A JAVA Program for the Gale-Shapley Algorithm

Dr. Lisa J. Evered

And

William May

Department of Mathematics
Iona College

A JAVA Program for the Gale-Shapley Algorithm

The Gale-Shapley algorithm was developed to pair men and women who had expressed their individual preferences about one another. Their pairings should result in stable marriages. A marriage is considered stable if no spouse is motivated to select another. Perhaps the problem is not realistic but it has attracted considerable mathematical interest. If there are 100 men and 100 women, each of who have ranked those of the opposite gender from 1 to 100, the Gale-Shapley algorithm will indeed match 100 couples in stable marriages. With so many men and women the algorithm is lengthy and tedious. The following example pairs four men with four women. The first number of each entry gives the ranking of the women by the men. The second number in the entry is the ranking of the men by the women.

	Michelle	Teresa	Kim	Ashley
Brent	1,3	2,2	3,1	4,3
Mike	1,4	2,3	3,2	4,4
Jason	3,1	1,4	2,3	4,2
Dennis	2,2	3,1	1,4	4,1

The Gale-Shapley procedure uses these rankings to determine stable pairings. First, assign each man to the woman he most prefers. If two or more men ask the same woman, she must choose whom she prefers. The other men then ask the next woman on their lists. After the first application of the Gale-Shapley procedure, the pairs look like this:

Brent, Michelle
Mike, Michelle
Jason, Teresa
Dennis, Kim

Since both Brent and Mike prefer Michelle, Michelle must choose between them. Notice that Michelle's first two choices, Jason and Dennis, are temporarily taken and she must chose her third choice, Brent. Mike's second choice is Teresa. After Round 2, the pairs look like this:

Brent, Michelle
Mike, Teresa
Jason, Teresa
Dennis, Kim

Now both Mike and Jason prefer Teresa, so Teresa must choose between them. Teresa's first and second choices are temporarily taken so she chooses here third choice, Mike. Jason picks his second choice, Kim. After Round 3, the pairs look like this:

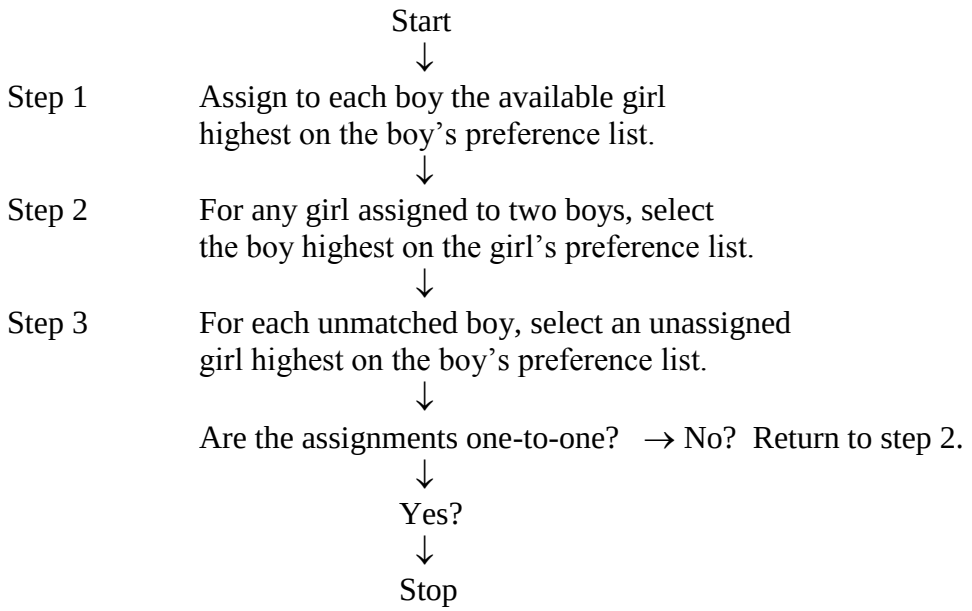
Brent, Michelle
Mike, Teresa
Jason, Kim
Dennis, Kim

This process should be continued until stable pairings are obtained. In all, ten Gale-Shapley iterations are needed. Can you show that the final pairings are as follows?

Brent, Kim
Mike, Ashley
Jason, Michelle
Dennis, Teresa

The algorithms' complexity is polynomial -- for n individuals the complexity is $\geq n^2$. Clearly for n larger than, say ten, the algorithm becomes impracticable for most who may wish to use it. A computer program that implements the Gale-Shapley algorithm electronically is needed (see appendix 1). Design of this algorithm and its application in simple cases is an interesting project for computer science students even at the high school level.

The flow chart for the Gale-Shapley algorithm will look something like the following:



The computer language that my college student preferred to use is JAVA. While other programs are more sophisticated programmatically, the program produced here is faithful to the algorithm and when applied to the 4x4 marriage problem above, arrives at stable pairings quickly and correctly. It should be noted that stable pairings are not necessarily unique; however the algorithm given produces a single pairing for each preference matrix provided.

The stable marriage problem has spawned considerable variations. It would be unreasonable for 100 women to rank 100 available men in serial order. A more realistic ordering might be to rank the three highest preferences, 1, 2, and 3 and the three lowest preferences, 98, 99, and 100 with no ranking attempted among the ninety-four remaining males. Could the Gale-Shapley algorithm be altered to accommodate rankings of this kind?

A more interesting variation asks if it is possible to enhance an individual's spousal assignment by submitting specious preferences. This possibility is the subject of contemporary research in game theory. Results indicate that perhaps honesty may not be the best policy when confronted by a computer algorithm that makes choices on our behalf.

