Lecture 7

Matching

Prep

* TMA homework plan based on recitation, many boys to 1 girl
* Handout on TMA
* Thm 4&5 in recitation or PS4
* Get regular bipartite graph decomposition into matchings in recitation
* Work up hall’s thm in recitation
* Copies of TMA handout

Take

* Handout on the mating algorithm

**Handout: The Mating Algorithm**

**Reminders:**

**PS 4 due Monday at 7:30 pm \*\***

**Read sec 5.0 – 5.3**

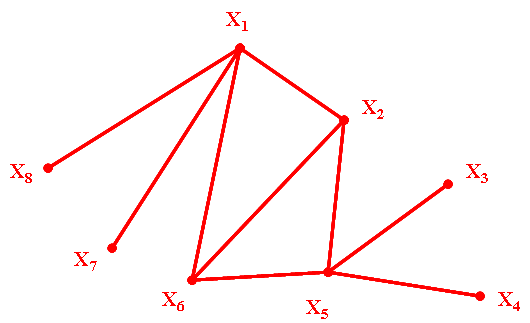
On Tuesday, we talked about sex – so today we are going to talk about marriage. In terms of graph theory, marriage can be expressed as a matching problem. Matching problems arise in all sorts of applications. Today, we’ll talk about a matching algorithm that is used by on-line dating services to match people together and by hospitals to match up doctors to residency programs. At Akamai, we use a variant of the algorithm to assign web traffic to servers. The algorithm is considered to be so important that the inventors of the algorithm won the Nobel Prize in economics for it in 2012.

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 max # of compatible pairs.

**Def: Given a graph G = (V, E), a matching is a subgraph of G where every node has degree 1.**

So everyone is married to at most one person.

**Ex:**

****

**X1  - X6, X2  - X5 is a matching of size 2. Show on graph.**

**Q:** Can we find a larger matching?

**A:** Yes X1  - X8, X2  - X6, X5  - X4, is a matching of size 3.

**Show on Graph**

**Q:** Can we do a larger matching?

**A:** No

**Q:** Why not? Can anyone give me a proof?

**A:** Since we have 8 nodes, a size 4 matching would mean every node is paired. Only possible mate for X7 & X8is X1 and X1 can only be paired with 1 node.

A matching that includes every node is called a perfect matching.

**Def: A matching** of a graph G = (V, E) **is perfect if it has  edges.**

So there is no perfect matching for this graph.

Matching problems often arise in the context of bipartite graphs – for example the scenario where you want to pair boys with girls.

**Ex: b1 g1**

**b2 g2**

**b3 g3**

**b4 g4**

**Q:** Does this graph have a perfect matching?

**A:** Yes – **show it**. b1 - g2, b2 - g3, b3 - g1, b4  - g4

In some applications, some pairings are more desirable than others. And we represent the desirability with a weight. For example, maybe b1 and g2 like each other a lot more than b1 and g1. So we might weight the edge between b1 and g2 with 5 and give the edge between b1 and g1 weight 10.

**Draw weights on edges**

Usually, we use lower weight to denote a stronger liking. The goal then is to find the perfect matching with the minimum total weight.

**Def: The weight of a matching M is the sum of the weights on the edges of M.** (non edges often considered to have ∝ weight).

**Def: A min-weight matching for** a graph **G is a perfect matching for G with minimum weight.**  (If it exists).

For example, here is a weighted graph with four people:

**10**

**Ex: Brad Jennifer**

**16**

**5**

**Billy Bob Angelina**

**10**

**Q:** what is the weight of the min-weight matching in this graph?

**A:** **Brad – Jenn**

**20**

**Billy Bob – Angelina**

The other way would be 21 – not as good, even though Brad and Angelina really like each other, or at least they did.

Questions?

It turns out that finding max matchings in un-weighted graphs and min-wt matchings in weighted graphs is doable with fast algorithms. These problems are not NP-complete like the graph coloring problem that we saw on Tuesday so no prize money for doing it. But the algorithms are fairly complicated and we don’t have time to cover them in 6.042. Instead, for the rest of today, we are going to talk about a different variant of the matching problem that does have an elegant solution which is frequently used in practice.

In this version of the problem, every node has a preference order of the possible mates and the preferences don’t have to be symmetric. For example, maybe Jennifer really likes Brad but Brad has the hots for Angelina instead.

