A Look into Sex-Equality and Other Aspects of Stable Matchings

Kathryn J. Kent
Western Maryland College

with Significant Contributions by
Erik Winfree
University of Chicago

REU Summer Program
1990

A Look into Sex-Equality and Other Aspects of Stable Matchings

I. Abstract

In this paper we hope to outline the basic Gale-Shapely algorithm for finding stable matchings and make apparent the need for an algorithm for finding a sex-equal stable matching. Sex-equality will be defined, and different weightings proposed to most efficiently find such matchings. We propose one possible algorithm that is effective in many cases, and illustrate it with a computer program. Lastly, we will take a look at "stable marriages" for more than two "sexes". The basic source book that we have used is The Stable Marriage Problem - Structure and Algorithms by Dan Gusfield and Robert W. Irving, which seems to be a comprehensive, thorough, and up-to-date overview of what is widely (and not so widely) known about stable marriages. In order to make things less confusing, we often alter the notation used by Gusfield and Irving.

II. Introduction

To begin, we will have to go over a few definitions and examples. For our purposes, a **matching** is a one-to-one, onto mapping from one group, or sex, to another. In the cases we will be looking at, there are some n "men" and n "women" (n a natural number) that have certain lists denoting which members of the opposite sex they prefer first, second, etc., as illustrated below. The men are denoted by letters, and the women by numbers. In this case, n=4.

Men's preferences				
a	2	3	1	4
b	1	2	3	4
c	3	4	2	1
d	1	3	4	2

Women's preferences				
1	a	d	b	c
2	d	b	c	a
3	b	c	d	a
4	c	d	a	b

To interpret the tables, man "a" prefers woman "2" to all of the others, and prefers woman "4" the least. An example of a matching, then, is the four **pairs** a-2, b-1, c-3, d-4 which is clearly one-to-one and onto.

A **stable matching**, whose definition is "a matching that is not unstable", is a matching in which there are no two people who would strictly prefer being together to being with their current partner. An matching is unstable if there exists such a pair.

The example above is unstable, because "d" would rather be with "1" than with "4", and "1" would rather be with "d" than with "b". The pair d-1, then, is called a **blocking pair**, or a pair which makes the matching unstable. A pair blocks only if BOTH parties prefer each other to their current partner. Again looking at the example, "3" prefers "b" to her current partner "c", but "b" would rather stay with his partner, "1", than switch to "3", and thus b-3 does not block the matching.

To check for instability, one need only look at the preference list of one sex, say the men, and then look only at the women that each man prefers to the person he is currently matched with. In our example matching (current partners bold shadowed)

Men's preferences					Women's preferences				
a	**2**	3	1	4	1	a	d	**b**	c
b	**1**	2	3	4	2	d	b	c	a
c	**3**	4	2	1	3	b	c	d	a
d	1	**3**	4	2	4	c	d	a	b

one would only need to check d-1 and d-3, because none of the other men would be willing to switch, seeing as they prefer their current partners to any of the others. Checking d-1 (outlined), we find that they both prefer each other to their partners. At this point we would be done, having found a blocking pair. To illustrate further, though, let's check d-3 (underlined). It is quite clear that "3" prefers her partner to "d", so this pair does not block. From here on we will general use the terms "matching" and "stable matching" interchangeably.

III. The Gale-Shapley algorithm

In 1962, Gale and Shapley introduced their algorithm for finding a stable matching. They proved that there is always at least one stable matching, and that their algorithm will always terminate in a stable matching. It is easiest to understand the algorithm if one sees it in action, so rather than simply explain it, we will execute it on the first example.

Men's preferences					Women's preferences				
a	2	3	1	4	1	a	d	b	c
b	1	2	3	4	2	d	b	c	a
c	3	4	2	1	3	b	c	d	a
d	1	3	4	2	4	c	d	a	b

Begin by pairing "a" with the first woman on his preference list, "2". Next, pair "b" with the first woman on his preference list, "1". Continue in this manner, pairing c-3. One would continue to match d-1, but "1" is already matched, so first we must decide who she prefers. Looking at her preference list, she prefers "d" to "b", so she breaks her pairing with "b" in favor of "d". Let's see where we are so far:

Men's preferences					Women's preferences				
a	2	3	1	4	1	a	d	b	c
b	1	2	3	4	2	d	b	c	a
c	3	4	2	1	3	b	c	d	a
d	1	3	4	2	4	c	d	a	b

So, "b" is free, and we try to match him with the next woman on his preference list, since the first one rejected him. Woman "2" is already paired, but she prefers "b" to her current partner "a". So we break the pairing of a-2 and then pair b-2 together.

Men's preferences					Women's preferences				
a	2	3	1	4	1	a	d	b	c
b	1	2	3	4	2	d	b	c	a
c	3	4	2	1	3	b	c	d	a
d	1	3	4	2	4	c	d	a	b

Looking at the next woman on "a"'s list, we find that "3" would rather stay with "c" than break her match in order to pair with "a", so "a" tries the next woman on his list. "1" prefers "a" to "d", so we make a-1 a pair, and "d" is once again free.

