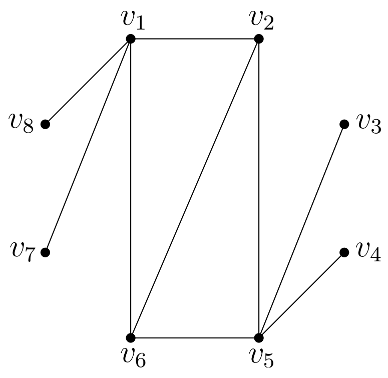
Stable Marriage

October 5-6, 2017

Matchings

In this lecture we are going to talk about matching problems. Matching problems arise in numerous applications. For example, dating services want to pair up compatible couples. Interns need to be matched to hospital residency programs. Other assignment problems involving resource allocation arise frequently, including balancing the traffic load among servers on the Internet. In the simplest form of a matching problem, you are given a graph where the edges represent compatibility and the goal is to create the maximum number of compatible pairs.

Definition. Given a graph $G = (V, E)$, a matching is a subgraph of G where every node has degree 1. In particular, the matching consists of edges that do not share nodes.

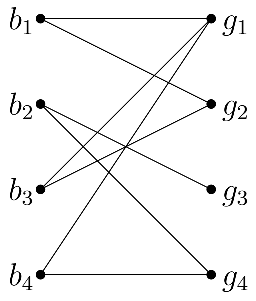


In this graph, $\{\{v_1, v_6\}, \{v_2, v_5\}\}$ is a matching of size two. But there is a larger matching – namely, $\{\{v_1, v_8\}, \{v_2, v_6\}, \{v_4, v_5\}\}$ is a matching of size three. Can there be a larger matching? Well, that would mean that every node is paired. But each of v_7 and v_8 can only be paired with v_1, and v_1 can only be paired with one other node in a matching. So, the answer is no!

Let's now define a matching that includes every node:

Definition. A matching of a graph $G = (V, E)$ is perfect if it has $|V|/2$ edges.

There is no perfect matching for the previous graph. Matching problems often arise in the context of the bipartite graphs – for example, the scenario where you want to pair boys with girls.

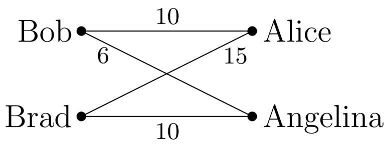


For example, the above graph has a perfect matching, namely $\{\{b_1, g_2\}, \{b_2, g_3\}, \{b_3, g_1\}, \{b_4, g_4\}\}$.

In many applications, not all matchings are equally desirable. For example, maybe b_1 and g_2 like each other a lot more than b_1 and g_1. Often, we can represent the desirability of a matching with a weight on the edge. For example b_1 and g_2 get weight 5 while b_1 and g_1 get weight 10. The goal then is to find a perfect matching with minimum weight.

Definition. The weight of matching M is the sum of the weights on the edges in M. A minimum weight matching for a graph G is a perfect matching for G with minimum weight (if it exists).

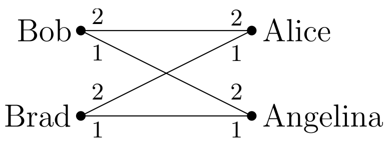
For example, a minimum weight matching for the following graph is 20 (Bob gets matched with Alice, and Brad with Angeline).



It turns out that there are fast algorithms for finding maximum matchings in unweighted graphs and minimum weight matchings in weighted graphs, but they are complicated and we don't cover them in this course (in particular, the greedy algorithm doesn't work in general).

Stable Marriage

Instead, we are going to talk about a different variant of the matching problem that does have an elegant solution and that is frequently used in practice. In this version of the problem, every node has a preference order of the possible mates. The preferences don't have to be symmetric. For example, maybe Alice really likes Brad but Brad has the hots for Angelina. Suppose Angelina also likes Brad more than Bob but that Bob really likes Angelina.



In the above figure, suppose we were to pair Brad with Alice and Bob with Angelina. Well, that would lead to a very dicey situation! Suffice it to say that pretty soon, Brad and Angelina are likely to start spending late nights doing discrete mathematics homework together. The main problem is that Brad and Angelina each prefer each other to their mates in the matching. In such a circumstance we say that Brad and Angelina form a rogue couple. More precisely, we'll say that given a matching M, boy b and girl g are a rogue couple for M if b and g prefer each other to their mates in M.

Obviously, the existence of rogue couples is not a good thing if you are making matchings, since they lead to instability. So, we'll say that a matching is stable if there are no rogue couples.

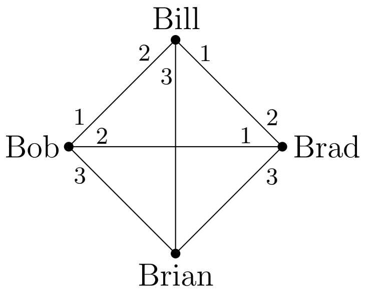
We are going to assume that preferences do not change with time. So we are not modeling the situation where you get tired. Preferences are known at the start and never change.