**Fill out 1’s & 2’s on graph:**

**1**

**1**

**1**

**1**

**2**

**2**

**2**

**2**

**Ex:** **Brad Jennifer**

**BTW: should make clear that I’m not referring to our TA Angelina here – she’s not been hanging with Brad at least as far as I know. \*\***

**Billy Bob Angelina**

Suppose Angelina also likes Brad more than Billy Bob but that Billy Bob really likes Angelina. Basically, both guys prefer Angelina and both women prefer Brad.

**Q:** What happens if we were to set up our marriages to pair Brad with Jenn and Billy Bob with Angelina?

**Show on graph**

What if we put them all on a desert island together—what do you think might happen?

**A:** Trouble! Brad & Angelina like each other best so don’t be surprised if you see Brad and Angelina doing 6.042 homework together late at night.

Because Brad & Angelina prefer each other to their mates, we say that Brad and Angelina form a rogue couple.

**Def: Given a matching M , x & y are a rogue couple in M if x & y prefer each other to their mates in M.**

Show on graph

Obviously, the existence of rogue couples is not a good thing if you are making matchings since they lead to instability. This leads to definition of stability for a matching.

**Def: A matching is** said to be **stable if there are no rogue couples.**

Note: We are going to assume that preferences do not change with time. So we are not modeling the situation where you get tired of your mate or you want to play the field. Preferences known at start and never change.

**Goal: Find a perfect matching that is stable.**

We want everybody married and no rogue couples.

**Q:** Is it doable in this example?

**A:** Yes

Brad – Angelina, Billy Bob – Jenn

**Q:** Why no rogue couple?

**A:** Brad & Angelina not going anywhere so Billy Bob & Jenn have nowhere to go either. Brad won’t leave Angelina to fool around with Jenn and Angelina is not going to drop Brad for Billy Bob. Both people in a rogue couple must prefer each other to their mates.

Note that we are not making everyone happy here. In fact, both Billy Bob and Jenn could well be unhappy about getting their second choices but no rogue couples this way and so stable.

Questions?

In this 4-person example, there was a stable matching, but what about in general? Is there always a stable matching for any number of people and every set of preference orders?

**Q:** Who is thinking yes?

**Q:** Who is thinking no?

**A:** In some sense, you are both right!

If you allow boys to match up with boys and girls to match up with girls, then there are examples where there is no stable matching. But in special case where boys only get paired with girls, then you can always find stable matching. We’ll spend rest of today showing how to find one, but first, let’s look at unisex example where it is not always possible.

The idea is to create a love triangle among 3 people and then have a 4th person who is everyone’s last choice.

**Draw love triangle first and Mergatoid last.**

**Alex**

**2**

**1**

**3**

**2**

**1**

**Robin**  **Bobby Joe**

**3**

**1**

**2**

**3**

**Mergatoid**

Nobody likes poor Mergatoid. (Hopefully, nobody is named Mergatoid in the class—will get complaints.)

Lets prove that there is no stable matching for this group.

**Thm ¬∃ stable matching**

**Pf: by contradiction**

**Assume for purp of cont**. **that ∃ stable matching M. Then Mergatoid is matched with someone in M. WLOG (by symmetry) assume Mergatoid matched to Alex.**

**Explain WLOG: would be cases & implies argument is the same for every case. Be careful when you use this.**

**Show Mergatoid matched to Alex & then robin matched to Bobby Joe**

**Q:**  **∃** rogue couple?

**A:** Yes. Robin and Alex

**Show on graph**

**Q:** Why?

**A:** Robin likes Alex best and Alex for sure likes Robin better than Mergatoid.

**⇒ Alex and Robin are rogue couple for M .**

**⇒ M is not stable. #**

Questions?

Not very surprising. Getting a stable matching is a hard thing. The surprising fact is that you can always do it in the case where boys only allowed to pair with girls and vice versa. We’ll spend rest of lecture showing how to do it. The result is pretty famous--known as:

**Stable Marriage Problem**

* **N boys & N girls**  - same number each. Scenario where 2 or more boys for every girl or vice versa will be covered in PS4 – turns out to be useful in practice (not with boys and girls of course but in real world assignment problems) as we’ll see later.
* **Each boy has his own ranked** preference **list of all the girls**
* **Each girl has her own ranked preference list of all the boys**
* Lists are complete and have no ties

i.e. each boy ranks every girl and vice versa.