Men's preferences					Women's preferences				
a	2	3	1	4	1	a	d	b	c
b	1	2	3	4	2	d	b	c	a
c	3	4	2	1	3	b	c	d	a
d	1	3	4	2	4	c	d	a	b

He tries "3", but she again prefers her present partner, so "d" tries "4". Since she is not matched with anyone, we can immediately pair her with "d", since being paired with anyone on the list is considered preferable to not being paired at all. So we end up with the final matching a-1, b-2, c-3, d-4.

Men's preferences					Women's preferences				
a	2	3	1	4	1	a	d	b	c
b	1	2	3	4	2	d	b	c	a
c	3	4	2	1	3	b	c	d	a
d	1	3	4	2	4	c	d	a	b

If one checks for stability, it is quite evident that all of the pairs that must be checked are ones that have already been tried during the execution of the algorithm. If we look at a-2, for example, we know that it does not block the matching, because "2" already rejected "a" for someone she prefered more.

There are some interesting properties that result from the performance of the Gale-Shapley algorithm. As it turns out, the matching that results from going through with the men's preferences first, culminates with each man having the best possible match he can have in any stable matching, and each woman having her worst. This is called the **man-optimal** stable matching. Likewise, obviously, if you start the algorithm looking at the women's preferences, the women get their best matches, and the men their worst, thus resulting in the **woman-optimal** stable matching. One could see, as the algorithm progressed through the example, that the men were going lower down on

their preference lists, and the women were simultaneously getting higher.

Gusfield and Irving also illustrate what they call the extended algorithm, in which the preference lists are reduced as the algorithm is executed, eliminating pairs that would not be stable, no matter what the matching. We will once again use the original example

Men's preferences					Women's preferences				
a	2	3	1	4	1	a	d	b	c
b	1	2	3	4	2	d	b	c	a
c	3	4	2	1	3	b	c	d	a
d	1	3	4	2	4	c	d	a	b

We will go through in steps:

1) pair a-2 and eliminate any pairings that occur after "a" on "2"'s preference list (in this case there aren't any)

2) pair b-1 and eliminate any pairings that occur after "b" on "1"'s preference list (c-1) so the lists now look like this:

Men's preferences					Women's preferences				
a	2	3	1	4	1	a	d	b	
b	1	2	3	4	2	d	b	c	a
c	3	4	2		3	b	c	d	a
d	1	3	4	2	4	c	d	a	b

The idea is to delete "c" from "1"'s list and "1" from "c"'s list because it would be useless for "c" to propose, since "1" would automatically reject him.

3) pair c-3 and eliminate d-3 and a-3, and the lists are:

Men's preferences					Women's preferences				
a	2		1	4	1	a	d	b	
b	1	2	3	4	2	d	b	c	a
c	3	4	2		3	b	c		
d	1		4	2	4	c	d	a	b

4) pair d-1 (note: if "1" is still left on "d"'s list, then "1" automatically prefers "d" to her present partner, or else the pair would have been eliminated already.) eliminate b-1

5) pair b-2 eliminate c-2 and a-2

Men's preferences					Women's preferences				
a			1	4	1	a	d		
b		2	3	4	2	d	b		
c	3	4			3	b	c		
d	1		4	2	4	c	d	a	b

6) pair a-1 eliminate d-1

7) pair d-4 eliminate a-4 and b-4

We are now done, and have the following lists left:

Men's preferences					Women's preferences				
a			1		1	a			
b		2	3		2	d	b		
c	3	4			3	b	c		
d			4	2	4	c	d		

It is quite clear that in this case the extended algorithm speeds up the process (not having to try pairings that won't work), and makes the final lists quite a bit smaller.

The extended algorithm is continued, so that it is performed again, this time with the women choosing first on their lists. For this purpose, we will switch their positions on the page.

Women's preferences				Men's preferences			
1	a			a	1		
2	d	b		b	2	3	
3	b	c		c	3	4	
4	c	d		d	4	2	

So we start by pairing 1-a, then 2-d, 3-b, and 4-c, eliminating 2-c from the lists. We have now found the woman-optimal stable matching. What is left of the preference lists is refered to as the **GS-lists**, which, as it turns out, contain all of the possible stable matchings. Note that in this case, a-1 is the only possible pairing for either "a" or "1". They are called a **fixed pair**. The

first matching on the men's G-S lists is the man-optimal, and the last is the woman-optimal.

In this particular case, there are only two stable matchings. If the man-optimal and woman-optimal matchings had been the same, there would only the one stable matching would have existed. On the other hand, some preference lists are such that no pair is eliminated, and the GS-lists are the same as the complete preference lists.

IV. Weight functions

It is quite evident that although the Gale-Shapley algorithm works, it is almost always to the advantage of one sex over the other. One way of looking at this phenomenon is to assign weights to each matching. The most obvious way of doing this is through a **direct weighting.** If a man gets matched with the first woman on his list, the weight for that man is one, if they get the second, the weight is two, and so on. The same is true for the women. You then take the sum of these weights to get the **total weight.** We shall define **sex-equal** as the particular matching or matchings (there may be a tie) that are closest to having or have the sum of the weights for just the men equal to that of just the women.

