CSE 417 Autumn 2025

Lecture 4: Gale–Shapley analysis

Glenn Sun

Announcements

HW1 due Friday @ 11:59pm!

HW1 LaTeX template fix: Check Ed!

Continue to get those Concept Checks in on time, please 🙂

Review of Gale–Shapley

Gale–Shapley algorithm

```
while there is a free proposer p ∈ P do
    Let r be the top remaining person on p's preference list.
    if r is also free then
        Have r accept p.
    else if r is paired but prefers the new proposer p then
        Have r accept p and reject their current match p′.
    else (if r is paired and prefers their current match)
        Have r reject p.
return all matches
```

Proposer optimality

Proposer optimality theorem: The Gale–Shapley algorithm always finds the unique stable matching that is both **best for proposers** and **worst for receivers**.

“best”: Every proposer is happier in this matching than any other stable matching (or equally as happy).

“worst”: Similar.

Proofs with while loops

Proofs with while loops (1/4)

Input: A positive integer n

Goal: An integer k such that $(k-1)^2 < n \le k^2$

1. Let $k = 1$.
2. while $k^2 < n$ do
3. Update $k = k + 1$.
4. return k

Proofs with while loops (2/4)

"Loops terminate": For a while loop, we give an **upper bound on the number of iterations.**

Scratch work (working backwards):

- We need $\boldsymbol{k^2 \geq n}$ for termination.
- This happens when $\boldsymbol{k = \lceil \sqrt{n} \rceil}$.
- At the end of the $\boldsymbol{i}$th iteration, $\boldsymbol{k = i + 1}$. (Loop invariant!)
- Thus, we should be good after $\boldsymbol{\lceil \sqrt{n} \rceil - 1}$ iterations.

Proofs with while loops (3/4)

“Loops terminate”: For a while loop, we give an **upper bound on the number of iterations.**

Claim. The loop terminates within $\lceil \sqrt{n} \rceil - 1$ iterations.

Proof. It is a loop invariant that after i iterations, we have $k = i + 1$.

After $\lceil \sqrt{n} \rceil - 1$ iterations, $k = \lceil \sqrt{n} \rceil$. Thus $k^2 = \lceil \sqrt{n} \rceil^2 \geq n$, so the while loop exits.

Proofs with while loops (4/4)

To prove "meets specification" with a while loop, it's usually not enough to use loop invariants. Must use

loop invariant + while exit condition

Example:

- loop invariant: $(\boldsymbol{k}-\mathbf{1})^{2}<\boldsymbol{n}$
- while exit condition: $\boldsymbol{k}^{2} \geq \boldsymbol{n}$

Recall final goal: an integer $\boldsymbol{k}$ such that $(\boldsymbol{k}-\mathbf{1})^{2}<\boldsymbol{n} \leq \boldsymbol{k}^{2}$

Proving Gale–Shapley correct

Three requirements for correctness

"No exceptions": In line 2 when p picks the next top person on their preference list, p has not yet exhausted the entire list.

"Loops terminate": Every proposer gets eventually matched.

"Meets specification": The final set of matches is a perfect matching with no unstable pairs.

Proving no exceptions (1/7)

Claim. In line 2 when p picks the next top person on their preference list, p has not yet exhausted the entire list.

Proof. Suppose for contradiction p has exhausted the entire list.

In other words, they have proposed to everyone and also got rejected by everyone.

Q: What must be true if p was rejected by everyone?

Proving no exceptions (2/7)

while there is a free proposer $p \in P$ **do**
    Let r be the top remaining person on p's preference list.
    if r is also free **then**
        Have r accept p.
    else if r is paired but prefers the new proposer p **then**
        Have r accept p and reject their current match p'.
    else (if r is paired and prefers their current match)
        Have r reject p.
return all matches

A: Every r was already paired when they rejected p!

Proving no exceptions (3/7)

We know: Every r was already paired when they rejected p.