**Goal: Find perfect matching without rogue couples.**

i.e., Find perfect matching so every boy and girl paired up 1- 1, and no funny business going on.

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

**Preference Lists**

**Boys Girls**

**1: C B E A D A: 3 5 2 1 4**

**2: A B E C D B: 5 2 1 4 3**

**3: D C B A E C: 4 3 5 1 2 SAVE**

**4: A C D B E D: 1 2 3 4 5**

**5: A B D E C E: 2 3 4 1 5**

**Q:** Any ideas for what kind of algorithm might produce good pairings?

**A:** A Greedy algorithm is always a good first choice. For example, we could process the boys in order and give each one his best available choice.

**Do on board**

**A:**  **Greedy** boy 1 picks favorite **1 → C**

**Do on Board**

boy 2 picks favorite **2 → A**

boy 3 picks favorite **3 → D**

**← Since top 3 taken**

boy 4 picks 4th choice **4 → B**

boy 5 picks last one left **5 → E**

**Only one left**

OK, let’s see if there are any rogue couples.

**Q:** Can boy 1 be in a rogue couple?

**A:** No, since he got first choice and would never think of running off.

Same goes for boys 2 and 3.

But what about boy 4? Boy 4 is not so happy with B as mate – 4th on his list. So let’s check to see if boy 4 forms a rogue couple with one of the other girls.

**Q:** Is 4, A a rogue couple?

**A:** No. A likes her mate -2- better than 4. A ranked 4 dead last – A wouldn’t be caught dead in an affair with 4!

Let’s try 4, B. … They can’t be rogue since they got married.

**Q:** What about 4, C? Can they be rogue?

**A:** Yes! Girl C has the hots for boy 4. Boy 4 is her first choice and boy 4 prefers C to his mate (B).

So 4,C is a rogue couple and this matching is not stable.

So greedy algorithm did not work. Too bad, but that sometimes happens in practice and so we need to think up a different algorithm.

**Maybe observe: girls can get 1st choices in this case, but that is coincidence.**

**Maybe observe: you can try to patch up greedy algorithm but not clear that works in general.**

**Q:** How about another approach?

**A:** Could try induction or recursion:

Pair 1 → C & Solve rest by induction

By I.H. only rogue couples would involve 1 or C. Can’t involve 1 since he got 1st choice. But might well involve C since 1 might be her last choice! And, in fact, you might end up with 4 & C being rogue, just as with greedy algorithm.

Induction would work if there were boy and girl who ranked each other 1st. They have to get paired (or they would be rogue couple), but such a boy & girl might not exist. Often people do not like those who like them! People often want most what they can’t have. For example, there is no 1-1 pair in this case.

Turns out this is a tricky problem. The best approach is to use a dating protocol that was popular in the 1950’s. The protocol is explained on the handout. Let’s look at that now.

**Hold up handout**

The Mating Algorithm (TMA)

**Initial Condition:** Each of the N boys has an ordered list of the N girls according to his preferences. And vice-versa.

***The mating ritual takes place over several days.***

**Each Day is broken into 3 parts: morning, afternoon, and evening.**

* **Morning:**
  + Each girl goes out 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 6.042 homework.
* **Afternoon:**
  + Girls who have at least one suitor say to their favorite from among the suitors that day: “Maybe I’ll marry you. Come back and serenade me again tomorrow.”
  + ***Girls don’t want to make it too easy ☺***
  + To the others, they say “No, I will never marry you! Go away and don’t ever come back.”
* **Evening:**
  + Any boy who hears “No” crosses the girl off his list. ***It’s the only practical thing to do.***
  + ***If the boy hears maybe, he’ll go back the next day since that girl is still his favorite among those who are not crossed off. He’s hoping that she will eventually say “Yes.”***

**Termination Condition:** If there is a day when every girl has at most one suitor, we stop and each girl who has a suitor says “Yes: I will marry you.” If there is no suitor, then the girl does not get married—but we’ll show that never happens.

Let’s try out this algorithm on our example and see if it works.