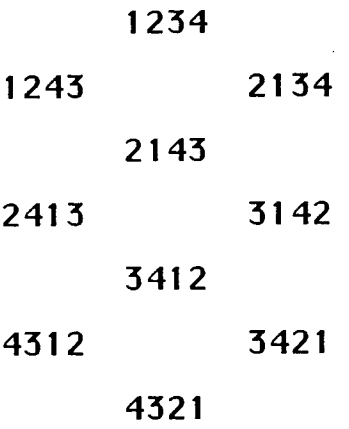
In the example we used earlier, if the sum of the weights for just the men is compared to that of just the women, it is clear that the man-optimal matching (9 vs 7) is more equal than the woman-optimal one (12 vs 4). Looking at the total weight, though, discloses nothing, because the total weight for both of these is 16.

We will now introduce another example, again with n=4.

Men's preferences					Women's preferences				
a	1	2	3	4	1	d	c	b	a
b	2	1	4	3	2	c	d	a	b
c	3	4	1	2	3	b	a	d	c
d	4	3	2	1	4	a	b	c	d

We shall refer to this as the **total lattice** example for n=4, because it produces the highest number of stable matchings for n=4, which is 10 matchings, that form a full lattice using the relation of men prefering one matching to another. In other

words, a matching is higher on the lattice than some other matching if the sum of the weights for the men is less than that for the other matching. If it is the same, then the matchings are on the same level. In the lattice, we list only the women's numbers, in the order such that the first listed is paired to man "a", the second to man "b", the third to "c", and the fourth to "d". The lattice is:



where the men have the lowest sum of weights from a-1, b-2, c-3, d-4 (man-optimal sum of weights is 1+1+1+1=4) and the highest sum of weights from a-4, b-3, c-2, d-1 (woman-optimal sum of weights is 4+4+4+4=16). The women's sum of weights is the exact reverse. On the center of the lattice the men's weights equal the women's (both are 10). The result of this symmetry is that the **total** sum of weights for all of these matchings is the same (20). It is quite evident that using the total sum of weights reveals very little.

The next attempt at a better weight function was to square the individual weights, and then sum them. In this case, the male-optimal **sum of squares** would be 1+1+1+1+ 16+16+16+16=68, the next level down 60, the next, 52, the center would be 52, and the rest increase again symmetrically. Apparently, this at least differentiates the most unequal matchings from some of the more equal ones. The next step then would be trying the sum of the cubes of the weights, right? Well, it sounds good theoretically, but let's look at what happens. If we look towards the center of the lattice, say 2143 and 3142, the matchings are a-2, b-1, c-4, d-3, and a-3,b-1, c-4, d-2. These are easier to see if we go back to the preference lists.

	Men	weight	Women	weight
	a 1 2 3 4	2	1 d c b a	3
	b 2 1 4 3	2	2 c d a b	3
	c 3 4 1 2	2	3 b a d c	3
	d 4 3 2 1	2	4 a b c d	3
sum:		8		12

	a 1 2 3 4	3	1 d c b a	3
	b 2 1 4 3	2	2 c d a b	2
	c 3 4 1 2	2	3 b a d c	2
	d 4 3 2 1	3	4 a b c d	3
sum:		10		10

It is quite obvious now that no matter what power one takes of the weights, their sums will be the same, because even though the second is clearly sex-equal, and the first isn't, they still have the same numbers for their weights. So the sum of any powers of the original weights will still be equal.

So we suppose that if we happen to find one matching with a total weight, or sum of squares, that is strictly less than all of the others, then it is the sex-equal one. Unfortunately, this is not true. A counter example is:

		weights	squares		weights	squares
	a 1 4 3 2	1	1	1 a d c b	1	1
	b 3 4 1 2	2	4	2 b a d c	3	9
	c 4 1 2 3	4	16	3 d c a b	2	4
	d 2 1 4 3	1	1	4 a d b c	3	9
sums:		8	22		9	23

total sum of weights = 17 total sum of squares = 55

	a 1 4 3 2	1	1	1 a d c b	1	1
	b 3 4 1 2	1	1	2 b a d c	3	9
	c 4 1 2 3	1	1	3 d c a b	4	16
	d 2 1 4 3	1	1	4 a d b c	4	16
sums:		4	4		12	42

total sum of weights = 16 total sum of squares = 46

In both the sum of weights and the sum of squares, the second matching had the lower totals. It is quite apparent, though, that the first is the more sex-equal of the two matchings, therefore, the conjectures are false.

The next step is to try a different weight function. Since the sum of powers did not reveal anything, let's try the power of a sum.

If we take the square of the sum of the men's weights and add that to the square of the sum of the women's, does this tell us anything?

As a matter of fact, the **square of the sums** does indeed differentiate between these different matchings. Using the same examples, for the first, the square of the sums is 64+144=208, for the second, sex-equal one, it is 100+100=200. Lets look at this for the total lattice:

		sum of squares	square of sums
1234		68	272
1243	2134	60	232
2143		52	208
2413	3142	52	200
3412		52	208
4312	3421	60	232
4321		68	272

So the conjecture to make is:

If you find the matching(s) with the strictly lowest square of the sum of weights, then that is the sex-equal matching.

Proof: Take the sum of the weights of the men, a, and the sum of the weights of the women, b. I believe that without loss of generality we can say $a < b-1$. (If $a=b$ or $a=b-1$ then that is the sex-equal matching).

