proposal from m', then leave w' unmarked; go to step 2 and continue with breakmarriage (M, m) (i.e. the man who was engaged to w' when m' made his proposal is now free and makes the next proposal).

Figure B.1: Gusfield's Algorithm A

B.2 Algorithm B

Below is Gusfield's[1987] description of Algorithm B taken from his "Three Fast Algorithms for Four Problems in Stable Marriage" paper.



ALGORITHM B

0. Find the man optimal stable marriage M_0, and find the woman optimal stable marriage. Set $i = 0$.

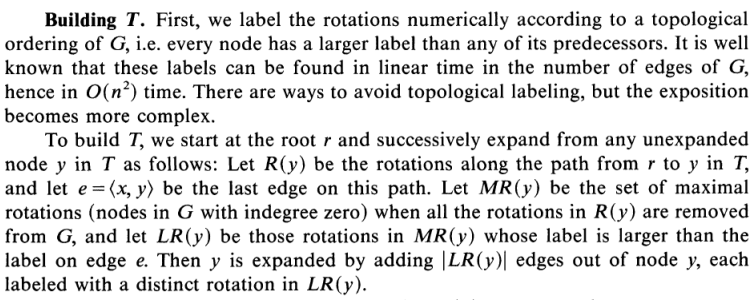
1. Let w be a woman with regret $r(M_i)$ in M_i, and let m be her mate in M_i. If m and w are a pair in the woman optimal marriage, then stop and output M_i; M_i is a woman regret minimum. Else perform operation breakmarriage (M_i, m) and let M_{i+1} be the resulting stable marriage.

2. If there are no women with regret $r(M_{i+1})$ in M_{i+1}, then stop and output M_i; marriage M_i is a woman regret minimum. Else set $i = i+1$, and go to step 1.

Figure B.2: Gusfield's Algorithm B

B.3 Enumeration Tree Construction

Below is Gusfield's[1987] description of the algorithm to construct the enumeration tree taken from his "Three Fast Algorithms for Four Problems in Stable Marriage" paper.



Building T. First, we label the rotations numerically according to a topological ordering of G, i.e. every node has a larger label than any of its predecessors. It is well known that these labels can be found in linear time in the number of edges of G, hence in $O(n^2)$ time. There are ways to avoid topological labeling, but the exposition becomes more complex.

To build T, we start at the root r and successively expand from any unexpanded node y in T as follows: Let $R(y)$ be the rotations along the path from r to y in T, and let $e = \langle x, y \rangle$ be the last edge on this path. Let $MR(y)$ be the set of maximal rotations (nodes in G with indegree zero) when all the rotations in $R(y)$ are removed from G, and let $LR(y)$ be those rotations in $MR(y)$ whose label is larger than the label on edge e. Then y is expanded by adding $|LR(y)|$ edges out of node y, each labeled with a distinct rotation in $LR(y)$.

Figure B.3: Gusfield's Enumeration Tree Construction

Appendix C

Test Runs

Here is one of the test runs we performed to see that our program was working correctly. For the full testing please see the appendix folder attached as part of the electronic submission.

C.1 Test Run 1

Here is an example taken from Gusfield's[1987] paper "Three Fast Algorithms for Four Problems in Stable Marriage".

Men	Men's preference lists	Women	Women's preference lists
1	5,7,1,2,6,8,4,3	1	5,3,7,6,1,2,8,4
2	2,3,7,5,4,1,8,6	2	8,6,3,5,7,2,1,4
3	8,5,1,4,6,2,3,7	3	1,5,6,2,4,8,7,3
4	3,2,7,4,1,6,8,5	4	8,7,3,2,4,1,5,6
5	7,2,5,1,3,6,8,4	5	6,4,7,3,8,1,2,5
6	1,6,7,5,8,4,2,3	6	2,8,5,4,6,3,7,1
7	2,5,7,6,3,4,8,1	7	7,5,2,1,8,6,4,3
8	3,8,4,5,7,2,6,1	8	7,4,1,5,2,3,6,8

Rotations
{(1, 5), (3, 8)}
{(1, 8), (2, 3), (4, 6)}
{(3,5), (6,1)}
{(5, 7), (7, 2)}
{(3, 1), (5, 2)}

No. of Stable Marriages
9

Male Opt. Marriage Social Cost
48

Male Opt. Marriage Max. Regret
6

Egalitarian Marriage Social Cost
48

Egalitarian Marriage Max. Regret
6

Min. Regret Marriage Social Cost
48

Min. Regret Marriage Max. Regret
6

Figure C.1: Example Problem with Expected Outcomes

We will now give the actual outcomes from our program.