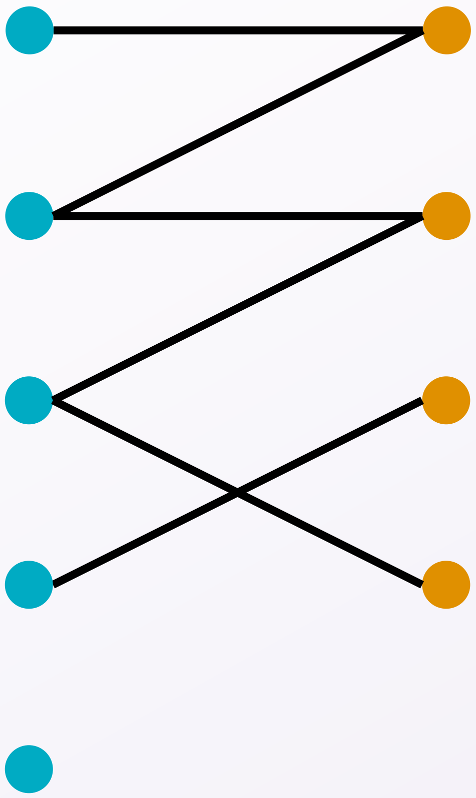
Loop invariant 1: At the end of every iteration, if r was ever paired to someone in any previous iteration, it is still paired to someone.

- Essentially because we only unpaired r in this iteration if we immediately paired them with someone else.

Conclusion: Every r is still paired.

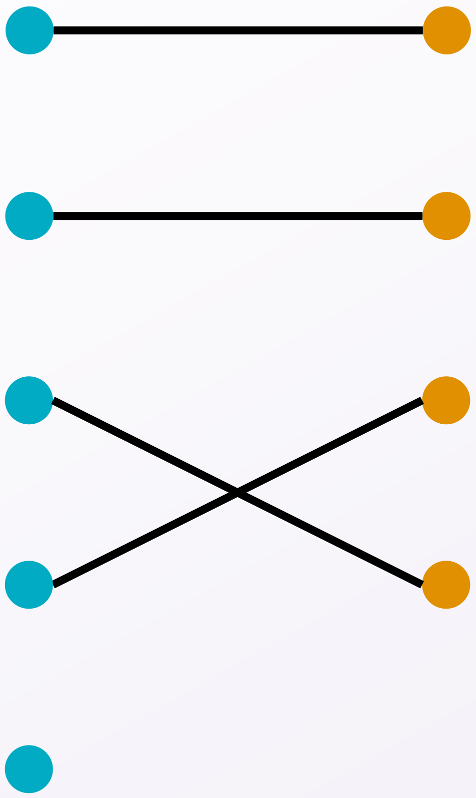
Proving no exceptions (4/7)

We know: Every r is still paired.



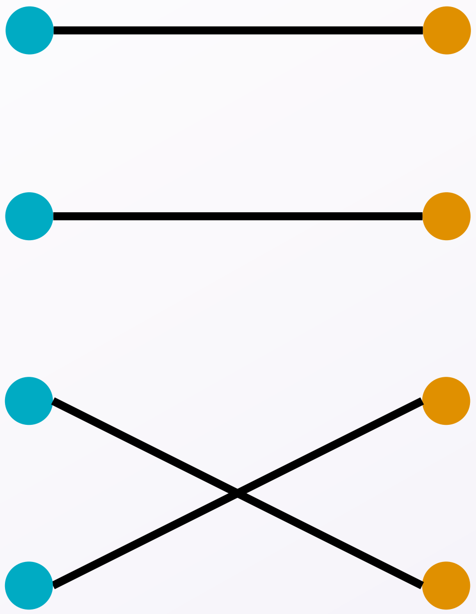
Proving no exceptions (5/7)

Loop invariant 2: After every iteration, each person is matched to at most one other person.



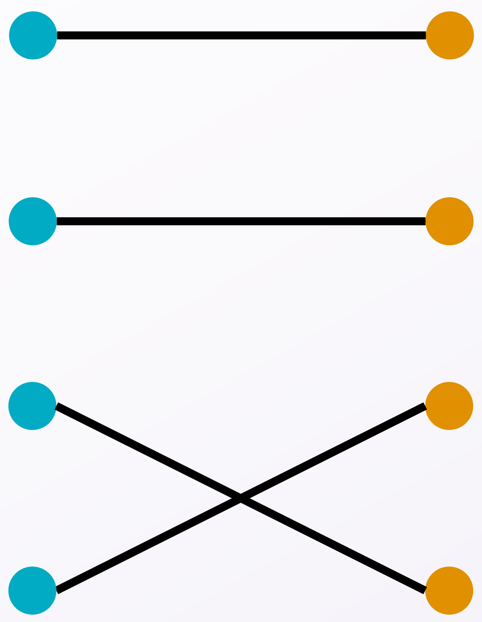
Proving no exceptions (6/7)

But $|P| = |R| = n$ (equal number of proposers and receivers).