$a < b-1$	the original inequality
$a - b + 1 < 0$	move everything to left
$2(a - b + 1) < 0$	multiply both sides by 2
$2(a - b + 1) + a^2 + b^2 < a^2 + b^2$	add and subtract $a^2 + b^2$
$(a^2 + 2a + 1) + (b^2 - 2b + 1) < a^2 + b^2$	regroup

$(a + 1)^2 + (b - 1)^2 < a^2 + b^2$ write in terms of squares

So the square of the sum of the weights decrease as a and b get closer together (more equal), whether a is less than or greater than b. Therefore, if you find the lowest square of sums, it is the sex equal one.

V. The Algorithm and Program

In order to devise an algorithm to find sex-equal stable matchings using the properties of the different weight functions, it was necessary to write the preferences in a different form. Instead writing the preferences in the usual way, we transform them into a two valued or complex matrix, with each entry representing the position, or weight, of each individual on another's preference list. To illustrate, let's take the original example again:

Men's preferences					Women's preferences				
a	2	3	1	4	1	a	d	b	c
b	1	2	3	4	2	d	b	c	a
c	3	4	2	1	3	b	c	d	a
d	1	3	4	2	4	c	d	a	b

change this into a 4 x 4 matrix, with men along the rows, i , and women along the columns, j. The first ij entry is where woman j is on man i's preference list, and the second entry is where man i is on woman j's preference list. So in transforming the above lists, the first entry in position (a,1) is 3, since woman "1" is third on man "a"'s preference list. The second entry in (a,1) is then 1, because man "a" is first on woman "1"'s list. So filling out the matrix in this manner, we get:

	1	2	3	4
a	3 1	1 4	2 4	4 3
b	1 3	2 2	3 1	4 4
c	4 4	3 3	1 2	2 1
d	1 2	4 1	2 3	3 2

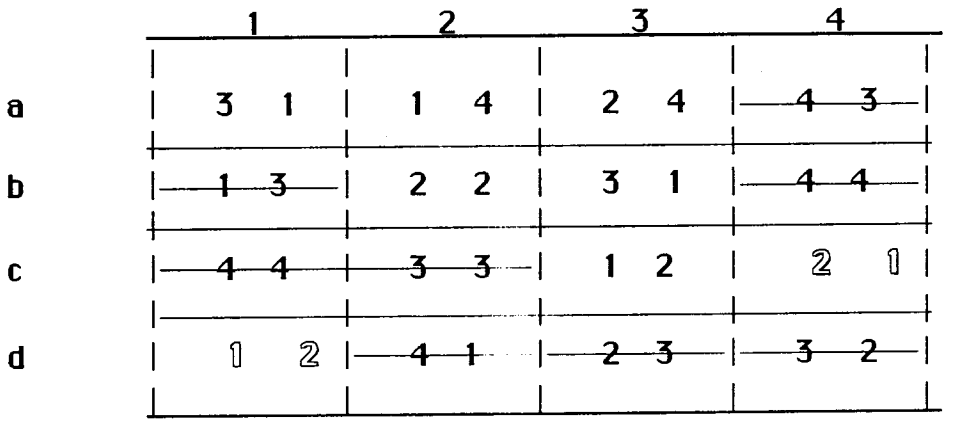
A third value can then be assigned to each double entry, namely one of the different weights that can be applied. In the case that we have worked with most, we applied the sum of squares weight, so that the value of each double entry is equal to the sum of the squares of the two individual entries. The third value for (a,1) in our example would then be $3^2 + 1^2 = 10$. The general idea of the algorithm is to then choose a matching (exactly one entry in any given row or column) using the preferences and the weights. The choices are made by determining which entries, when added on, contribute the least to the total sum of squares, while still maintaining a matching (no more than one pair highlighted in any given row or column) that only throws out the pairs which are prefered less than the chosen ones, thus ensuring stability when finished.

The first step in the algorithm is to find the/a pair with the lowest sum of squares, in this case, there are three, (c,3), (c,4), and (d,1). So we choose one of these randomly, say (c,4). Now, look across the "c" row, and cancel out any of the pairs that are less preferable according to "c". These are (c,1) and (c,2). Man "c" still prefers "3" to "4". Next, we look at the "4" column, and cross out all of the other pairs, because "4" prefers "c" to all of the others.

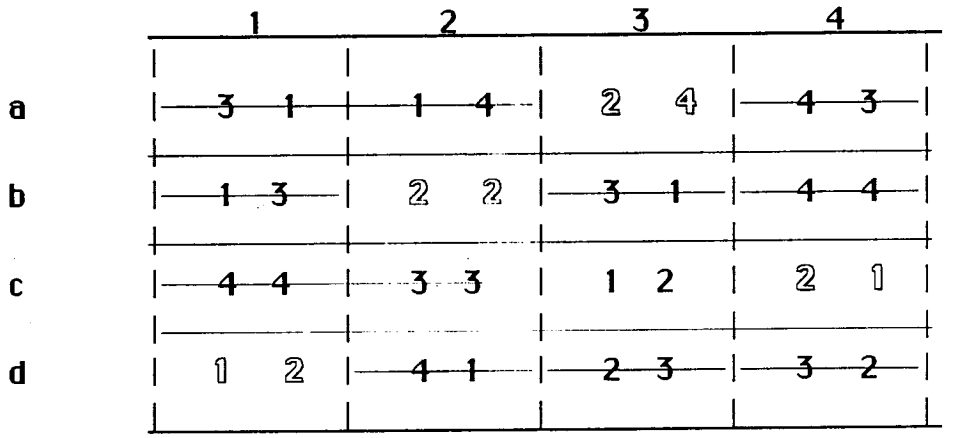
	1	2	3	4
a	3 1	1 4	2 4	~~4 3~~
b	1 3	2 2	3 1	~~4 4~~
c	~~4 4~~	~~3 3~~	1 2	2 1
d	1 2	4 1	2 3	~~3 2~~

