h_1, h_2 r_1 h_1 r_2, r_1

h_2, h_1 r_2 h_2 r_1, r_2

What happens if you run Gale Shapley on the example above.
Does it matter which of the "free riders" goes first?
What happens if the horses propose to the riders?

Initially all r in R and h in H are free
While there is a free r
    Let h be highest on r's list that r has not proposed to
    if h is free, then match (r, h)
    else //h is not free
suppose (r', h) are matched
        if h prefers r to r′
            unmatch (r', h)
            match (r, h)

More Stable Matchings

CSE 417 Winter 2021
Lecture 3

Announcements

If the news was overwhelming this week, you're not alone on that.

You can watch asynchronously if you need to.

This week's canvas quizzes delayed to Wednesday.

Starting (sometime) next week, we'll use preassigned breakouts (so you don't have to introduce yourself every time). Announcement coming on Ed today. Please follow directions **even if you're usually asynchrounous.**

Goals for Today

Does it matter which free rider we choose?

Does it matter what order

How many stable matchings can there be?

What can one do in practice?

At the end: Introduction to induction.

Multiple Stable Matchings

Suppose we take our algorithm and let the horses do the "proposing" instead.

h_1 , h_2 r_1 h_1 r_2 , r_1

h_2 , h_1 r_2 h_2 r_1 , r_2

We got a different answer…

What does that mean?

Proposer-Optimality

Some agents might have more than one possible match in a stable matching. Call these people the "feasible partners."

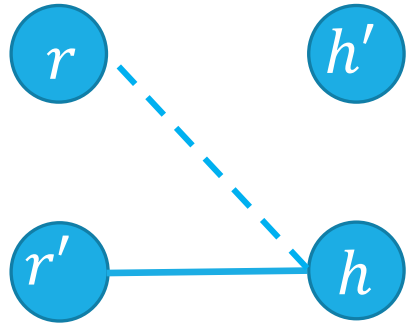
When there's more than one stable matching, there is a tremendous benefit to being the proposing side.

Proposer-Optimality

Every member of the proposing side is matched to their favorite of their feasible partners.

Proposer-Optimality

Every member of the proposing side is matched to the favorite of their feasible partners.



Let's prove it – again by contradiction

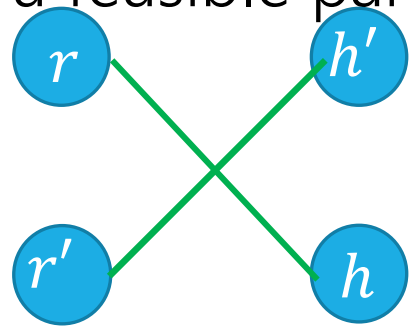
Suppose some rider is not matched to their favorite feasible partner.
Then some r must have been the **first** to be rejected by their favorite feasible partner, h. (Observation A)
And there is an r' that h (temporarily) matched to causing that rejection.

Let M' be a stable matching where (r, h) are matched. The rider r' is matched to some h'.

What can we say about r'? They had never been rejected by a feasible partner. So they prefer h to h'.

And h prefers r' to r (by the run of the algorithm).

But then (r', h) are a blocking pair in M'!



Implications of Proposer Optimality

Proposer-Optimality

Every member of the proposing side is matched to their favorite of their feasible partners.

We didn't specify which rider proposes when more than one is free

Proposer-optimality says it doesn't matter! You always get the proposer-optimal matching.

So what happens to the other side?

Chooser-Pessimality

A similar argument (it's a good exercise!), will show that choosing among proposals is a much worse position to be in.

Chooser-Pessimality

Every member of the choosing (non-proposing) side is matched to their least favorite of their feasible partners.

Some More Context and Takeaways

Stable Matching has another common name: “Stable Marriage”

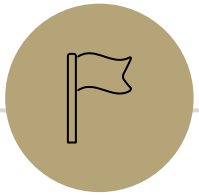
The metaphor used there is “men” and “women” getting married.

When choosing or analyzing an algorithm think about everyone involved, not just the people you’re optimizing for; you might not be able to have it all.

Stable Matchings always exist, and we can find them efficiently.

The GS Algorithm gives proposers their best possible partner

At the expense of those receiving proposals getting their worst possible.



Practical Concerns

How Many Stable Matchings?

