# CS 70 Su23: Lecture 4

Stable matching

Berkeley
UNIVERSITY OF CALIFORNIA

# The premise

- We have **n** candidates and **n** jobs
  - Each candidate can only work one job
- Each candidate has a preference list for jobs
- Each job(‘s hiring manager) has a preference list for candidates

# The problem

How can we pair the candidates and jobs such that it is **stable**? Formally:

- An **instance** is **n** candidates $\{C_1, C_2, \dots, C_n\}$ and **n** jobs $\{J_1, J_2, \dots, J_n\}$
- A **pairing** is a set of pairs $(C_1, J_{c\_1}), (C_2, J_{c\_2}), \dots, (C_n, J_{c\_n})$ where each candidate and job appear exactly once
- A **rogue couple** is a pair (C, J) *not in the pairing* where:
  - J is higher on C's preference list than C's current job
  - C is higher on J's preference list than J's current candidate
  - *Note: C and J are not matched with each other*
  - *A **rogue couple** only makes sense in the context of a **pairing**!*
- A pairing is **stable** iff there are no rogue couples

# Example

| **Candidate** | Job preference | | |
|---|---|---|---|
| **Alice** | 1 | 2 | 3 |
| **Bob** | 1 | 3 | 2 |
| **Cwop** | 2 | 1 | 3 |

| **Job** | Candidate preference | | |
|---|---|---|---|
| **1** | Alice | Cwop | Bob |
| **2** | Bob | Alice | Cwop |
| **3** | Alice | Bob | Cwop |

Consider the following pairing: **(Alice, 2), (Bob, 1), (Cwop, 3)**. Are these rogue couples?

- (Alice, 1)
  - **Yes.** Alice and 1 both prefer each other to their current matches
- (Bob, 1)
  - **No.** Bob and 1 are currently matched with each other (this isn't a trick question we'll ask you on an exam, just a concept check)
- (Cwop, 2)
  - **No.** Cwop prefers 2, but 2 does not prefer Cwop

# Does a stable matching always exist?

- Trying to stably pair roommates is not always possible (see notes)
  - Moral of the story: you can't just tweak pairs and hope it eventually converges
- Stable matching was an unsolved problem until (relatively) recently
  - Matching residents to hospitals used to be even more insane
  - Gale and Shapley in 1962 formally analyzed the algorithm that solves this
  - ok, 60 years may sound like an eternity, but math has been around *way* longer
- So what is this fancy algorithm?

# Propose and reject

Each day proceeds as follows:*

- **Morning:** each candidate applies to the first job on their list that has not rejected them
- **Afternoon:** each job looks at all applicants from the morning
  - they (say “maybe”) hold on to the candidate who is highest on their list, and reject the rest
- **Evening:** each candidate crosses off any job that rejected them in the afternoon from their list

Repeat until no candidates are rejected. Each job accepts who they are holding onto

* You can use any analogy/terminology you like here, so long as the concept holds

# Example

| **Candidate** | Job preference | | |
|---|---|---|---|
| **Alice** | 1 | 2 | 3 |
| **Bob** | ~~1~~ | 3 | 2 |
| **Cwop** | 2 | 1 | 3 |

| **Job** | Candidate preference | | |
|---|---|---|---|
| **1** | Alice | Cwop | Bob |
| **2** | Bob | Alice | Cwop |
| **3** | Alice | Cwop | Bob |

Day 1

- Morning:
  - Alice applies to 1, Bob applies to 1, Cwop applies to 2
- Afternoon:
  - 1 holds on to Alice, rejects Bob
  - 2 holds on to Cwop
- Evening:
  - Bob crosses 1 off their list

Day 2

- Morning:
  - Alice applies to 1, Bob applies to 3, Cwop applies to 2
- Afternoon:
  - 1 holds on to Alice
  - 2 holds on to Cwop
  - 3 holds on to Bob

No one is rejected; return (Alice, 1), (Bob, 3), (Cwop, 2)

# How does this work?

Does it work at all?

Let’s prove some stuff (hooray)

# Property: guaranteed termination

*Claim:* Propose-and-reject terminates (i.e. can never infinitely loop).

*Proof:* We will prove a stronger claim: Propose-and-reject terminates after at most $n^2$ days (where n is the number of candidates/jobs).

By definition, the algorithm terminates on the day where no one is rejected.

This means that each day (before the last), at least one job is crossed off someone's list.

The most number of days results from one job being crossed off a day.