Now, we choose another "open" square which contributes the least to the total weight function. The options are (c,3) and (d,1). Both of these are suitable, so we might as well choose one that does not rule out (c,4), our already existing pair. So we

choose (d,1), and strike out (d,2), (d,3), and (b,1). We would also cross out (d,4) and (c,1), but they are already crossed out.

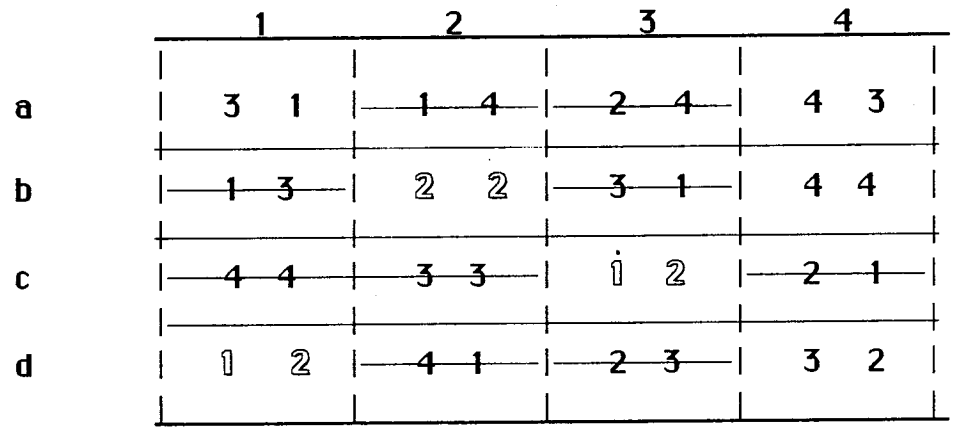


The next choice might be (c,3), but seeing as that does not lower the total sum of squares, we can avoid the contradiction of (c,3) vs (c,4) by choosing another pair, so we go on to choose (b,2).
This gets rid of (b,3) and (a,2). The next choice would be (a,3), crossing out (a,1).

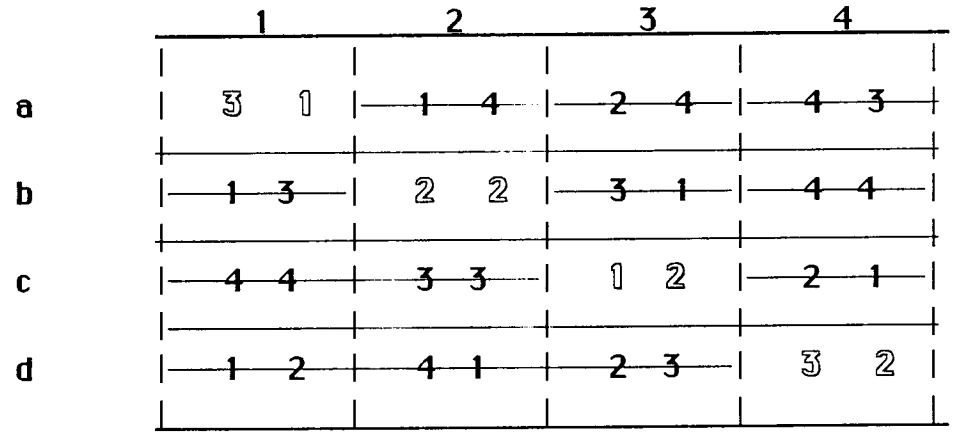


This would be our desired matching, but we still have one "open" pair, (c,3), which makes the matching unstable, seeing as (c,3) would rather be together than with their present partners. So we choose (c,3), and cross out (c,4), and (a,1). But we still

have to reopen (a,1), (a,4), (b,4), and (d,4) since (a,3) and (c,4) are no longer prefered to these other pairs. We then have:



The process continues in the same manner, until we end up with:



This is obviously stable, because we canceled out only those pairs that were mutually less prefered than the chosen ones, and since there are no open spaces, there are no pairs that could block this matching. As for sex-equality, we know that there are only two matchings for this example, (a,1), (b,2), (c,3), (d,4), which is what we have above, with a man to woman weight ratio of 9 to 8. The other matching is (a,1), (b,3), (c,4), (d,2) with a ratio of 12 to 4. Therefore, this IS the sex-equal matching.

We have included a copy of the program that runs this algorithm at the end of the paper, which has all of the details for those so inclined. We believe, but have not proved, that this

always terminates. We do know that if it does terminate, it does so with a stable matching.