We've seen there is always at least one stable matching.

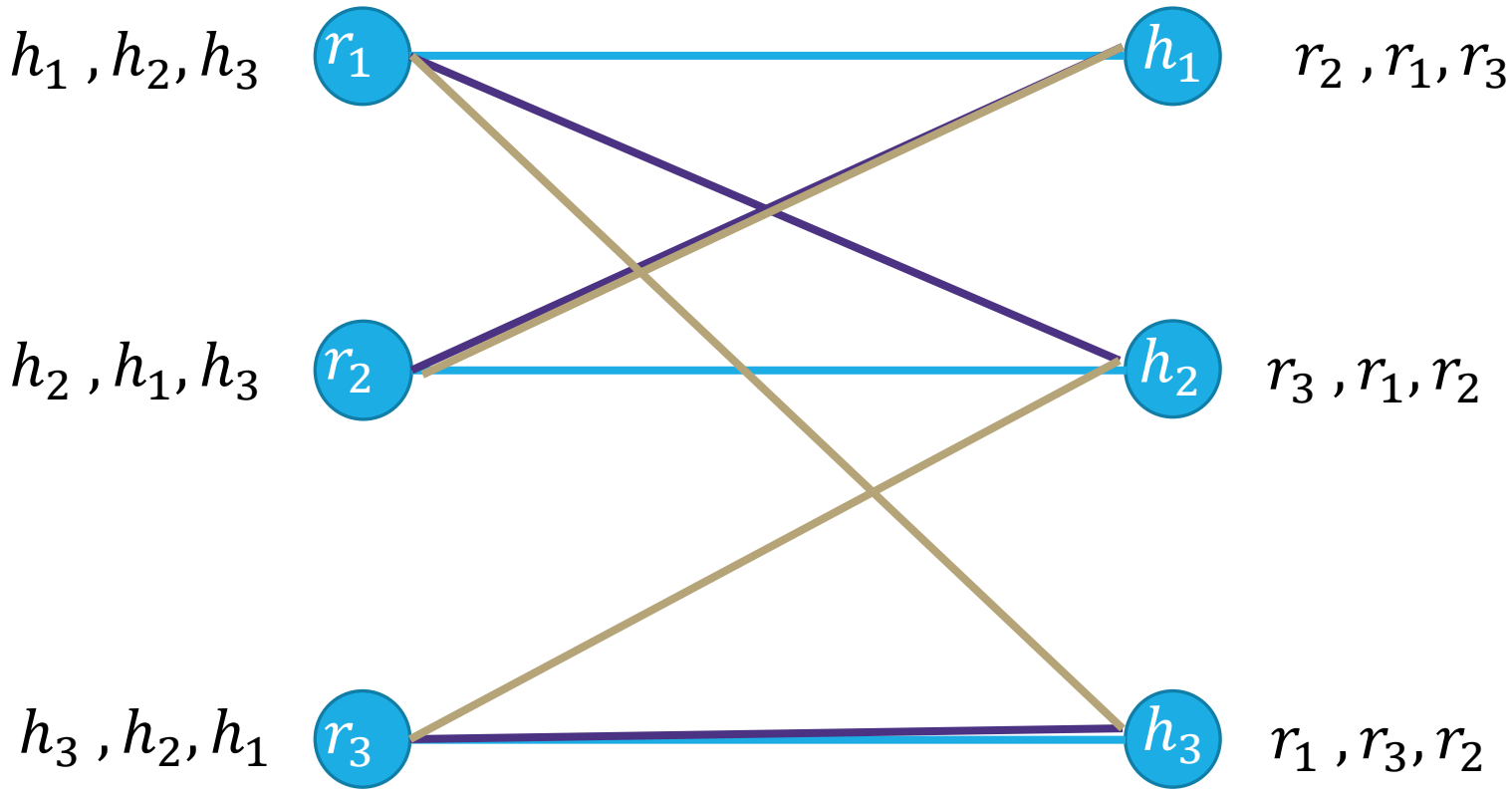
We've seen sometimes there are at least two stable matchings.

Can there be only one?

Can there be more than two (i.e. can there be stable matchings not found by Gale-Shapley?)

Why do we care? GS advantages one side or the other. If other matchings are available, maybe those are "more balanced"

More than two?