*Therefore*, since each candidate's list is **n** jobs long, the most time is $\mathbf{n^2}$. //

# What we've shown so far

1. Propose-and-reject terminates

# Improvement lemma

*Claim:* if a job J is holding onto some candidate C on day *k*, every subsequent day, they will be holding onto someone they like at least as much as C.

- In other words, the improvement lemma states that jobs can only do better over time

*Proof: Suppose* J is holding on to C on day *k*. Let *S* be the set of all days > *k* where J is holding on to someone lower on their list than C (or no one).

*Claim: S* is empty.

*Proof by contradiction. Assume S* is non-empty. *Let* **i** be the first day after *k* where J is holding on to someone lower on their list than C (or no one).

# Improvement lemma

*Proof by contradiction. Assume S* is non-empty. *Let* **i** be the first day after *k* where J is holding on to someone lower on their list than C (or no one).

This means on day **i - 1**, J is holding onto someone (C') they like at least as much as C.

On day **i**, C' will apply to J. This means J will have someone at least as preferable as C.

*However*, on **i**, J has someone less preferable than C. →←

*Therefore,* the improvement lemma must be true. QED

- Why are we allowed to do this?
  - Well-ordering principle (which is the same as induction)

# Improvement lemma

This time, with induction:

*Proof by induction on* **i**, the number of days after *k*.

*Base case:* **i** = 0. On day *k*, J has someone at least as good as C. //

*Inductive hypothesis: suppose* for some **i** > 0, J has someone at least as preferable as C on day **k + i**.

# Improvement lemma

*Claim:* if a job J is holding onto some candidate C on day *k*, every subsequent day, they will be holding onto someone they like at least as much as C.

*Inductive step: consider day* **k + i + 1**.

Using our *IH*, we know J is holding on to someone (C') who is at least as preferable as C on day **k + i**.

On day *k + i + 1*, C' will apply to J (since J did not reject C'). This means J will be holding onto someone at least as preferable as C on day *k + i + 1*.

*Therefore,* we have shown the improvement lemma is true. QED

# What we've shown so far

1. Propose-and-reject terminates
2. Improvement lemma
    - why? turns out we'll need this (a lot)

# Property: valid output

*Claim:* Propose-and-reject terminates with a (valid) pairing.

*Proof by contradiction: assume* there exists some candidate C who does not have a job at the end of propose-and-reject.

This means C must have applied to every job.

Using the improvement lemma, every job has someone at least as preferable as C.

This means there are **n + 1** candidates. *However*, there are only **n** candidates. →←

*Therefore,* propose-and-reject must terminate with everybody paired up. //

# What we've shown so far

1. Propose-and-reject terminates
2. Improvement lemma
3. Propose-and-reject terminates with a pairing

# Property: stability

*Claim:* Propose-and-reject produces a stable pairing.

(Spoiler alert: the improvement lemma does all the heavy lifting again)

*Proof:* We will prove the claim that no rogue couples can exist.

*Consider* any output pair (C, J), and *suppose* C prefers J' to J. *Let* C' be the candidate J' is paired with by the algorithm.

Since J' appears before J on C's list, C must have applied to J' first and gotten rejected.

- (at which point, J' would be holding onto someone they prefer at least as much as C)

*Therefore*, C' is not C, and J' prefers C' more than C, and there can be no rogue couples. //

# What we've shown

1. Propose-and-reject terminates
2. Improvement lemma
3. Propose-and-reject terminates with a pairing
4. Propose-and-reject is stable

Let's keep going

(I know, I'm sorry)

# Property? uniqueness

*Claim:* The pairing output by propose-and-reject is unique.

What if we switched who does the proposing?

Let’s revisit our previous example (same preference lists)

# Property: uniqueness?

| **Job** | Candidate preference | | |
|---|---|---|---|
| **1** | Alice | Cwop | Bob |
| **2** | Bob | Alice | Cwop |
| **3** | ~~Alice~~ | Cwop | Bob |

Day 1

- Morning:
  - 1 offers Alice, 2 offers Bob, 3 offers Alice
- Afternoon:
  - Alice holds onto 1, rejects 3
  - Bob holds onto 2
- Evening:
  - 3 crosses Alice off their list

| **Candidate** | Job preference | | |
|---|---|---|---|
| **Alice** | 1 | 2 | 3 |
| **Bob** | 1 | 3 | 2 |
| **Cwop** | 2 | 1 | 3 |

Day 2

- Morning:
  - 1 offers Alice, 2 offers Bob, 3 offers Cwop
- Afternoon:
  - 1 holds on to Alice
  - 2 holds on to Bob
  - 3 holds on to Cwop

No one is rejected; return (Alice, 1), (Bob, 2), (Cwop, 3)

# What we've shown

1. Propose-and-reject terminates
2. Improvement lemma
3. Propose-and-reject terminates with a pairing
4. Propose-and-reject is stable
5. Multiple stable pairings can exist

# Optimality

What is the best you can do?