Proving no exceptions (7/7)

Every proposer must be paired, contradiction with p being free.



Proving loops terminate

Claim. Every proposer gets matched within n^2 iterations.

Proof. There are n^2 possible proposals (each $p \in P$ to each $r \in R$).

Because line 2 always picks a new proposal and never throws an error, the while loop must end within n^2 iterations.

Proving perfect matching (1/5)

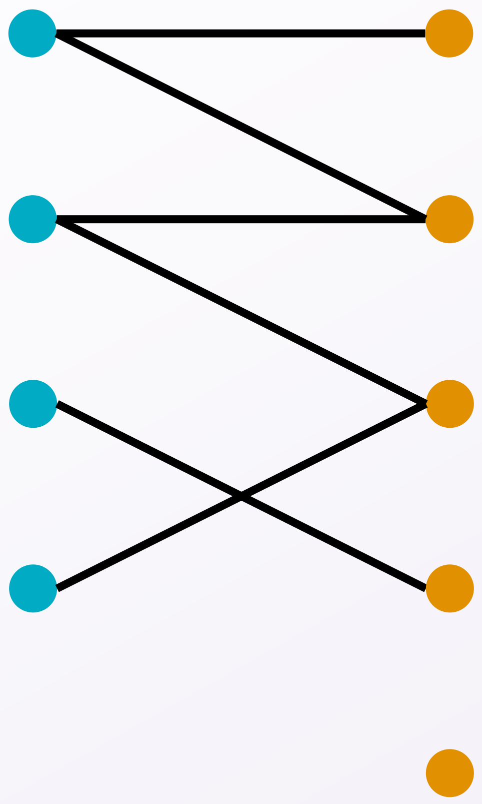
Remember, when there is a while loop, correctness should use both a **loop invariant** and the **while exit condition**.

Claim. The output is a perfect matching.

Proof. Very similar to before — in following slides.

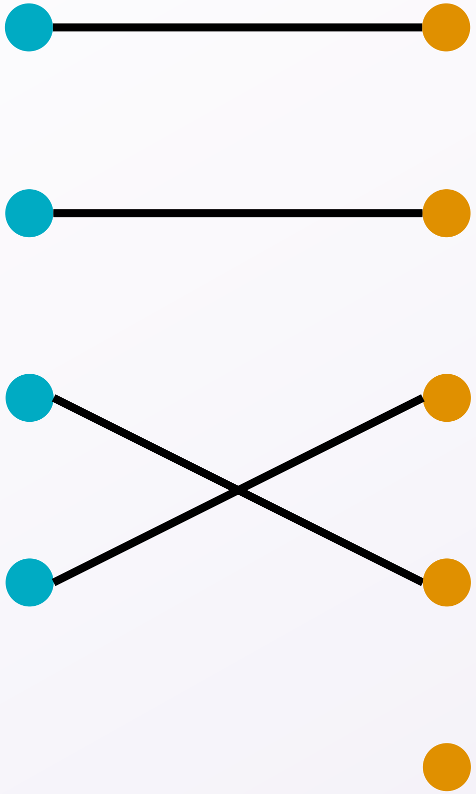
Proving perfect matching (2/5)

While exit condition: Every p is paired.



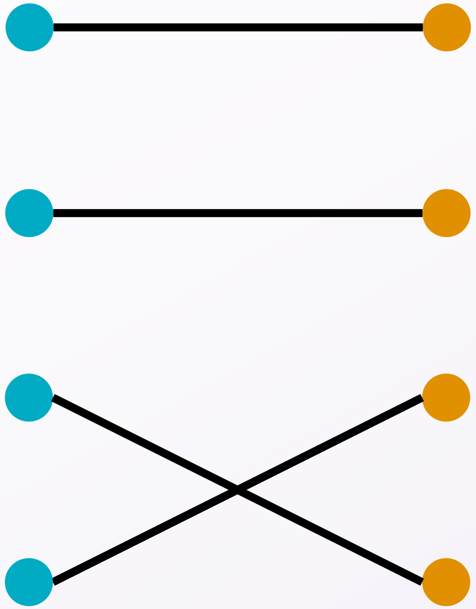
Proving perfect matching (3/5)

Loop invariant 2: After every iteration, each person is matched to at most one other person.