Our main goal is to find a perfect matching that is stable. In this example, a possible stable perfect matching is to pair Brad with Angelina, and Bob with Alice. Bob and

Alice may not be so happy, but no rogue couple is possible and so it is a stable matching. That's because neither Brad nor Angelina like anyone better than each other, so even though Alice and Bob are not happy with each other, no one else will form a rogue couple with either of them.

In general, it isn't so clear that there is always a stable matching for any number of people and set of preference orders. In fact, if you allow boys to prefer boys and girls to prefer girls, then there are examples where there is no stable matching. But the strange thing is that in the special case where boys only get pairs with girls, then you can always find a stable matching.

We're going to show how to find such a stable matching shortly. But first, let's look at a unisex example where a stable matching is not possible. The idea is to create a love triangle with a fourth person who is everyone's last choice:



It turns out the fourth person's preferences don't even matter. Let's see why there is no stable matching. We'll prove this by contradiction. Assume, for contradiction, that there is a stable matching. Then there are two members of the love triangle that are matched. Without loss of generality, (by symmetry) assume that Bill is matched to Bob. Then the other pair must be Brad matched with Brian. But then there is a rogue couple since Bill likes Brad best and Brad prefers Bill to Brian. So, Bill and Brian are a rogue couple. Thus, there cannot be a stable matching.

This proposition is not very surprising. Getting a stable matching is a hard thing to do. What is surprising is that you can always do it in bipartite graphs – that is, where boys are only allowed to pair with girls and vice versa.

Let's formalize the statement of the problem that we are discussing here.

The setting:

- There are n boys and n girls. We assume the number of boys and girls is the same.
- Each boy has his own ranked preference list of girls.
- Each girl has her own ranked preference list of boys.
- The lists are complete and have no ties. Each boy ranks every girl and vice versa.

The goal:

Pair each boy with a unique girl so that there are no rogue couples. That is, find a perfect matching so that every boy and girl are paired up one to one, with no potential funny business.

Let's see if we can figure out a method for finding a stable matching by looking at an example:

$$\begin{aligned}
b_1 &\to (g_3, g_2, g_5, g_1, g_4)\\
b_2 &\to (g_1, g_2, g_5, g_3, g_4)\\
b_3 &\to (g_4, g_3, g_2, g_1, g_5)\\
b_4 &\to (g_1, g_3, g_4, g_2, g_5)\\
b_5 &\to (g_1, g_2, g_4, g_5, g_3)
\end{aligned}$$

$$\begin{aligned}
g_1 &\to (b_3, b_5, b_2, b_1, b_4)\\
g_2 &\to (b_5, b_2, b_1, b_4, b_3)\\
g_3 &\to (b_4, b_3, b_5, b_1, b_2)\\
g_4 &\to (b_1, b_2, b_3, b_4, b_5)\\
g_5 &\to (b_2, b_3, b_4, b_1, b_5)
\end{aligned}$$

Let's try to use a greedy algorithm to find the matching. In this case, the greedy algorithm will have each boy pick his favorite girl that remains by the time his turn comes up.

Running the greedy algorithm on our example, boy b_1 picks his favorite, which is g_3, boy b_2 picks his favorite, which is g_1, boy b_3 picks his favorite, which is g_4, boy b_4 picks his favorite remaining girl, which is g_2 (since his top 3 choices are already taken), and finally, boy b_5 picks his favorite remaining girl (which at this point, is the only remaining girl), which is g_5.

Let's see – is there a rogue couple? Well, boys b_1, b_2, and b_3 are matched up with the loves of their lives, so they are too happy to be thinking of running off. However, boy b_4 is not so happy with g_2, who is his fourth choice. He approaches g_1, but she isn't interested in him, since she prefers boy b_2 – in fact, she ranked b_4 last so she wouldn't be caught dead in an affair with him. However, he runs into his love of his life, that is g_3, and she definitely prefers him to b_1, who is way down on her list. So we have a situation here. Both b_4 and g_3 prefer each other to their own mates. We could try to patch things up and pair b_4 with g_3 and then g_2 with b_1, but it's not clear that we would reduce the number of rogue couples. It happens that in this case pairing up b_4 and g_3 is an ok thing to do. But, this is getting more and more complicated.

How about using an algorithm that is based on induction (or recursion)? Pair boy b_1 with girl g_3 and solve the rest by induction. By the induction hypothesis, the only rogue couples would involve b_1 or g_3. But, they can't involve b_1, since he got his first choice. On the other hand, they might well involve g_3 since b_1 might be her last but one choice! Induction would work if there were some boy and some girl who each ranked the other first. If there were such a boy and girl, then they have to get paired to each other, or they would be a rogue couple. But, there might not be such a boy and girl. Too often people do not like those that like them!

Gale-Shapley algorithm

It turns out that finding a good way of pairing up the boys and girls is a tricky problem. The best approach is to use the Gale-Shapley algorithm named after the mathematicians

David Gale and Lloyd Shapley who devised it in 1962. Note that, in 2012, the Nobel Prize in Economics was awarded to Lloyd Shapley and Alvin Roth "for the theory of stable allocations and the practice of market design."

Here is the method for getting everyone paired up. The mating ritual takes place over several days. The idea is that each of the boys go after the girls one by one, in order of preference, crossing off girls from their list as they get rejected. Here is a more detailed specification:

Initial Condition:

Each of the n boys has an ordered list of the n girls according to his preferences. Each of the girls has an ordered list of the boys according to her preferences.