*Definition*: Let C be a candidate. The **optimal** job for C is the highest job on C's preference list that C can be paired with in any stable matching.

[If this code snippet helps; else ignore] Let S be the set of all stable pairings.

```
def get_optimal(candidate):
    jobs = [stable_pairing.get_job(candidate) for stable_pairing in S]
    return candidate.best(jobs)
```

# Pessimality

Similarly, the **pessimal** job for a candidate is the lowest job on C's preference list that C can be paired with in any stable pairing.

# Property: candidate-optimality

*Claim:* Propose-and-reject produces a pairing that is candidate-optimal.

*Proof by (strong) induction* on **k**, the number of days:

$\forall\ k \in \mathbb{N}$, no candidate is rejected by their optimal job on day k.

*Base case:* k = 1.

On the first day, each candidate applies to the first job on their list.

# Property: candidate-optimality

*Proof by contradiction: assume* some candidate C is rejected by their optimal job J on day 1 for some candidate C'. This means:

- J prefers C' to C (because J rejected C for C' on day 1)
- C' prefers J to any other job (because C' applied to J on day 1)

The definition of optimality tells us there must exist some stable pairing T with **(C, J)** as a pair. *Let* **(C', J')** be the pair in T that describes the match for C'.

From what we just showed, (C', J) is a rogue couple in T.

*However,* T is stable. →←

*Therefore,* our claim holds for k = 1. //

# Property: candidate-optimality

*Inductive hypothesis: suppose* for some k > 1, no candidate is rejected by their optimal job on any day before k.

*Inductive step:* consider day k.

*Proof by contradiction: assume* some candidate C is rejected by their optimal job J on day k for some candidate C'. This means:

- J prefers C' to C (because J rejected C for C' on day k + 1)
- C' prefers J at least as much as their optimal job J*
    - This is because C' has not been rejected by J* [*IH*]

# Property: candidate-optimality

[Trying to reason about C′ and J] C′ prefers J at least as much as J*

The definition of optimality tells us there must exist some stable pairing T with **(C, J)** as a pair. *Let* **(C′, J′)** be the pair in T that describes the match for C′.

J′ is at best J* (and also by construction, J′ cannot be J).

- recall that you cannot do better than your optimal match in *any* stable pairing

Thus, C′ prefers J to J′, and so **(C′, J)** is a rogue couple in T.

*However,* T is stable. →←

# Property: candidate-optimality

[Recap of what just happened] We showed[†]:

- C′ prefers J at least as much as J* using our *inductive hypothesis*
  - $J \geq J^*$
- C′ prefers J* at least as much as J′ by the definition of optimality
  - $J^* \geq J'$
- J cannot be J′ because C must be paired with J in T
  - $J \neq J'$
- C′ prefers J to J′
  - $J > J'$

---

*Therefore,* propose-and-reject produces a pairing that is candidate-optimal. //

# What we've shown

1. Propose-and-reject terminates
2. Improvement lemma
3. Propose-and-reject terminates with a pairing
4. Propose-and-reject is stable
5. Multiple stable pairings can exist
6. Propose-and-reject is proposer-optimal

# Property: job-pessimality

*Claim:* if a pairing is candidate-optimal, then it is job-pessimal.

*Let* T be the (candidate-optimal) pairing output by propose-and-reject, and *let* **(C, J)** be an arbitrary pair in T.

*Proof by contradiction: assume* C is not J's pessimal candidate. That is, there exists some stable pairing S such that **(C', J)** is a pair, and J prefers C' less than C.

By definition, J prefers C to C'. And because we know T is candidate-optimal, C prefers J to whoever they're paired with in S. Thus, **(C, J)** is a rogue couple.

*However,* S is stable. →←

*Therefore*, a candidate-optimal pairing must be job-pessimal. QED

# What we've shown

1. Propose-and-reject terminates
2. Improvement lemma
3. Propose-and-reject terminates with a pairing
4. Multiple stable pairings can exist
5. Propose-and-reject is proposer-optimal
6. Propose-and-reject is proposee-pessimal

"We are pleased to extend you this offer"

# Next class: sets and functions

After today, we will be exploring discrete math in earnest. Hopefully the concepts will make sense now that you know how to prove things!

**Quiz 1 is on Wednesday evening!**