C.1.1 Outputs

Male/ Female Optimal Marriage Output

```
MAN OPTIMAL
W   M
1   6
2   7
3   2
4   8
5   1
6   4
7   5
8   3
isStable = true

Social Cost: 48
Max Regret: 6
```

```
WOMAN OPTIMAL
W   M
1   5
2   3
3   1
4   8
5   6
6   2
7   7
8   4
isStable = true

Social Cost: 54
Max Regret: 8
```

Figure C.2: Man and Woman Optimal Output

Rotations Output

```
ROTATIONS
π0: { ( 1, 5 ), ( 3, 8 ),  }
π1: { ( 6, 1 ), ( 3, 5 ),  }
π2: { ( 7, 2 ), ( 5, 7 ),  }
π3: { ( 3, 1 ), ( 5, 2 ),  }
π4: { ( 2, 3 ), ( 4, 6 ), ( 1, 8 ),  }
```

Figure C.3: Rotations Output

Minimum Regret Marriage Output

```
Minimum Regret Stable Marriage
W   M
1   6
2   7
3   2
4   8
5   1
6   4
7   5
8   3
isStable = true

Social Cost: 48
Max Regret: 6
```

Figure C.4: Minimum Regret Stable Marriage Output

Egalitarian Marriage Output

```
Egalitarian Optimal Stable Marriage
W   M
1   3
2   7
3   2
4   8
5   6
6   4
7   5
8   1
isStable = true

Social Cost: 48
Max Regret: 6
```

Figure C.5: Egalitarian Stable Marriage Output

Enumeration Output
See fig 4.2 for the Enumeration tree for this example.

Appendix D

Original design

Here we include some important elements of our original design document. For the full original design document please see the appendix folder attached as part of the electronic submission.

D.1 Gale and Shapley Algorithm

Here is the psuedo-code for the Gale and Shapley algorithm taken from our original design:

```
1.  Person[][] galeShapley(MarriageProblem prob)
2.  {
3.          Person[][] stableMarriages = new Person[noOfWomen][2];
4.          Male[] men = prob.getMen();
5.          Female[] women = prob.getWomen();
6.          Female[] malePref = new Female[noOfMen];
7.
8.          while (!allHaveProposal(women))
9.          {
10.                 for each man i∈men do
11.                 {
12.                         if (!men[i].isEngaged())
13.                         {
14.                                 malePref[i] = men[i].getNextPref();
15.                                 pref[i].considerProposal(men[i]);
16.                         }
17.                 }
18.         }
19.
20.         for each woman i∈women do
21.         {
22.                 stableMarriages[i][0] = women[i].getCurrProposer();
23.                 stableMarriages[i][1] = women[i];
24.         }
25.         return stableMarriages;
26. }
```

Figure D.1: Gale-Shapley Algorithm Psuedo Code

D.2 Original UML Class Diagram

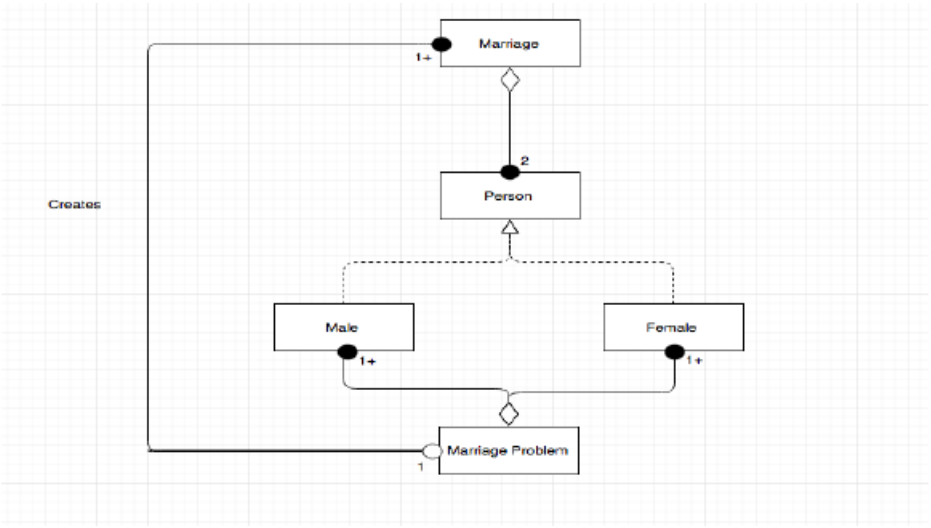


Figure D.2: Original Design for Classes

D.3 Break Marriage

Here is the pseudo code for Gusfield's[1987] break marriage method taken from our original design:

```
1.  Person[][] breakMarriage(Person[][] M, Male m)
2.  {
3.          Male freeMan = m;
4.          Female w = m.getSpouse();
5.
6.          while(freeMan.hasNextPref() and !w.preferSpouse(m))
7.          {
8.                  Female pref = freeMan.getNextPref();
9.                  Male currSpouse = M[pref.personNo() - 1][0];
10.                 pref.considerProposal(freeMan);
11.
12.                 if (pref.isSpouse(freeMan))
13.                 {
14.                         M[pref.personNo() - 1][0] = freeMan;
15.                         w.changeSpouse(currSpouse);
16.                         M[w.personNo() - 1][0] = currSpouse;
17.                         freeMan = currSpouse;
18.                 }
19.         }
20.         return M;
21. }
```

Figure D.3: Break Marriage Method Pseudo Code

Our program was ran using Java Version 8.0 (build 1.8.0_111 – $b14$).

Appendix E

Appendix Folder

Please see the appendix folder submitted as part of the electronic submission for the rest of our appendix. This includes:

- Average outputs for Evaluation
- Declaration of Academic Integrity
- Evaluation Graphs (includes graphs not included in our dissertation)
- SPSS Evaluation data
- Original Design Document
- Software Report Document
- Source Code
- Full Software Testing
- Full Software Testing Outputs
- User Guide to Installation