Each Day

- Morning:
  - Each girl stands on her balcony.
  - Each boy stands under the balcony of his favorite girl whom he has not yet crossed off his list and serenades. If there are no girls left on his list, he stays home and does discrete mathematics homework.
- Afternoon:
  - Girls who have at least one suitor say to their favorite from among the suitors that day: "Maybe, come back tomorrow."
  - To the others, they say "No, I will never marry you!"
- Evening:
  - Any boy who hears "No" crosses that girl off his list.

Termination Condition:

If there is a day when every girl has at most one suitor, we stop and each girl marries her current suitor (if any).

Let's run the Gale-Shapley algorithm on the example from before. On the first morning, boy b_1 serenades girl g_3, boy b_2 serenades girl g_1, boy b_3 serenades girl g_4, boy b_4 serenades girl g_1, and boy b_5 serenades girl g_1. In the afternoon, girls g_1, g_3, and g_4 say "Maybe, come back tomorrow" to boys b_5, b_1, and b_3, respectively. Girl g_1 says "No!" to boys b_2 and b_4, who cross g_1 off their lists that evening.

On the second morning, boy b_1 serenades girl g_3, boy b_2 serenades girl g_2, boy b_3 serenades girl g_4, boy b_4 serenades girl g_3, and boy b_5 serenades girl g_1. In the afternoon, girls g_1, g_2, g_3, and g_4 say "Maybe, come back tomorrow" to boys b_5, b_2, b_4, and b_3, respectively. Girl g_3 says "No!" to boy b_1, who crosses g_3 off his list in the evening.

On the third morning, boy b_1 serenades girl g_2, boy b_2 serenades girl g_2, boy b_3 serenades girl g_4, boy b_4 serenades girl g_3, and boy b_5 serenades girl g_1. In the afternoon, girls g_1, g_2, g_3, and g_4 say "Maybe, come back tomorrow" to boys b_5, b_2, b_4, and b_3, respectively. Girl g_2 says "No!" to boy b_1, who crosses g_2 off his list in the evening.

On the fourth morning, boy b_1 serenades girl g_5, boy b_2 serenades girl g_2, boy b_3 serenades girl g_4, boy b_4 serenades girl g_3, and boy b_5 serenades girl g_1. In the afternoon, the girls realize that each girl has at most one suitor, so all five couples start planning their weddings.

Now let's show that the algorithm works. We need to show that

- the Gale-Shapley algorithm terminates,
- the Gale-Shapley algorithm terminates quickly,
- at termination, there are no rogue couples,
- everyone is married.

We will also analyze the fairness of the protocol. It is better for boys or for girls?

Let's start by showing that this algorithm terminates:

Theorem 1. The Gale-Shapley algorithm terminates within $n^2 + 1$ days.

Proof. We'll prove this theorem by contradiction. Suppose, for contradiction, that the Gale-Shapley algorithm does not terminate in $n^2 + 1$ days. Well, let's notice something that must happen on a day in which the Gale-Shapley algorithm doesn't terminate – it must be that some boy crosses a girl off his list that evening! Why is this? If the Gale-Shapley algorithm doesn't terminate, then some girl must have had at least 2 suitors. If a girl has at least 2 suitors then at least one gets rejected, and that boy crosses that girl off his list. So if the Gale-Shapley algorithm doesn't terminate in $n^2 + 1$ days, there are at least $n^2 + 1$ names crossed off in total. But at the start, each list is of size n, so the total size of all the lists put together is n^2. So we couldn't have crossed off $n^2 + 1$ names, and thus we have our contradiction.

This is a typical proof technique in Computer Science used to bound the running time of an algorithm. We show that the algorithm is always making progress by some measure. Then, since there is only a finite amount of progress to make, it must eventually terminate. Here the measure is the number of names on the union of the lists.

Next, we'll prove that everyone gets married by the Gale-Shapley algorithm, but first we'll need a couple of lemmas.

Lemma 1. If a boy marries, then he courted every girl he liked better.

Proof. In the Gale-Shapley algorithm, boys cross girls off their lists one at a time in preference order, starting with the girl he likes most. A boy marries the girl (if any) that he is courting at termination. So if a boy marries, he marries his least favorite girl among those he courted. Tough luck for the boy, but at least he likes her better than all the girls he never courted.

Lemma 2. If a boy never marries, then he courted every girl.

Proof. By Theorem 1, the Gale-Shapley algorithm terminates. At the time of termination, the boy is not courting. How can that be? If he is home doing his discrete mathematics homework, then he must have crossed off every girl on his list. So, he has courted every girl.

Lemma 3. A girl marries her favorite among her suitors.

Proof. A girl only rejects a boy when a better one comes along and always keeps stringing along her favorite among those seen so far.

This also means that:

Lemma 4. If a girl is ever courted, she gets married.

Proof. Once a girl has a suitor, she keeps him until she trades up.

Now we can prove that everyone gets married:

Theorem 2. Everyone is married in the Gale-Shapley algorithm.