The problem, as discussed earlier in the section on weight functions, is that the sum of squares does not always yield the sex-equal stable matching. In most cases, it does, but in the more structured ones, such as the full lattice, each execution may result in a different matching, not necessarily the sex-equal one. It appears, though, that in most cases, which are more random in terms of preference lists, the algorithm does find the sex-equal matching. The next step is to try altering the program so that the algorithm makes it's choices of which entries to include based on the effect the entry makes on the square of the sum, rather than the sum of the square. In that case, we know that we are more likely to get a sex-equal matching. This demands that we recalculate quite a bit every time we try to add a new pair. Better yet would be to alter it so that decisions are made solely on the contribution the entry makes to the DIFFERENCE in the weights of the men and the women, since this is what the definition of sex-equality hinges on. This, again, would need more calculations than the algorithm illustrated above. Unfortunately, this type of algorithm may require trying all of the possible stable matchings before making a choice, thus not improving on any already existing algorithms. These possibilities are yet to be probed.

VI. A quick peek at N-dimensional matchings

Our last inquiry into stable matchings involves a side trip from sex-equality. While studying stable matchings in general, a colleague, Stephanie Wukovitz, asked what would happen if we were on a planet where there were three sexes, and a "marriage" was formed by three people, one of each sex. I found this question extremely intriguing, and decided to look into it. To explain exactly what I am looking at, let's try an example. First, if working with three sexes, we need six preference lists, two for each sex, showing the order in which they prefer the other two sexes.

a 1 2 3	a A C B	A 2 3 1	A c a b	1 a b c	1 A C B
b 3 2 1	b B A C	B 1 3 2	B c b a	2 b c a	2 B A C
c 3 1 2	c B C A	C 2 1 3	C b a c	3 a c b	3 C A B

I chose to use only three individuals of each sex, because more people makes it more complicated. The first question that I looked at when trying to find a stable matching, is how to define stability in the first place. There are a number of possibilities. The first, a very strict definition, is to make threesomes, and then define as unstable any threesome in which there exists at least one pair that is not on the G-S lists of the two sexes involved. In this case, it is very difficult to find a stable matching, and in fact, one may not exist for all examples. This is true for the case above. Take it's G-S lists:

a 1	a A	A 2 3 1	A a	1 a	1 A C
b 2	b C	B 3 2	B c	2 b	2 B A
c 3	c B	C 1 3	C b	3 c	3 C A B

Looking at these, it is quite clear that there are many fixed pairs, and thus many pairs that must be together. Trying to make threesomes, we first pair up a-1, b-2, c-3, since these must be together because of the definition. Then, for the same reason, we must pair a-A, b-C, and c-B. So let's look at what this makes in terms of threesomes; a-1-A, b-2-C, and c-3-B. The pairing 1-A is all right, as is 3-B, but 2-C is not in the G-S lists, so the matching is unstable. One can see that with this definition, it may be very difficult, if not impossible, to find stable matchings.

A looser definition, then, is to say that a matching is unstable if there are any three individuals who would rather be with each other than the people they are already matched with. This concept seems to be a little harder to visualize, in that it is not quite clear how to check for stability, or how to come up with stable matchings. This seems to be an interesting path to follow, and I plan to continue looking into this question.

Acknowledgements

My primary source book has been The Stable Marriage Problem - Structure and Algorithms by Dan Gusfield and Robert W. Irving, 1989, MIT Press, Cambridge, Massachusetts.