REFERENCES

Gale, D., and L.S. Shapley. 1962. "College Admissions and the Stability of Marriage." *American Mathematical Monthly* 69: 9-14.

Roth, Alvin. 2002. "The Economist As Engineer: Game Theory, Experimentation, and Computation As Tools For Design Economists. *Econometrica* 70, no.4: 1341-1378.

Teo, C., J. Sethuraman, and W. Tan. "Gale-Shapley Stable Marriage Problem Revisited: Strategic Issues and Applications." *Management Science* 47, no. 9 (2001): 1252-1267.

```
// StableMarriage.java
// William May
// March 14, 2011

import java.util.*;

public class StableMarriage {
        public static void main(String[] args) {
                System.out.print("How many pairs would you like to
match up? ");
                Scanner input = new Scanner (System.in);
                int NUMBER_OF_COUPLES = input.nextInt();
System.out.println();
                LinkedList eligibleMen =
createEligibleMen(NUMBER_OF_COUPLES);
System.out.println();
                LinkedList eligibleWomen =
createEligibleWomen(NUMBER_OF_COUPLES);        System.out.println();
                createPreferences(eligibleMen, eligibleWomen);
                createPreferences(eligibleWomen, eligibleMen);
                SocialRegister sr = new
SocialRegister(eligibleMen,eligibleWomen);
                System.out.println(sr);
                while (sr.eligibleMenExist()) {
                        sr.getFirstEligible().makeProposal();
                        System.out.println(sr);
                }
        }

        static LinkedList createEligibleMen(int number) {
                System.out.println("What are the names of the men?
(Press Enter to submit a name)");
                LinkedList men = new LinkedList();
                Scanner input = new Scanner (System.in);
                for (int i = 1; i <= number; i++)
                        men.add(new Man(input.next()));
                return men;
        }

        static LinkedList createEligibleWomen(int number) {
                System.out.println("What are the names of the women?
(Press Enter to submit a
name)");
                LinkedList women = new LinkedList();
                Scanner input = new Scanner (System.in);
                for (int i = 1; i <= number; i++)
                        women.add(new Woman(input.next()));
                return women;
        }

        static void createPreferences(LinkedList a, LinkedList b) {
                Iterator it = a.listIterator();
```

```
                while (it.hasNext()) {
                        Person p = (Person) it.next();
                        Rankings r = p.getRankings();
                        r.addAll(b);
                        System.out.println("What are the preferences of
" + p.name + "? (Input numeric
position of persons)");
                        Scanner input = new Scanner (System.in);
                        int n = r.size();
                        for (int i = 0; i < n; i++) {
                                int loc = input.nextInt() - 1;
                                r.add(r.get(loc)); // put it at the
back
                        }
                        for (int i = 0; i < n; i++)
                                r.removeFirst();
                }
        }
}

class SocialRegister {
        public static SocialRegister defaultRegister; // singleton
pattern
        LinkedList eligibleMen, women;
        HashMap engagements; // maps Women to Men
        SocialRegister(LinkedList eligibleMen, LinkedList
eligibleWomen) {
                defaultRegister = this;
                this.eligibleMen = eligibleMen;
                women = eligibleWomen;
                engagements = new HashMap();
        }
        boolean eligibleMenExist() {
                return !eligibleMen.isEmpty();
        }
        Man getFirstEligible() {
                return (Man) eligibleMen.get(0);
        }
        void createEngagement(Woman w, Man m) {
                if (engagements.containsKey(w))
                eligibleMen.add(engagements.get(w));
                engagements.put(w,m);
                eligibleMen.remove(m);
        }

        public String toString() {
                String result = "";
                if (eligibleMenExist()) {
                        result += "\n ELIGIBLE MEN: \n" +
showList(eligibleMen);
                        result += " WOMEN: \n" + showList(women);
                        result += " ENGAGEMENTS: \n";
                } else {
                        result += " MARRIAGES: \n";
                }
                result += showEngagements();
                return result;
```

```
        }

        String showList(LinkedList eligible) {
                String result = "";
                Iterator it = eligible.listIterator();
                while (it.hasNext()) {
                        Person p = (Person) it.next();
                        result += "    " + p + ":" + p.getRankings() +
"\n";
                }
                return result;
        }

        String showEngagements() {
                String result = "";
                Set couples = engagements.entrySet(); // get the set of
couples
                Iterator it = couples.iterator();
                while (it.hasNext()) {
                        Map.Entry couple = (Map.Entry) it.next();
                        result += "   (" + couple.getKey() + "," +
couple.getValue() + ")\n";
                }
                return result;
        }
}

class Rankings extends LinkedList {
        public void trim(Object x) {
                // removes x and all elements after it
                Iterator it = listIterator();
                boolean found = false;
                while (!found)
                if (it.next() == x) {
                        found = true;
                        it.remove();
                }
                while (it.hasNext()) {
                        it.next();
                        it.remove();
                }
        }

        public String toString() {
                String result = "";
                Iterator it = listIterator();
                while (it.hasNext())
                result += " " + it.next();
                return result;
        }
}

class Person {

        String name;
        Rankings preferences;
```

```
        Person(String name) {
                this.name = name;
                preferences = new Rankings();
        }

        Rankings getRankings() {
                return preferences;
        }

        public String toString() {
                return name;
        }
}

class Man extends Person {

        Man(String name) {
                super(name);
        }

        void makeProposal() {
                ((Woman)
preferences.removeFirst()).considerProposal(this);
        }
}

class Woman extends Person {

        Woman(String name) {
                super(name);
        }

        void considerProposal(Man m) {
                if (preferences.contains(m)) {

SocialRegister.defaultRegister.createEngagement(this,m);
                        preferences.trim(m);
                }
        }
}
```