Proof. We'll show this one by contradiction. Assume, for contradiction, that some boy b is not married. But then, by Lemma 2, boy b has courted every girl. So, every girl has been courted. But then, every girl is married by Lemma 4. But since there are an equal number of boys and girls, it must be the case that every boy (including b) is married. So the theorem is true by contradiction.

Next we'll prove the main result, namely that the Gale-Shapley algorithm always produces stable marriages.

Theorem 3. The Gale-Shapley algorithm produces stable marriages.

Proof. Assume, for contradiction, that there is a rogue couple (b, g). Suppose b married g' and g married b' in the Gale-Shapley algorithm. If b married g', but likes g better, then b visited g first by Lemma 1, and g said "no" to b. But then, g must have married someone that she likes better than b by Lemma 3. So, g likes b' better than b, which means that (b, g) is not a rogue couple.

Well, who do you think is better off in the Gale-Shapley algorithm? In other words, who has the power, the proposers or the acceptors? Since the girls marry their favorite from among their suitors, and the boys get the worst girls that they court, it seems reasonable to assume that the girls do best. It seems hard to answer this question formally, especially since it isn't even clear what we mean by "doing better". But, in fact, we can show in a very precise and formal way that the algorithm is heavily biased toward the boys. To formalize this, we need to define the set of realistic potential mates.

Let S be the set of all stable matchings. Since the Gale-Shapley algorithm gives a stable matching, we know that $S \neq \emptyset$. For each person p, we define the realm of possibility for p to be $\{q \mid \exists M \in S,\ (p, q) \in M\}$. That is, q is within the realm of possibility for p if and only if there is a stable matching where p marries q.

Some mates just might be out of the question, since no stable pairings are possible if you married them. For example, Brad is just not realistic for Alice since if you ever pair them, Brad and Angelina will form a rogue couple – so there is no stable matching with Brad paired to Alice.

Definition. A person's optimal mate is his/her favorite from the realm of possibility.

An optimal mate must exist, since we know there is at least one stable matching, namely the one produced by the Gale-Shapley algorithm.

Definition. A person's pessimal mate is his/her least favorite from the realm of possibility.

Ok, now here is a pair of shocking results:

Theorem 4. The Gale-Shapley algorithm pairs every boy with his optimal mate!

Theorem 5. The Gale-Shapley algorithm pairs every girl with her pessimal mate!

This is too hard to believe, so let's do the proof.

Proof of Theorem 4. Assume, for contradiction, that some boy does not get his optimal girl (that is, his favorite girl within the realm of possibility). Let b be the first (in time) boy that gets rejected by his optimal girl g (resolving ties arbitrarily). Define b' to be the boy that caused g to reject b in the Gale-Shapley algorithm. Then g prefers b' to b.

Since b is the first to be rejected by the optimal mate in the Gale-Shapley algorithm, b' has not (yet) been rejected by the optimal mate when he is courting g. So, b' likes g at least as much as he likes his optimal mate g^* (g and g^* might be the same person).

Let M be a stable matching where b marries g. M exists since g is in the realm of possibility of b. M is not produced by the Gale-Shapley algorithm by assumption. Let g' be the spouse of b' in M. By definition, b' likes g^* at least as much as g' (again, they might be the same person), so b' prefers g to g' since b' likes g at least as much as g^*, whom he likes at least as much as g'. (Note that g can't be the same person as g'.) So b' and g are a rogue couple, which contradicts the fact that M is a stable matching!

Now let's show that the girls get their pessimal mate.

Proof of Theorem 5. Suppose, for contradiction, that there is a stable matching M where there is a girl g who fares worse than in the Gale-Shapley algorithm. Let b be the mate of g in the Gale-Shapley algorithm. Let b' be the mate of g in M. Then g likes b better than b' since she fared worse in M than in the Gale-Shapley algorithm. Let g' be the mate of b in M.

We know that b likes g better than g' since (by Theorem 4) the Gale-Shapley algorithm gives an optimal mate for b. Then b and g form a rogue couple in M, which is a contradiction.

Uniqueness

For obvious reason, the stable matching generated by the Gale-Shapley algorithm is called boy-optimal and girl-pessimal. If the roles of the sexes in the algorithm are interchanged, i.e., girls court boys, then the resulting stable matching is analogously girl-optimal and boy-pessimal. It may happen that the boy-oriented and the girl-oriented versions of the algorithm yield the same stable matching, in which case it is immediate, by combining the optimality and pessimality properties, that this is the unique stable matching.

Forbidden pairs

We can think about a generalization of the Stable Marriage Problem in which certain boy-girl pairs are explicitly forbidden. In this case we have a set B of n boys, a set G of n girls, and a set $F \subseteq B \times G$ of pairs who are simply not allowed to get married. Each boy b ranks all the girls g for which $(b, g) \notin F$, and each girl g ranks all the boys b for which $(b, g) \notin F$.

In this more general setting, we say that a matching M is stable if it does not exhibit any of the following types of instability.

(i) There are two pairs (b, g) and (b', g') in M with the property that $(b, g') \notin F$, b prefers g' to g, and g' prefers b to b'. (The usual kind of instability.)

(ii) There is a pair $(b, g) \in M$, and a boy b', so that b' is not part of any pair in the matching, $(b', g) \notin F$, and g prefers b' to b. (A single boy is more desirable and not forbidden.)