**Draw Frame on Board (LHS & RHS)**

**Serenaders**

**Girl Day 1 Day 2 Day 3 Day 4**

**A  5 5 5**

**B 2  2 2**

**C 1 , 4 4 4**

**D 3 3 3 3**

**E** **1**

**Crossouts**

**Boy Day 1 Day 2 Day 3 Day 4**

**1 C B**

**2 A**

**3**

**4 A**

**5**

Let’s see what happens on Day 1

**Do Boy Choices:**

A is very popular & has 3 boys there. B & E have no action at all.

**Q:** Who does A keep around with “maybe”?

**A:** Boy 5. She says no to 2 & 4 so they cross girl A off their lists.

That finishes day 1. Now on to day 2.

**Draw Day 2 boy choices:**

Boys 1,3, and 5 go back to where they were on day 1 and try again. Boys 2 and 4 try their second choices.

**Q:** What does girl C do now?

**A:** Girl C is thrilled with boy 4 so she says no to 1 – booted after getting “maybe” because better fish came along.

So now we head on to day 3.

**Draw Day 3 boy choices:**

Boys 2, 3, 4, and 5 go back to where they were on day 2 and try again. Since boy 1 got booted, he tries his 2nd choice: girl B.

**Q:** What does girl B do now?

**A:** B likes boy 2 better so she rejects boy 1 – poor boy 1 gets rejected again -- It’s a tough world out there for boys **☺**

**Q:** So where does boy 1 go on day 4?

**A:** Girl E, his 3rd choice.

**Draw Day 4**

**Q:** So now what happens?

**A:** Everyone gets married – pretty fast after 4 days

Questions?

Now let’s check if there are any rogue couples in this matching. Start by checking boy 1. He got bounced around a bunch so he might be rogue. Boy 1 got girl E – his 3rd choice so 1,C or 1,B might be rogue couples. Let’s check:

**Q:** Could 1,C be a rogue couple?

**A:** No C got 1st choice – so she’s not going anywhere

**Q:** What about 1,B?

**A:** Nope. B likes her mate (boy 2) better than boy so no way she is going to fool around with boy 1.

So boy 1 is not going to be in a rogue couple.

**Q:** What about boy 2?

**A:** Got girl B, his 2nd choice, so only possible rogue couple with boy 2 is with girl A.

**Q:** Are 2&A rogue?

**A:** No – A got 5 and likes him better so 2 not in rogue couple.

**Q:** What about Boy 3?

**A:** Got 1st choice – not even thinking about going elsewhere

**Q:** Boy 4 got 2nd choice – 4 & A might be rogue – Is it?

**A:** No way – A hates 4! A wouldn’t be caught dead with 4. So 4 is safe.

**Q:** Lastly – Boy 5

**A:** Got girl A – 1st choice – so not going anywhere.

So no rogue couples in this matching. The algorithm worked in this example and everyone lives happily ever after.

Questions?

It turns out that the Mating Algorithm always works. It’s a pretty amazing result that was first proved by David Gayle and Lloyd Shapley in 1962. The algorithm was applied to solve a bunch of real-world problems in the 1980s by Alvin Roth. And then in 2012, Shapley and Roth got the Nobel Prize in economics for their work. (Gayle died in 2008 or he would have gotten it too.)

The proof that the Algorithm always produces stable matchings is a little tricky. It will be the hardest proof we have done in class, but after all, it did win a Nobel prize…

To prove it works, we’ll need to show several things:

**Need to show:**

* **TMA Terminates** – don’t want those poor guys hearing “maybe” forever
* **TMA terminates (quickly)** – boys can only serenade for so long. Quick termination is why it is so useful in practice.

**SAVE**

* **No rogue couples** – this is main goal: would be a shame if after all this work, rogue couples spoiled things
* **Everyone is married** – stability is easy if no marriages!

Start by showing algorithm terminates.

**Thm 1: TMA terminates within N2 + 1 days. SAVE**

Recall – N is number of boys (and girls).

Actually, TMA usually takes a lot fewer steps, but all we need for now is that it terminates.

**Pf: by contradiction**

**Suppose for purp. of cont. that TMA does not terminate in N2 + 1 days.** Need to get a contradiction. One good way to do that is to show that some kind of progress is made each day that algorithm does not terminate. And that after N2+1 days without termination, we would have made more progress than is possible—and that will give us our contradiction.