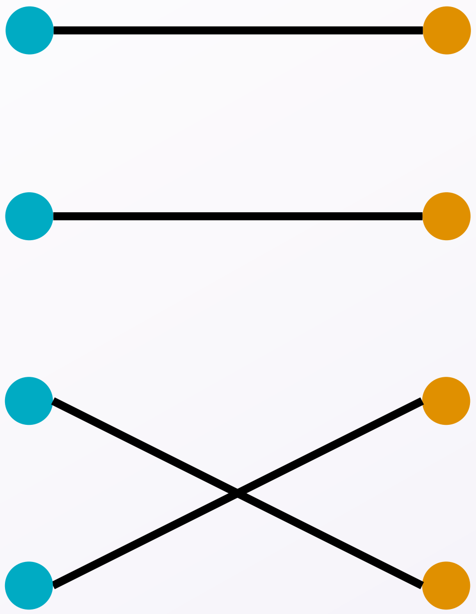
Proving perfect matching (4/5)

But $|P| = |R| = n$ (equal number of proposers and receivers).



Proving perfect matching (5/5)

Thus, everyone is paired with exactly one other, so this is a perfect matching.



Proving no unstable pairs (1/5)

Claim. The output has no unstable pairs.

Proof. Let (p, r) be any proposer-receiver pair.

Case 1: (p, r) is in the output matching.

Then (p, r) is not unstable.

Proving no unstable pairs (2/5)

Claim. The output has no unstable pairs.

Proof. Let $(\boldsymbol{p}, \boldsymbol{r})$ be any proposer-receiver pair.

Case 2: $(\boldsymbol{p}, \boldsymbol{r})$ is not in the output matching.

Why would a pair not be matched? Two possibilities:

- **Case 2a:** $\boldsymbol{p}$ stopped proposing before getting to $\boldsymbol{r}$.
- **Case 2b:** $\boldsymbol{p}$ proposed to $\boldsymbol{r}$, but got rejected or later unpaired.

Proving no unstable pairs (3/5)

Case 2a: p stopped proposing before getting to r.

Because we output a perfect matching, p is matched to someone.

Because p proposes in order of preference, p must be matched to someone they prefer over r.

Then (p, r) is not unstable.

Proving no unstable pairs (4/5)

Case 2b: p proposed to r, but got rejected or later unpaired.

5. else if r is paired but $\boxed{\text{prefers the new proposer } p}$ then
6. Have r accept p and $\boxed{\text{reject their current match } p'.}$
7. else (if r is paired and $\boxed{\text{prefers their current match)}}$
8. Have r $\boxed{\text{reject } p.}$

Conclusion: At the time of rejection, r must have been matched to someone that they rank higher than p.

Proving no unstable pairs (5/5)

Case 2b: p proposed to r, but got rejected or later unpaired.

Loop invariant 3 ("trading up"): At the end of every iteration, if r is paired, it prefers its current match over all previous matches.

Because we output a perfect matching, r is matched to someone.

Because of "trading up", r is still matched to someone that they rank higher than p.

So (p, r) is not unstable.

What’s next?

We proved correctness. What other questions can we ask?

- How fast is the algorithm? —wait for Friday!
- What about many-to-one matchings? —on your HW2!
- *Which* stable matching does it produce? —our next topic
- Can people gain advantage by lying on their preferences lists?
  What about “stable roommates” (matching within one group)?
  etc. —if we have time

Proposer optimality/receiver pessimality

Proposer optimality theorem

Proposer optimality theorem: The Gale–Shapley algorithm always finds the unique stable matching that is both **best for proposers** and **worst for receivers**.

Say that a pair (p, r) are called **valid partners** if there is *some* stable matching where they are matched together.

Proof of proposer optimality (1/3)

By contradiction. Suppose the output is not proposer-optimal.

This means that for some p, the output matches (p, r), but r' is also a valid partner and p prefers $r' > r$.

$$p\!: \quad \ldots \; > \; r' \; > \; \ldots \; > \; r \; > \; \ldots$$

Proof of proposer optimality (1/3)

By contradiction. Suppose the output is not proposer-optimal.

This means that for some p, the output matches (p, r), but r' is also a valid partner and p prefers $r' > r$.

Let p' be the reason that r' rejected p.

$$p: \quad \ldots > r' > \ldots > r > \ldots$$
$$p': \quad \ldots > \ldots > r' > \ldots > \ldots$$

$$r': \quad \ldots > p' > \ldots > p > \ldots$$

Proof of proposer optimality (2/3)

Since (p, r') are valid partners, consider that matching.