(iii) There is a pair $(b, g) \in M$, and a girl g', so that g' is not part of any pair in the matching, $(b, g') \notin F$, and b prefers g' to g. (A single girl is more desirable and not forbidden.)

(iv) There is a boy b and a girl g, neither of whom is part of any pair in the matching, so that $(b, g) \notin F$. (There are two single people with nothing preventing them from getting married to each other.)

Note that under these more general definitions, a stable matching need not be a perfect matching. Now we can ask: For every set of preference lists and every set of forbidden pairs, is there always a stable matching?

The Gale-Shapley algorithm is remarkably robust to variations on the Stable Marriage Problem. So, if you're faced with a new variation of the problem and can't find a counterexample to stability, it's often a good idea to check whether a direct adaptation of the Gale-Shapley algorithm will in fact produce stable matchings. That turns out to be the case here. We will show that there is always a stable matching, even in this more general model with forbidden pairs, and we will do this by adapting the Gale-Shapley algorithm.

To begin with, we notice some facts. As in the previous part, if a boy marries, then he courted every girl he liked better; and a girl marries her favorite among her suitors. Also, if b is a boy who is not part of a pair in M, then b must have courted every nonforbidden girl; and if g is a girl who is not part of a pair in M, then it must be that no boy ever courted g. Finally, the algorithm will terminate in at most $n^2 + 1$ iterations.

We need only to show that there is no instability with respect to the returned matching M. Our general definition of instability has four parts: This means that we have to make sure that none of the four bad things happens.

First, suppose there is an instability of type (i), consisting of pairs (b, g) and (b', g') in M with the property that $(b, g') \notin F$, b prefers g' to g, and g' prefers b to b'. It follows that b must have courted g'; so g' rejected b, and thus she prefers her final partner to b — a contradiction.

Next, suppose there is an instability of type (ii), consisting of a pair $(b, g) \in M$, and a boy b', so that b' is not part of any pair in the matching, $(b', g) \notin F$, and g prefers b' to b. Then b' must have courted g and been rejected; again, it follows that g prefers her final partner to b' — a contradiction.

Third, suppose there is an instability of type (iii), consisting of a pair $(b, g) \in M$, and a girl g', so that g' is not part of any pair in the matching, $(b, g') \notin F$, and b prefers g' to g. Then no boy courted g' at all; in particular, b never courted g', and so he must prefer g to g' — a contradiction.

Finally, suppose there is an instability of type (iv), consisting of a boy b and a girl g, neither of whom is part of any pair in the matching, so that $(b, g) \notin F$. But for b to be single, he must have courted every nonforbidden girl; in particular, he must have courted g, which means she would no longer be single — a contradiction.

Indifference

The Stable Marriage Problem, as discussed before, assumes that all boys and girls have a fully ordered list of preferences. In this section we will consider a version of the problem in which boys and girls can be indifferent between certain options. Again we have a set B of n boys and a set G of n girls. Assume each boy and each girl ranks the members of the opposite gender, but now we allow ties in the ranking. For example (with $n = 4$), a girl could say that b_1 is ranked in first place; second place is a tie between b_2 and b_3 (she has no preference between them); and b_4 is in last place. We will say that g prefers b to b' if b is ranked higher than b' on her preference list (they are not tied). With indifferences in the rankings, there could be two natural notions for stability. And for each, we can ask about the existence of stable matchings, as follows.

(a) A strong instability in a perfect matching M consists of a boy b and a girl g, such that each of b and g prefers the other to their partner in M. Does there always exist a perfect matching with no strong instability?

(b) A weak instability in a perfect matching M consists of a boy b and a girl g, such that their partners in M are g' and b', respectively, and one of the following holds:

- b prefers g to g', and g either prefers b to b' or is indifferent between these two choices; or
- g prefers b to b', and b either prefers g to g' or is indifferent between these two choices.

In other words, the pairing between b and g is either preferred by both, or preferred by one while the other is indifferent. Does there always exist a perfect matching with no weak instability?

(a) The answer is yes. A simple way to think about it is to break ties in some fashion (e.g., lexicographically) and then run the Gale-Shapley algorithm on the resulting preference lists. We claim that the matching produced would have no strong instability. But this is true because any strong instability would be an instability for the matching produced by the algorithm, yet we know that the algorithm produced a stable matching — a matching with no instabilities.

(b) The following is a simple counterexample. Let $n = 2$ and b_1, b_2 be the two boys, and g_1, g_2 be the two girls. Let b_1 be indifferent between g_1 and g_2, and let both of the girls prefer b_1 to b_2. The choices of b_2 are insignificant. There is no matching without weak instability in this example, since regardless of who was matched with b_1, the other girl together with b_1 would form a weak instability.

Truthfulness

Now we will explore the issue of truthfulness in the Stable Marriage Problem and specifically in the Gale-Shapley algorithm. The basic question is: Can a boy or a girl end up better off by lying about his or her preferences? More concretely, we suppose each participant has a true preference order. Now consider a girl g. Suppose g prefers boy b to b', but both b and b' are low on her list of preferences. Can it be the case that by falsely claiming that she prefers b' to b at some iteration of the Gale-Shapley algorithm, g will end up with a boy b'' that she truly prefers to both b and b'? (We can ask the same question for boys, but will focus on the case of girls for purposes of this question.)