**Q:** Any ideas on a measure of progress that is made every day that we don’t terminate? What must happen on a day when we don’t terminate?

**A: Claim: if we don’t terminate on a day, then some boy crosses a girl off his list that evening.**

**Q:** Why is this true?

**A:** failure to terminate means some girl had 2 or more suitors, at least one of whom was rejected and the rejected boy crosses that girl off his list.

**Q:** so if we don’t terminate in N2+1 days, how many names must have been crossed off all the lists?

**A:** **at least N2 + 1 names crossed off lists.**

**Q:** is it possible to have at least N2+1 crossouts?

**A:** No!

**Q:** How many lists are there?

**A:** N

**Q:** How many names on each list?

**A:** N

**N lists with N names ⇒ ≤ N2 crossouts. # **

Questions?

This is a very common proof technique in computer science—there are many examples where it is used to bound the running time of algorithm. Show that algorithm is always making progress by some measure and so must eventually terminate. Here, measure is # crossouts on boys’ lists.

We still have to prove that everyone gets married and that the marriages are stable. To do that, we are going to use an invariant. Remember when we talked about invariants 2 weeks ago? It is a property of the system that is true at the beginning and that is preserved at every step along the way.

**Q.** Can anyone think of an invariant of the marriage algorithm that might be useful?

**Explore ideas**

**Q.** What about the situation for girls? What can we say about a girl (say Gail) that rejects a boy (call him Bob)?

**A.** If Gail rejects Bob, then she has someone she likes better under her balcony. Day after day, things only get better for girls.

Let’s formalize this as a Lemma.

Lemma 1: If a girl G previously rejects a boy B, then going forward, G **always** has a suitor who she prefers to B. Eventually the suitor becomes her husband if the algorithm terminates.

**SAVE**

This lemma is not hard to prove by induction on the number of days, but I am not going to write down all the details on the board. Basically, you use the fact that on each day, a girl never goes backward—she keeps her current favorite coming back each day until she finds someone she likes better, and then she keeps that person coming back, and so on. This goes on forever or until the algorithm terminates and then she marries her suitor on the last day.

Questions?

OK, now we can prove that everyone gets married in TMA.

**Thm 2: Everyone is married in TMA**

**Pf: by contradiction.**

**Assume for purp. of cont. that NOT everyone is married.**

**⇒ ∃ boy B who is not married at termination.**

Remember that we know we terminate by Theorem 1.

**Q:** what had to have happened to boy B to not be married at the end?

**A:** **⇒ B was rejected by every girl.**

**Q:** what does this mean about the status of every girl?

**A: ⇒ Every girl has a suitor** at the end that she likes better than B.

**Q: Why?**

**A: (By Lemma 1)**

**Q:** so what do we know about every girl on termination day?

**A:** **⇒ Every girl is married**

**Q:** what does this mean about every boy, including boy B?

**A: ⇒ Every boy (including B) is married #**

So our assumption was wrong and so it must be that everyone was married by TMA.

Questions?

Proof by contradiction is a powerful technique!

Next, we’ll prove the main result – namely that TMA always produces stable marriages.

**Thm 3: TMA produces a stable matching.**

**Proof: By contradiction.**

**Assume TMA does NOT produce a stable matching.**

**⇒ ∃ rogue couple – call them Bob and Gail.**

Because Bob and Gail are rogue, they are not married to each other by the algorithm. There are now two cases to consider depending on whether or not Bob serenaded Gail during the algorithm.

**Case 1: Bob serenaded Gail.**

**Q:** What must have happened in this case? Remember that Gail did not marry Bob.

**A: ⇒ Gail rejected Bob** at some point during the process. Otherwise she would have ended up being married to him at the end.

**Q:** OK, so what do we know about how Gail feels about her husband given that she had rejected Bob at some point?

**A: ⇒ Gail prefers her husband to Bob (by Lemma 1)**

**⇒ Gail and Bob are not rogue! #** So we have a contradiction in this case.

We’re not done yet – still need to show that Case 2 leads to a contradiction.

**Case 2: Bob never serenaded Gail.**

**Q:** What must have happened for Bob to have never gone after Gail?

**A:** Bob never got far enough down his list to serenade Gail. So that means that

**⇒ Bob married someone that he prefers to Gail.**

**Q: S**o can Bob and Gail be rogue?

**A:** No, Bob likes his wife better.

**⇒ Gail and Bob are not rogue! #**

So this case also leads to a contradiction which means that TMA must produce a stable matching and the proof is done. 