What about when n increases?

Ok, ok, is 3 the maximum?

What we really care about is:

As n increases, can the number of stable matchings grow quickly enough that it's not possible to examine them all?

How do we justify that. I can't just give you one instance…

I need to give you a **family** of instances.

Instructions so that you can build an instance for larger and larger n.

What about when n increases?

Generalize the idea from the last slide.

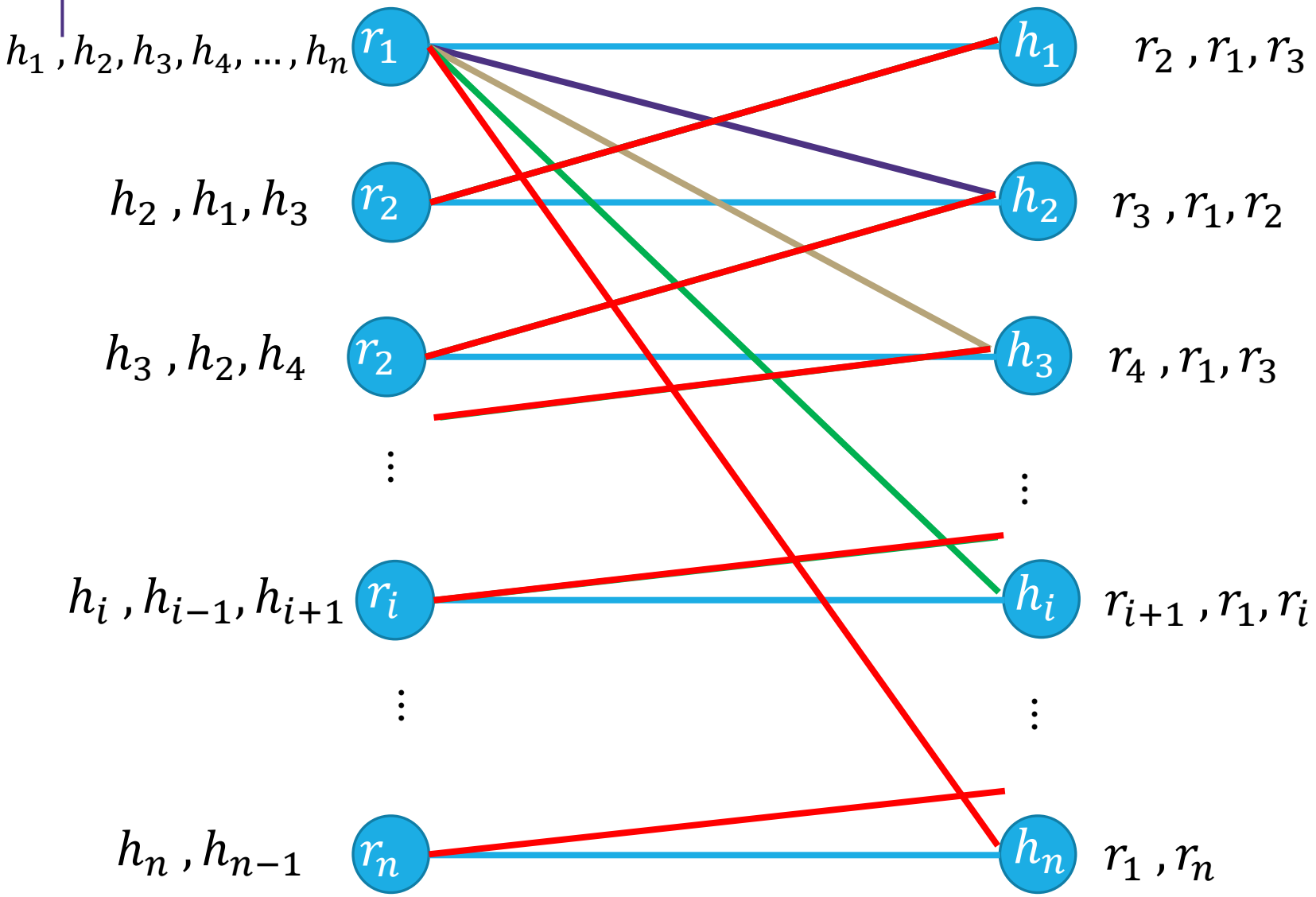
r_1 has list: $h_1, h_2, h_3, \ldots, h_n$

r_i has their list start h_i, h_{i-i}, h_{i+1} (for $i > 1$)

h_i has its list start: r_{i+1}, r_1, r_i (for $i < n$)

h_n has its list start: $r_1, \mathrm{r_n}$

What about when n increases?