Assume we have three boys b_1, b_2, b_3 and three girls g_1, g_2, g_3 with preferences as given below.

$$
\begin{aligned}
b_1 &\rightarrow (g_3, g_1, g_2)\\
b_2 &\rightarrow (g_1, g_3, g_2)\\
b_3 &\rightarrow (g_3, g_1, g_2)\\
\\
g_1 &\rightarrow (b_1, b_2, b_3)\\
g_2 &\rightarrow (b_1, b_2, b_3)\\
g_3 &\rightarrow (b_2, b_1, b_3)
\end{aligned}
$$

Let's run the Gale-Shapley algorithm on this example.

On the first morning, boy b_1 serenades girl g_3, boy b_2 serenades girl g_1, and boy b_3 serenades girl g_3. In the afternoon, girls g_1 and g_3 say "Maybe, come back tomorrow" to boys b_2, b_1, respectively. Girl g_3 says "No!" to boys b_3, who crosses g_3 off his list that evening.

On the second morning, boy b_1 serenades girl g_3, boy b_2 serenades girl g_1, and boy b_3 serenades girl g_1. In the afternoon, girls g_1 and g_3 say "Maybe, come back tomorrow" to boys b_2, b_1, respectively. Girl g_1 says "No!" to boys b_3, who crosses g_1 off his list that evening.

On the third morning, boy b_1 serenades girl g_3, boy b_2 serenades girl g_1, and boy b_3 serenades girl g_2. In the afternoon, the girls realize that each girl has at most one suitor, so all three couples start planning their weddings.

Now consider execution of the Gale-Shapley algorithm when g_3 pretends she prefers b_3 to b_1.

On the first morning, boy b_1 serenades girl g_3, boy b_2 serenades girl g_1, and boy b_3 serenades girl g_3. In the afternoon, girls g_1 and g_3 say "Maybe, come back tomorrow" to boys b_2, b_3, respectively. Girl g_3 says "No!" to boys b_1, who crosses g_3 off his list that evening.

On the second morning, boy b_1 serenades girl g_1, boy b_2 serenades girl g_1, and boy b_3 serenades girl g_3. In the afternoon, girls g_1 and g_3 say "Maybe, come back tomorrow" to boys b_1, b_3, respectively. Girl g_1 says "No!" to boys b_2, who crosses g_1 off his list that evening.

On the third morning, boy b_1 serenades girl g_1, boy b_2 serenades girl g_3, and boy b_3 serenades girl g_3. In the afternoon, girls g_1 and g_3 say "Maybe, come back tomorrow" to boys b_1, b_2, respectively. Girl g_3 says "No!" to boys b_3, who crosses g_3 off his list that evening.

On the fourth morning, boy b_1 serenades girl g_1, boy b_2 serenades girl g_3, and boy b_3 serenades girl g_1. In the afternoon, girls g_1 and g_3 say ”Maybe, come back tomorrow” to boys b_1, b_2, respectively. Girl g_1 says ”No!” to boys b_3, who crosses g_1 off his list that evening.

On the fifth morning, boy b_1 serenades girl g_1, boy b_2 serenades girl g_3, and boy b_3 serenades girl g_2. In the afternoon, the girls realize that each girl has at most one suitor, so all three couples start planning their weddings.

As we see, girl g_3 ends up with the boy b_2, who is her true favorite. Thus we conclude that by falsely switching order of her preferences, a girl may be able to get a more desirable partner in the Gale-Shapley algorithm.

College Admission

Next, we are going to talk about a generalization of the stable marriage problem. In the new problem, there are n students $s_1, s_2, \ldots, s_n$ and m universities $u_1, u_2, \ldots, u_m$. University u_i has n_i slots for students, and we're guaranteed that

$$\sum_{i=1}^{m} n_i = n.$$

Each student ranks all universities (no ties) and each university ranks all students (no ties). Design an algorithm to assign students to universities with the following properties

1. Every student is assigned to one university.
2. University u_i gets assigned n_i students.
3. There does not exist s_i, s_j, u_k, u_ℓ where student s_i is assigned to university u_k, student s_j is assigned to university u_ℓ, student s_j prefers university u_k to university u_ℓ, and university u_k prefers student s_j to student s_i.
4. It is student-optimal. This means that of all possible assignments satisfying the first three properties, every student gets his/her top choice of university amongst these assignments.

The algorithm will be a slight modification of the mating algorithm given before.

Each Day

- Morning:
  - Each university asks which students are interested in applying.
  - Each student applies to his/her favorite university that has not yet rejected him/her. If there are no universities left on the student's list, the student takes some time off to think about life and the future.
- Afternoon:
  - Each university u_i tells its favorite n_i applicants ”Maybe, we are still processing your application.” If u_i has less than n_i applicants, it tells all of its applicants this message.

    - If u_i has more than n_i applicants, it tells the remaining ones "Sorry, there were a large number of very qualified students applying this year, yet we can only accept a very limited number. We regret to inform you that you were not accepted. Thank you for applying to our university."

- Evening:
    - Any student who hears "Sorry, ..." from some university, crosses off that university from his/her list.