Questions?

Ok, so we’ve shown that TMA terminates in at most N2+1 days, gets everyone married and no rogue couples. Very nice result. We are essentially done. But there is one issue left to think about when evaluating the usefulness of the algorithm.

**Q:** Any idea what that might be?

**A:** Fairness

Who is better off in TMA

1. boys
2. girls
3. neither – unclear?

**VOTE**

Seems like maybe girls – they sit back and get best of suitors while boys do all the work and get worst of girls courted. On the other hand, the boys do try to get their 1st choice and the girls do have to wait to be asked – they may never see Mr. Right.

This is fundamental question in sociology: who has the power in courtship? Proposers or acceptors?

Seems hard to answer, but it turns out that we can show in a very formal way that the boys have all the power here. To formalize this, we need to some definitions:

For any collection of preference lists,

**Let *S* = set of all stable matchings**

We know that *S* is not empty.

**Q:** Why?

A: TMA gives 1 stable paring **⇒ (*S* ≠ )**

**Def: For each person P, the realm of possibility for P is** the set of mates that you can have in a stable matching

**{Q | ∃ M ∈ *S*, {P, Q} ∈ M }.**

**Explain complicated notion**

i.e. Q is within realm of possibility for P iff ∃ stable matching where P marries Q.

**Q:** Can realm of possibility ever be empty?

**A:** No. ∃ stable matching so there is always someone in your realm of possibility.

Now it may be that some people are not within the realm of possibility. Similar concept to when your parents tell you that so-and-so is “not in your league” when it comes to marriage. **☺**

In our case, some mates might be out of the question since no stable pairings possible if you married them. If you married them, they might run off and form a rogue couple.

**Ex:**

**Jennifer**

**Brad**

**2**

**1**

**1**

**1**

**Angelina**

For example, Brad is just not realistic for Jennifer since if you ever pair them, Brad and Angelina will form rogue couple. No stable matching with Brad paired to Jennifer. So Brad is not in the realm of possibility for Jenn & vice-versa.

Questions?

Now that we have this notion, we can define who is your optimal mate and who is your pessimal mate.

**Def: A person’s optimal mate is his/her favorite from the realm of possibility.**

Must exist since ∃ at least one person in the realm of possibility.

**Def: A person’s pessimal mate is his/her least favorite from the realm of possibility.**

Ok now here is a pair of shocking results.

**Thm 4: TMA pairs every boy with his optimal mate!**

**Thm 5: TMA pairs every girl with her pessimal mate!**

Wow! Talk about a sexist algorithm! It is true that boys have to work harder in courtship – serenade under balcony & suffer rejection. While girls sit back and choose. But these theorems show that the pain and effort pays off in the end.

You will prove these theorems in recitation tomorrow.

**Stop if no time**

TMA arises in all sorts of applications. Perhaps the most famous application is in matching fresh MDs to residency programs.

How many pre-meds here?

When you are in 4th year of med school – you fill out a form with your top 20 choices for residency program. Teaching hospitals do the same thing with their top choices for doctors.

Then the data is fed to an algorithm – which matches doctors to a hospital. Doctors learn their assignment on Match Day: huge event – determines your life/career.

The Algorithm that is used is TMA. They use TMA so there won’t be any situations where a doctor and a hospital prefer each other to their choices since that would invite a rogue couple situation where they would try to swap.

Q: Now who do you suppose gets the role of the boys here?

A: hospitals – they want to get their optimal choices.

Now in this case, there are multiple girls for every boy since many doctors go to each hospital. The proof that everything works is a bit harder to what we did in class, but it follows along the same lines.

Not surprisingly, TMA is also used by some of the largest dating agencies.

At Akamai, we use a variation of TMA to assign web traffic to our servers. In the early days, we used min cost flow – but that got to be too slow as number of servers and amount of traffic increased and we switched to TMA since it is fast and can be run in a distributed manner. In this case, web traffic is boys and web servers are girls. Servers have preferences based on latency and packet loss. Traffic has preferences based on cost of bandwidth and co-lo or vice versa depending on which we need to prioritize.