Rider optimal: match r_i to h_i

Switch (r_1, h_2) (r_2, h_1)
Still stable -- h's that changed got happier, no new blocking pairs created.

For some j: Match (r_1, h_j)
Up to j: (r_i, h_{i-1})
After j: (r_i, h_i)
Still stable (for same reason)

At the end:
Match (r_i, h_{i-1}) and (r_1, h_n)
Still stable (for same reason)

How high can that go?

One of your homework problem asks you to show that the number of stable matchings can grow exponentially.

i.e. there are stable matching instances with n riders and n horses with c^n stable matchings (where $c > 1$).

The exact maximum is unknown!

When there's more than two…

Can we find them all?

Well…we could just check all $n!$ possible matchings to see if each is stable. That's only $O(n! \cdot n^2)$ time.

That's really slow.

There's a better way!

There's an algorithm that runs in $O(n^2 + nS)$ time to print all S possible stable matchings.

Beyond the scope of this course,

If we can find them all…

…could we just find the “median” one?

It’s not clear what “median” means.

There is a reasonable definition (ask Robbie after)…but it’s NP-hard to find the “median” matching

What does that mean: set a reasonable expectation – you shouldn’t expect to always find the median stable matching (but you can sometimes!)

Can there be only one?

If Gale-Shapley returns the same matching regardless of which side proposes, then there must be only one stable matching.

Proof:

For every agent, when it proposes, it gets its favorite feasible partner (proposer-optimality) and when it chooses, it gets its least favorite feasible partner (chooser-pessimality). So if that agent is the same, it only has one feasible partner. If every agent has only one possible feasible partner, then there is only one possible stable matching.

Can we make an example like that? Yes! r_i and h_i have each other as first choices for all i.

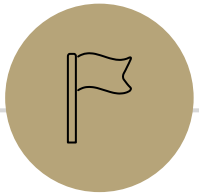
So what should you do?

Choosing a GS run could give a significant advantage to one side, but…

Sometimes the advantage is minimal.

The NRMP changed from a hospital-proposing based algorithm to a resident-proposing based algorithm. Only about 0.1% of residents got a different assignment when the algorithm changed.

In practice, there might be few enough stable matchings to just see them all and pick one (on average, there are only $O(n \log n)$ stable matchings).



Induction

Induction

We proved that Gale-Shapley produces a matching with no blocking pairs with proof by contradiction.

We could also have used proof by induction.

Induction is useful when you have a claim that is true "step-by-step"

Our big goal is "no blocking pairs at all" – step-by-step, we can say:

After i iterations of the while-loop of GS, there are no blocking pairs among (tentatively) matched agents.

i.e. no pairs where **both** agents are tentatively matched.

Induction

Show a claim step-by-step.

"Base Case" – show the calculation "starts" in the right place (usually that variables store the right values, or some property is true about what we've calculated so far).

If the variables are right before step k, then they are right after step k.

"inductive hypothesis" – if our property is true before step k

"inductive step" – then it is still true after step k.

Induction

Prove a loop does the right thing.

Before the loop starts, everything is right.

Each time through, if the variables start with the right information, then they are updated correctly.

Therefore, after we exit the loop, we have the right answer.

Prove recursive code works

The base case of the recursion produces the right value.

If the recursive calls we make produce the right value, then we return the right value.

Therefore, the first recursive call also produces the right answer.

Induction in 5 easy(?) steps

1. Define $P(n)$. State that your proof is by induction on n.

2. Base Case(s): Show the smallest value $P(b_{min})$ is true.

3. Inductive Hypothesis: Suppose $P(b_{min}) \wedge P(b_{min}+1) \wedge \cdots \wedge P(k)$ for an arbitrary $k \geq b_{max}$. (The smallest value of k assumes **all** bases cases, but nothing else)

4. Inductive Step: Show $P(k+1)$ (i.e. get $[\mathrm{P}(\mathrm{b}_{min} \wedge \cdots \wedge P(k)] \rightarrow P(k+1)$)

5. Conclude by saying $P(n)$ is true for all $n \geq b_{min}$ by the principle of induction.

Things to check for

$P(n)$ should:
1. be Boolean valued (i.e. output "true" or "false)

2. Take in an integer.

3. If you knew $P(n)$ holds for all n, you should know your main claim.

If your inductive step goes back exactly s steps every time, then you need s base cases. (good rule of thumb: if you're analyzing recursive code with multiple base cases, your induction proof needs multiple base cases)

Be careful with your IH! You want to suppose everything from the base case(s) up to some fixed value k.