I would like to give my sincere thanks to the entire staff of the REU summer program, in particular Jim Simpson and Mary Flahive, who gave me many suggestion, not all of which worked, but many of which led me on the right track.

```
/* Marg.c by Erik Winfree

Currently this can find stable marriages for up to 16 couples.
Change the definition of 'BIG' to a greater value if desired - but I don't
guarantee either the IBM will have enough memory or that the screen will
be big enough to display the matrix.

'A'...'Z' are considered the males. 'a'...'z' are considered the females.
In the matrix, the male preference is listed first; it operates on the row.
   yellow is unallocated and open (i.e. an engagement disruptive to the
          current partial matching)
   blue is unallocated and stable
   white is an allocated engagement.
 Thus if the matrix is just white and blue, we have a stable marriage.
In the preference list, for each letter,
 grey means we know nothing about this as a possible matching
 blue means that we know there exists a stable matching that includes this
 white indicates the most recently found matching
 - after finding the GS list -
 grey means the matching is impossible
 yellow means the matching is in the GS list (i.e. COULD be in a matching).

When entering the preference lists, CAPITALIZATION COUNTS!
If you simply press <enter> rather than entering a list for a particular
person, then a random preference list is chosen for that person.

When entering the a,b for the weight function, 0,0 requests a random
weight distribution. This usually causes the algorithm to get into a loop.

The algorithm is still deterministic - that is, when there are several
equally good choices, it chooses just the first one it gets to.

The numbers on the right side are the most recent sums of weights of the
pairs in the partial matching of 1,2,..,n couples, next to which pair
was chosen to get to that partial matching.
s= (sum male prefs)^2 + (sum of female prefs)^2
d= (sum female prefs) - (sum of male prefs).

If things die, just try again...
*/



#include <graphics.h>
#include <math.h>
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

#define BIG 16
int    n,match,showweights=0;
char   Mp[BIG][BIG],Wp[BIG][BIG];
int    Pm[BIG][BIG],Pw[BIG][BIG];
int    sw,sm;
int    wt[BIG][BIG],a,b,sum;
int    al[BIG][BIG],nal; /* 0=open 1=engaged 2=impossible */

int fc[3]={YELLOW,WHITE,BLUE};
drawal(int i, int j)
{  char t[10];
   setcolor(fc[al[i][j]]);
   if (showweights) sprintf(t," %4d",wt[i][j]);
   else             sprintf(t," %2d%2d",Pm[i][j],Pw[i][j]);
   outtextxy(8*(3+5*j),16*(n+2+i),t);
}
```

```
drawMpWp(int i,int j) { char t[10];
   sprintf(t,"%c",Mp[i][Pm[i][j]-1]); outtextxy(8*(3+Pm[i][j]),16*i,t);
   sprintf(t,"%c",Wp[j][Pw[i][j]-1]); outtextxy(8*(43+Pw[i][j]),16*j,t);
}
drawmatch(int c)
{  int i,j;
   setcolor(c);
   for (i=0;i<n;i++) for (j=0;j<n;j++) if (al[i][j]==1) drawMpWp(i,j);
}

drawperms(int c)
{  int i; char t[40];
   setcolor(c); bar(0,0,8*(44+n),16*(n));
   for (i=0;i<n;i++) {
      sprintf(t,"%3c %s",'A'+i,&Mp[i][0]); outtextxy(8*0,16*i,t);
      sprintf(t,"%3c %s",'a'+i,&Wp[i][0]); outtextxy(8*40,16*i,t);
   }
}

openp(int i,int j)
{  int k,ok=1;
   if (al[i][j]!=1) {
      for (k=0;k<n;k++) if (al[i][k]==1 && Pm[i][k]<Pm[i][j]) ok=0;
      for (k=0;k<n;k++) if (al[k][j]==1 && Pw[k][j]<Pw[i][j]) ok=0;
      al[i][j]=ok?0:2; drawal(i,j);
   }
}

int Mopt[BIG],Wopt[BIG],Mdet[BIG],Wdet[BIG];
typedef char plist[BIG][BIG];
int conflict(int *ii, int *jj, plist P, int *O)
{  int i,j;
   for (i=0;i<n;i++) for (j=i+1;j<n;j++)
      if (P[i][O[i]]==P[j][O[j]]) { *ii=i; *jj=j; return 1; }
   return 0;
}
int rejected(int ii, int jj, char *P, char off)
{  int i;
   for (i=n-1;i>=0;i--)
      if (P[i]-off==ii) return ii; else if (P[i]-off==jj) return jj;
}
gslist()
{  int i,j;
   drawperms(YELLOW);
   for (i=0;i<n;i++) { Mopt[i]=Wopt[i]=0; }
   while (conflict(&i,&j,Mp,Mopt))
      Mopt[rejected(i,j,&Wp[Mp[i][Mopt[i]]-'a'][0],'A')]++;
   while (conflict(&i,&j,Wp,Wopt))
      Wopt[rejected(i,j,&Mp[Wp[i][Wopt[i]]-'A'][0],'a')]++;
   for (i=0;i<n;i++) {
      Mdet[Wp[i][Wopt[i]]-'A']=Pm[Wp[i][Wopt[i]]-'A'][i]-1;
      Wdet[Mp[i][Mopt[i]]-'a']=Pw[i][Mp[i][Mopt[i]]-'a']-1;
   }
   setcolor(LIGHTGRAY);
   for (i=0;i<n;i++) {
      for (j=0;j<Mopt[i];j++) drawMpWp(i,Mp[i][j]-'a');
      for (j=Mdet[i]+1;j<n;j++) drawMpWp(i,Mp[i][j]-'a');