Q: Where is p''s partner?

$$p\text{:} \quad \dots > r' > \dots > r > \dots$$

$$p'\text{:} \quad \dots > \dots > r' > \dots > \dots$$

$$r'\text{:} \quad \dots > p' > \dots > p > \dots$$

Proof of proposer optimality (2/3)

Since (p, r') are valid partners, consider that matching.

Q: Where is p''s partner?

If p''s partner is worse than r', then (p', r') is unstable, contradiction.

p: ... > r' > ... > r > ...
p': ... > ... > r' > ... > ...

r': ... > p' > ... > p > ...

Proof of proposer optimality (3/3)

What if p''s partner is better than r'?

$$
\begin{array}{lllllllll}
p: & \dots & > & r' & > & \dots & > & r & > & \dots \\
p': & \dots & > & \dots & > & r' & > & \dots & > & \dots \\
\\
r': & \dots & > & p' & > & \dots & > & p & > & \dots
\end{array}
$$

Proof of proposer optimality (3/3)

What if p''s partner is better than r'?

Common technique: Upgrade the original assumption so that the situation was the *first* time a valid partner was rejected.

Then p''s match in the new matching cannot be better than r'!

$$p\colon \quad \ldots > r' > \ldots > r > \ldots$$
$$p'\colon \quad \ldots > \ldots > r' > \ldots > \ldots$$

$$r'\colon \quad \ldots > p' > \ldots > p > \ldots$$

Proof of receiver pessimality (1/2)

By contradiction. Suppose the output is not receiver-pessimal.

This means that for some r, the output matches (p, r), but p' is also a valid partner and r prefers $p > p'$.

$$r\colon \quad \ldots > p > \ldots > p' > \ldots$$

$$p\colon \quad \ldots > \ldots > r > \ldots > \ldots$$

Proof of receiver pessimality (2/2)

Since (p', r) are valid partners, consider that matching.

r: … > p > … > p' > …

p: … > … > r > … > …

Proof of receiver pessimality (2/2)

Since (p', r) are valid partners, consider that matching.

By proposer optimality, r is the best possible match for p.

$$r\colon\quad \ldots > p > \ldots > p' > \ldots$$

$$p\colon\quad \ldots > \ldots > r > \ldots > \ldots$$

Proof of receiver pessimality (2/2)

Since (p', r) are valid partners, consider that matching.

By proposer optimality, r is the best possible match for p.

So (p, r) is an unstable pair, contradiction.

r: … > p > … > p' > …

p: … > … > r > … > …

Miscellaneous notes (just for fun)

Lying can help receivers (1/2)

Proposers					
1:	A	>	B	>	C
2:	B	>	A	>	C
3:	A	>	B	>	C

Receivers' true prefs					
A:	2	>	1	>	3
B:	1	>	2	>	3
C:	1	>	2	>	3

With true preferences, A gets their second choice.

Lying can help receivers (2/2)

Proposers					
1:	A	>	B	>	C
2:	B	>	A	>	C
3:	A	>	B	>	C

Receivers' true prefs					
A:	2	>	1	>	3
B:	1	>	2	>	3
C:	1	>	2	>	3

With A lying					
A:	2	>	3	>	1
B:	1	>	2	>	3
C:	1	>	2	>	3

With true preferences, A gets their second choice.

When A lies about second/third choices, A gets their top choice!

(Lying does not help proposers because of proposer optimality.)

Stable roommates problem (1/2)

One group of $2n$ people and preferences over all $2n-1$ others.

$$\begin{array}{llllll} 1: & 2 & > & 3 & > & 4 \\ 2: & 3 & > & 1 & > & 4 \\ 3: & 1 & > & 2 & > & 4 \\ 4: & 1 & > & 2 & > & 3 \end{array}$$

(1, 2) and (3, 4)	**(1, 3) and (2, 4)**	**(1, 4) and (2, 3)**
(2, 3) unstable	(1, 2) unstable	(1, 3) unstable

Stable roommates problem (2/2)

Stable solution may not always exist.