Claim: Gale-Shapley produces a matching without blocking pairs

$P(n)$ after n iterations of the while-loop of GS, there are no blocking pairs among (tentatively) matched agents.

Base Case: After $\mathbf{0}$ iterations no one is matched, so there are no blocking pairs among tentatively matched agents (because there aren't any).

IH: Suppose that after each of the first k iterations of the while-loop of GS, there are no blocking pairs among (tentatively) matched agents.

IS: Want to show after the $(k+1)^{\mathrm{st}}$ iteration, there are no blocking pairs among tentatively matched agents.

…

Claim: GS produces a matching without blocking pairs

IH: Suppose that after each of the first k iterations of the while-loop of GS, there are no blocking pairs among (tentatively) matched agents.

IS: Want to show after the $(k+1)^{\text{st}}$ iteration, there are no blocking pairs among tentatively matched agents.

Proposal Rejected

OR Proposal accepted because h preferred r to its previous tentative match

OR Proposal accepted because first one h received

…

Claim: GS produces a matching without blocking pairs

IH: Suppose that after each of the first k iterations of the while-loop of GS, there are no blocking pairs among (tentatively) matched agents.

IS: Want to show after the $(k+1)^{\text{st}}$ iteration, there are no blocking pairs among tentatively matched agents.

During the $(k+1)^{\text{st}}$ iteration, we have one proposal from a rider r.
If r is rejected, the matching doesn't change, so there are no blocking pairs by inductive hypothesis.

OR proposal accepted because h preferred r to previous tentative match.

OR proposal accepted because first one h received

…

Claim: GS produces a matching without blocking pairs

IH: Suppose that after each of the first k iterations of the while-loop of GS, there are no blocking pairs among (tentatively) matched agents.

IS: Want to show after the $(k+1)^{\text{st}}$ iteration, there are no blocking pairs among tentatively matched agents.

During the $(k+1)^{\text{st}}$ iteration, we have one proposal from a rider r.
If r is rejected, the matching doesn't change, so there are no blocking pairs by inductive hypothesis.

Otherwise, r is tentatively matched to h. We claim r doesn't form a blocking pair: since it proposed to h, any horse it prefers to h already rejected it, so by Observation C, they prefer their partner to r. And h won't form a blocking pair either –
if it was matched, by IH it preferred its former match to any other tentative match. And it just decided it prefers r.

OR proposal accepted because first one h received

…

Claim: GS produces a matching without blocking pairs

IH: Suppose that after each of the first k iterations of the while-loop of GS, there are no blocking pairs among (tentatively) matched agents.

IS: Want to show after the $(k+1)^{\text{st}}$ iteration, there are no blocking pairs among tentatively matched agents.

During the $(k+1)^{\text{st}}$ iteration, we have one proposal from a rider r.
If r is rejected, the matching doesn't change, so there are no blocking pairs by inductive hypothesis.

Otherwise, r is tentatively matched to h. We claim r doesn't form a blocking pair: since it proposed to h, any horse it prefers to h already rejected it, so by Observation C, they prefer their partner to r. And h won't form a blocking pair either –
if it was matched, by IH it preferred its former match to any other tentative match. And it just decided it prefers r.
If it wasn't matched, it hadn't received a proposal from any matched rider, the matched riders all prefer their current matches, so it can't form a blocking pair with them.

Wrapping up the proof

By the principle of induction, $P(i)$ holds for all i.

In particular, after the last iteration of the while-loop of GS, there are no blocking pairs among (tentatively) matched agents.

Since we also produce a perfect matching, there are no blocking pairs at all! So we produce a stable matching.

Want more induction practice?

Lots of practice materials on the webpage. Look at the resources tab.