Termination Condition:

If there is a day when each university u_i has at most n_i applicants, we stop and each university accepts all of its applicants (if any).

Before we can say anything about our algorithm, we need to show that it terminates.

Theorem 1. The algorithm terminates within $nm + 1$ days.

Proof. On each day, if the algorithm has not terminated, then some university u_i has more than n_i applicants. It follows that in the afternoon, at least one student s_j hears "Sorry, ...", and thus in the evening s_j crosses off u_i from his/her list. As there are n students and m universities, it follows that the algorithm must terminate after $nm + 1$ days, as otherwise there would be no university left for any student to cross off.

Next, we show that the four properties stated earlier are true for our algorithm. To start, we show the following:

Lemma 1. If during some day a university u_j has at least n_j applicants, then when the algorithm terminates it accepts exactly n_j students.

Proof. At this day, each of the students applying to u_j has u_j as their favorite university that has not yet rejected him/her. Therefore, if u_j tells a student "Maybe, ...", that student will come back the next day. Since there are at least n_j applicants, it follows that u_j will tell its favorite n_j applicants "Maybe, ...". It follows by induction that every day after this day, u_j will have at least n_j applicants. Thus, this holds when the algorithm terminates. Since when the algorithm terminates there are at most n_j applicants, it follows that exactly n_j students are assigned to u_j.

Lemma 2. Every student is assigned to one university.

Proof. It is clear that no student can apply to more than one university at once since a student applies to at most one university on any given day, so this means the students can be assigned to at most one university. So we just need to show that each student is assigned to at least one university.

We argue by contradiction. Suppose not, and let s_j be a student not assigned to any university. Then, since the algorithm terminates, and when the algorithm terminates each university u_i accepts at most n_i students, it follows that some university u_i accepts less than n_i students. By Lemma 1, it follows that in every day, u_i had less than n_i applicants. But then consider the day that s_j applied to u_i. Since there were less than n_i applicants to u_i that day, it follows that u_i would have told s_j "Maybe, ..." in that day, and thus in every future day. Thus, s_j would be assigned u_i when the algorithm terminates. This is a contradiction.

Theorem 2. For each $1 \leqslant i \leqslant m$ university u_i gets assigned n_i students.

Proof. Since the algorithm terminates, on some day each u_i gets assigned at most n_i students. Suppose some u_i got assigned strictly less than n_i students. Since

$$\sum_{i=1}^{m} n_i = n,$$

this means that some student is not assigned which contradicts the previous lemma.

Before continuing, we need to establish the following property. Suppose that on some day a university u_j has at least n_j applicants. Define the rank of an applicant s_i with respect to a university u_j as s_i's location on u_j's preference list. So, for example, u_j's favorite student has rank 1.

Lemma 3. The rank of u_j's least favorite applicant that it says "Maybe, ..." to cannot decrease on any future day.

Proof. On the next day, there are two cases: u_j either says "Maybe, ..." to its least favorite applicant s_i from the previous day, or it says "Sorry, ..." to s_i. In the first case, this means that all of the $n_j - 1$ applicants u_j liked more than s_i on the previous day will also be told "Maybe, ...", and so s_i will again be u_j's least favorite applicant it did not reject. Thus, the rank of its least favorite applicant did not decrease. In the second case, this means that there were at least n_j applicants that u_j preferred to s_i, and thus the rank of its least favorite applicant it said "Maybe, ..." to did not decrease. As shown above, on any future day u_j has at least n_j applicants, and so by applying this analysis again, we conclude that the rank of u_j's least favore applicant that it says "Maybe, ..." to cannot decrease on any future day.

Theorem 3. There does not exist s_i, s_j, u_k, u_ℓ where student s_i is assigned to university u_k, student s_j is assigned to university u_ℓ, student s_j prefers university u_k to university u_ℓ, and university u_k prefers student s_j to student s_i. Note that this is analogous to a "rogue couple" considered before.

Proof. Assume, for contradiction, that such s_i, s_j, u_k, and u_ℓ existed. Since s_j prefers u_k to u_ℓ, but is assigned to u_ℓ, on some day u_k told s_j "Sorry, ...". On that day, there must have been more than n_k applicants to u_k. If s_i was also an applicant to u_k on that day, then s_i would have also been rejected since u_k prefers s_j to s_i, and thus s_i could not have been assigned to u_k. On the other hand, if at any later day s_i were to apply to u_k, it would have been rejected since s_i's rank is greater than s_j's with respect to u_k, and by Lemma 3 we know that the rank of the least favorite applicant that u_k says "Maybe, ..." to, cannot decrease. Thus, it is impossible for s_i to be assigned to u_k, which is a contradiction.

Finally, we show that this algorithm is student-optimal. As before, define the realm of possibility of a student to be the set of all universities u, for which there exists some assignment satisfying the first three properties above, in which the student is assigned to

u. Of all universities in the realm of possibility of a student we say that the student's favorite is optimal for that student.

Theorem 4. Each student is assigned to his/her optimal university.