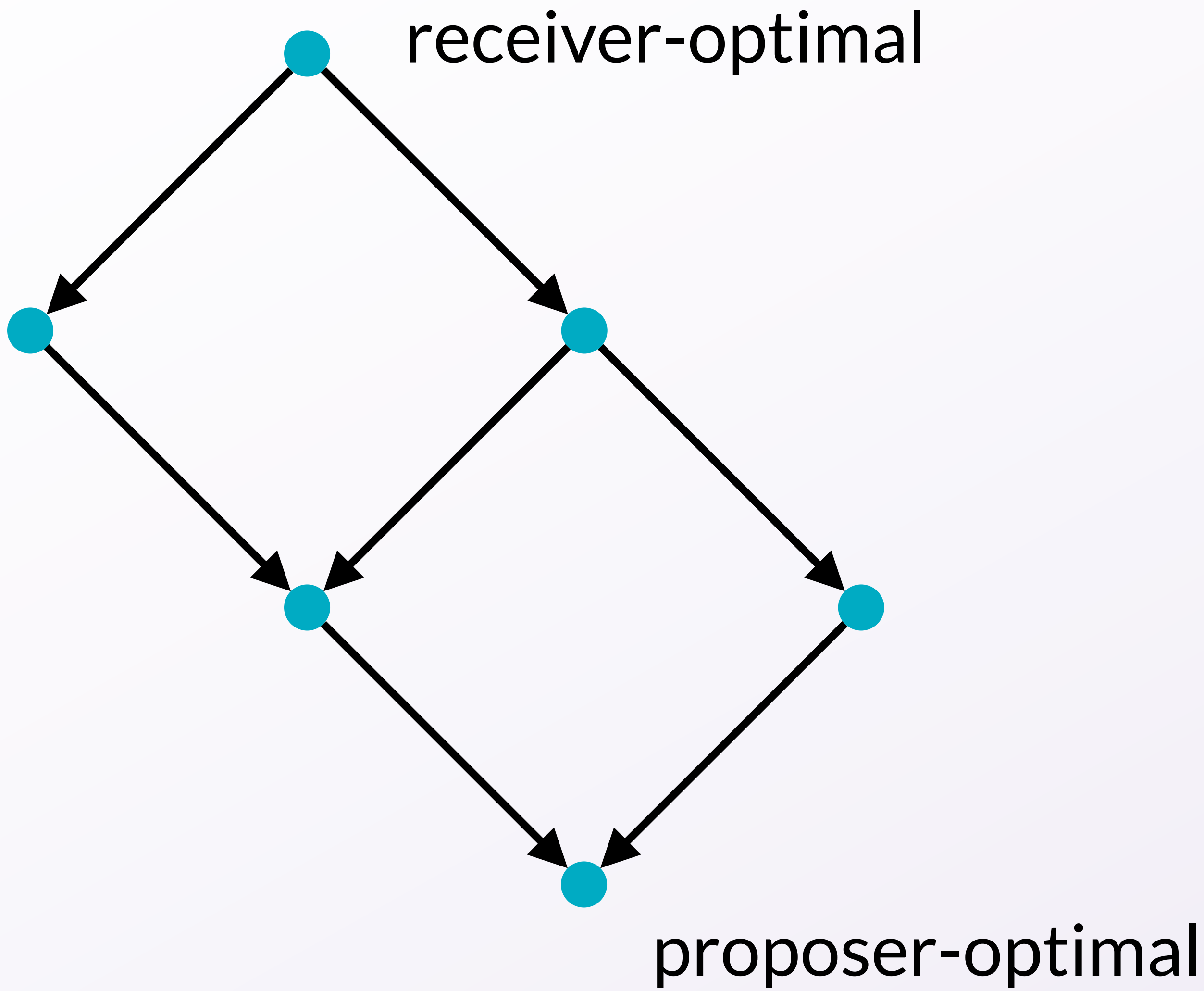
Irving’s algorithm (1985) extends Gale–Shapley to determine if a stable solution exists, and if so, find it!

Characterizing all stable matchings (1/2)

The set of all stable matchings forms an interesting structure called a **lattice**:

- Draw an arrow between from matching $\boldsymbol{M_1}$ to $\boldsymbol{M_2}$ if every proposer is happier in $\boldsymbol{M_2}$ over $\boldsymbol{M_1}$ (or equally happy).
- Then, every pair of matchings has a latest common ancestor and earliest common descendant.
- Gives alternate proof that a proposer-optimal matching exists!

Characterizing all stable matchings (2/2)



Final reminders

HW1 due Friday @ 11:59pm.

I have OH now-12:30pm:

- Meet at front of classroom, we'll walk over together
- CSE (Allen) 214 if you're coming later

Nathan has online OH 12–1pm:

- Link on Canvas/course website
- https://washington.zoom.us/my/nathanbrunelle