      for (j=0;j<Wopt[i];j++) drawMpWp(Wp[i][j]-'A',i);
      for (j=Wdet[i]+1;j<n;j++) drawMpWp(Wp[i][j]-'A',i);
   }
   setcolor(BLUE);
   for (i=0;i<n;i++) {
      drawMpWp(i,Mp[i][Mopt[i]]-'a'); drawMpWp(Wp[i][Wopt[i]]-'A',i);
   }
}
```

```
int rematch()
{  int i,j,k,bi,bj,bnal=0,tnal,bsum=0,tsum;
   bi=-1;
   for (i=0;i<n;i++) for (j=0;j<n;j++) if (al[i][j]==0) {
       tsum=sum+wt[i][j]; tnal=nal+1;
       for (k=0;k<n;k++) if (al[i][k]==1) { tnal--; tsum-=wt[i][k]; }
       for (k=0;k<n;k++) if (al[k][j]==1) { tnal--; tsum-=wt[k][j]; }
       if (bi==-1 || tsum*bnal*bnal<bsum*tnal*tnal)
          { bsum=tsum; bnal=tnal; bi=i; bj=j; }
   }
   if (bi==-1) { match=1; return 0; }
   sum=bsum; nal++; al[bi][bj]=1; drawal(bi,bj);
   for (i=0;i<n;i++) {
       if (al[i][bj]==0 && Pw[i][bj]>Pw[bi][bj]) al[i][bj]=2;
       if (al[i][bj]==1 && i!=bi) {
          al[i][bj]=2; nal--;
          for (j=0;j<n;j++) openp(i,j);
       } drawal(i,bj);
   }
   for (j=0;j<n;j++) {
       if (al[bi][j]==0 && Pm[bi][j]>Pm[bi][bj]) al[bi][j]=2;
       if (al[bi][j]==1 && j!=bj) {
          al[bi][j]=2; nal--;
          for (i=0;i<n;i++) openp(i,j);
       } drawal(bi,j);
   }
   gotoxy(45,n+nal); printf("%5d  (%c,%c)",sum,'A'+bi,'a'+bj);
   return 1;
}

randomperm(char *t,int n,char off)
{  int i,j,ok;
   for (i=0;i<n;i++) {
      do {
         t[i]=off+random(n);
         for (ok=1,j=i-1;j>=0;j--) if (t[j]==t[i]) ok=0;
      } while (!ok);
   }
}

int makeprefs()
{  int i,j,ok;
   for (i=0;i<n;i++) for (j=0;j<n;j++) { Pm[i][j]=0; Pw[i][j]=0; }
   for (i=0;i<n;i++) for (j=0;j<n;j++) {
      int x=Mp[i][j]-'a',y=Wp[i][j]-'A';
      if (x<0 || x>=n || y<0 ||y>=n) {ok=0; break;}
      Pm[i][x]=j+1; Pw[y][i]=j+1;
   }
   if (!ok) printf("- use 'a'..'%c' and 'A'..'%c'\n",'a'+n-1,'A'+n-1);
   else {
      for (i=0;i<n;i++) for (j=0;j<n;j++)
         if (Pm[i][j]==0 || Pw[i][j]==0) ok=0;
      if (!ok) printf("- try a permutation next time\n");
   }
   return ok;
}

getprefs()
{  int i;
   for (i=0;i<n;i++) {
      gotoxy(1,i+1);  printf("%3c ",'A'+i); gets(&Mp[i][0]);
      gotoxy(41,i+1); printf("%3c ",'a'+i); gets(&Wp[i][0]);
      if (Mp[i][0]==0) randomperm(&Mp[i][0],n,'a');
      if (Wp[i][0]==0) randomperm(&Wp[i][0],n,'A');
   }
   drawperms(LIGHTGRAY);
```

```
etgoodprefs()

  setfillstyle(SOLID_FILL,BLACK); bar(0,0,getmaxx(),getmaxy());
  do getprefs(); while (!makeprefs());
  match=0;


etweights()
  int i,j;
  gotoxy(1,n+1); printf("Weights: a Pm2 + b Pw2 ? "); scanf("%d,%d",&a,&b);
  for (i=0;i<n;i++) for (j=0;j<n;j++) {
     wt[i][j]=a*Pm[i][j]*Pm[i][j]+b*Pw[i][j]*Pw[i][j];
     if (a==0 && b==0) wt[i][j]=random(3*n*n)+1;
     al[i][j]=0;
  } sum=0; nal=0;
  for (i=0;i<=n;i++) { gotoxy(45,n+i+1); printf("                "); }
  match=0;


rawgrid()
  int i,j;
  setfillstyle(SOLID_FILL,BLACK); bar(0,16*(n+1),getmaxx(),getmaxy());
  gotoxy(5,n+2); for (i=0;i<n;i++) printf("  %c  ",'a'+i);
  for (i=0;i<n;i++) { gotoxy(1,n+3+i); printf("%3c ",'A'+i); }
  for (i=0;i<n;i++) for (j=0;j<n;j++) drawal(i,j);


alcsums()
  int i,j;
  sw=sm=0;
  for (i=0;i<n;i++) for (j=0;j<n;j++) if (al[i][j]==1) {
     sw+=Pw[i][j]; sm+=Pm[i][j];
  }


isual()
  int i,j; char key=' ';
  getgoodprefs(); setweights(); drawgrid();
  do {
     if (key=='s' || key=='c') if (!rematch()) {
        calcsums(); drawmatch(WHITE); key=' ';
        gotoxy(45,2*n+1); printf("s=%d d=%d",sw*sw+sm*sm,sw-sm);
     }
     if (kbhit() || key!='c') key=getch();
     if (key=='O') for (i=0;i<n;i++) for (j=0;j<n;j++) openp(i,j);
     if (key=='G') gslist();
     if (key=='P') { getgoodprefs(); key='W'; }
     if (key=='W') { if (match) drawmatch(BLUE); setweights(); key='D'; }
     if (key=='D') drawgrid();
     if (key=='S') { showweights=!showweights; drawgrid(); }
  } while (key!='q');


ain()

  int g_driver,g_mode,g_error; int key; char t[4];
  randomize();
  detectgraph(&g_driver,&g_mode);
  initgraph(&g_driver,&g_mode,"a:\\");

  do {
     gotoxy(1,1);
     printf("\ns=step c=continuous q=quit\n");
     printf("O=recheck open spots D=redraw allocation G=light GS list\n");
```

```
    printf("P=new preference lists W=new algorithm weights\n");
    printf("S=switch from show preference/weights matrix\n");
    printf("Number of couples? "); scanf("%d",&n); gets(t);
    visual(n);
    gotoxy(1,24); printf("Another? "); key=getch();
  } while(key=='y');

  closegraph();
}
```