Proof. We argue by contradiction. Consider the first (in time) student s_i that gets rejected by his/her optimal university u_k. Then on the date s_i is rejected from u_k, there was another student s_j which u_k preferred to s_i. Since u_k is in the realm of possibility of s_i, there is an assignment M of students to universities assigning s_i to u_k with the properties above. Suppose M assigns s_j to u_ℓ. Suppose u_{opt} is s_j's optimal university. Then, since s_i was the first student not assigned its optimal university, s_j prefers u_k to u_{opt}, though u_k may equal u_{opt}. On the other hand, s_j prefers u_{opt} to u_ℓ, since u_{opt} is its favorite university in its realm of possibility, and u_ℓ occurs in its realm of possibility. It follows that s_j prefers u_k to u_ℓ. But then in the assignment M we have found an s_i, s_j, u_k, and u_ℓ with s_i assigned to u_k, s_j assigned to u_ℓ, s_j prefers u_k to u_ℓ, and u_k prefers s_j to s_i. This is a contradiction to the property of M established in Theorem 3.

College Admission Revised

In this version of the problem, there are n students $s_1, s_2, \ldots, s_n$ and m universities $u_1, u_2, \ldots, u_m$ again. University u_i has n_i slots for students, but now we're guaranteed only that

$$\sum_{i=1}^{m} n_i \leqslant n.$$

Each student ranks all unviersities (no ties) and each university ranks all students (no ties).

The interest, naturally, is in finding a way of assigning each student to at most one university, in such a way that all available positions in all universities are filled. (Since we are assuming a surplus of students, there would be some students who do not get assigned to any university.) We say that an assignment of students to universites is stable if neither of the following situations arises.

- First type of instability: There are students s_i and s_j, and a university u_k, so that
  - s_i is assigned to u_k,
  - s_j is assigned to no university,
  - u_k prefers s_j to s_i.
- Second type of instability: There are students s_i and s_j, and universities u_k and u_ℓ, so that
  - s_i is assigned to u_k,
  - s_j is assigned to u_ℓ,
  - u_k prefers s_j to s_i,
  - s_j prefers u_k to u_ℓ.

We show that there is always a stable assignment of students to universities, and give an algorithm to find one.

It is easy to see that the previous algorithm produces a stable assignment in this case too. Here is an alternate solution. At any point in time, a student is either "committed" to a university or "free", and a university either has available slots, or it is "full".

```
While there is a university $u_i$ which has available slots and hasn't
offered a position to every student
   $u_i$ offers a position to the next student $s_j$ on its preference list
   if $s_j$ is free
      then
         $s_j$ accepts the offer
      else /* $s_j$ is already committed to a university $u_k$ */
         if $s_j$ prefers $u_k$ to $u_i$
            then
               $s_j$ remains committed to $u_k$
            else
               $s_j$ becomes committed to $u_i$
               the number of available slots at $u_k$ increases by one
               the number of available slots at $u_i$ decreases by one
```

The algorithm terminates in $O(mn)$ steps because each university offers a position to a student at most once, and in each iteration some university offers a position to some student.

The algorithm terminates with an assignment in which all available slots are filled, because any university that did not fill its slots must have offered a position to every student, but then, all these students would be committed to some university, which contradicts the assumption that

$$\sum_{i=1}^{m} n_i \leqslant n.$$

Finally, we argue that the assignment is stable. For the first type of instability, suppose there are students s_i and s_j, and a university u_k as above. If u_k prefers s_j to s_i, then u_k would have offered a position to s_j before it offered one to s_i; from then on, s_j would have a position at some university, and hence would not be free at the end — a contradiction. For the second type of instability, suppose there are students s_i and s_j, and universities u_k and u_ℓ as above, and u_k and s_j form a pair that causes instability. Then u_k must have offered a position to s_j, for otherwise it has n_k students all of whom it prefers to s_j. Moreover s_j must have rejected u_k in favor of some university which he/she preferred to u_k. Since s_j is finally committed to u_ℓ, therefore s_j also prefers u_ℓ to u_k — a contradiction again.

Concluding remarks

Perhaps the most famous application of the Gale-Shapley algorithm is in matching fresh MDs to residency programs. Fourth year medical students have to fill out a form with their top 20 choices for residency programs. Teaching hospitals do the same thing with their top choices for doctors. Then the data is fed to the algorithm which matches doctors to hospitals. The doctors find out their assignments on match day, which is a huge event.

Not surprisingly, the Gale-Shapley algorithm is also used by large dating agencies.

Bibliography

[1] Dan Gusfield and Robert W. Irving : *The Stable Marriage Problem: Structure and Algorithms.* The MIT Press, 1989.

[2] Robert Kleinberg, Éva Tardos: *Algorithm Design.* Addison-Wesley, 2006.

[3] Donald E. Knuth: *Stable Marriage and its Relation to Other Combinatorial Problems.* CRM Proceedings and Lecture Notes, American Mathematical Society, 1997.

[4] Eric Lehman, Thomas F. Leighton, Albert R. Meyer: *Mathematics for Computer Science.* Notes for class MIT 6.042, 2017.

[5] David F. Manlove: *Algorithmics of Matching under Preferences.* World Scientific, 2013.

[6] Alvin E. Roth and Marilda A. Oliveira Sotomayer: *Two-Sided Matching: A Study in Game-Theoretic Modeling and Analysis.* Cambridge University Press, 1990.