Stable Matching

UC Berkeley – Summer 2025 – Steve Tate

Lecture 4

The Matching Problem

Two equal size sets, with goal to match one item from each set to the other

Jobs and candidates; residents and hospitals; customers and resources; tenants and rooms; students and discussion section seats; ...

Candidates list jobs in order of preference

Candidates

1	C	A	B
2	A	B	C
3	A	C	B

Jobs list candidates in order of preference

Jobs

A	1	2	3
B	1	2	3
C	2	1	3

How to match?

Matching: How to Match?

Jobs			
A	1	2	3
B	1	2	3
C	2	1	3

Candidates			
1	C	A	B
2	A	B	C
3	A	C	B

How should they be matched?

- Maximize total satisfaction
- Maximize number of first choices
- Minimize difference between preference ranks
- Ensure pairs don't want to switch

Stable Matching

Our Focus: Produce a matching where pairs don't want to switch

Definition: A **matching** is disjoint set of n job-candidate pairs.

Matching: (j, c)
(j^*, c^*)

Definition: A **rogue pair** (j, c^*) for this matching:

j and c^* prefer each other to their match

Definition: A matching is **stable** if there are no rogue pairs.

Terminology Note: Called a "rogue couple" in the notes.

Example Matchings

Jobs			
A	1	2	3
B	1	2	3
C	2	1	3

Candidates			
1	C	A	B
2	A	B	C
3	A	C	B

Matching 1

(A,1)
(B,2)
(C,3)

Any rogue pairs?

Rogue: (C,1)

C prefers 1 to 3
1 prefers C to A

Matching 2

(A,2)
(B,3)
(C,1)

Any rogue pairs?

No rogues!

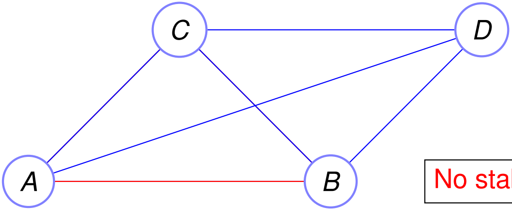
Existence of Stable Matchings

Questions we might ask:

- Does a stable matching always exist?
- How can one find a stable matching?
- How do conditions on the problem affect these questions?

Consider a single-set version: stable roommates.

A	B	C	D
B	C	A	D
C	A	B	D
D	A	B	C



No stable matchings!

The Propose and Reject Algorithm

Each Day:

1. Each job **proposes** to its favorite candidate that hasn't rejected it
2. Each candidate rejects all but their favorite job
   (which they "put on a string")
3. Each rejected job crosses out rejecting candidate from its list

Stop when all candidates have a job on a string.

Example

Jobs

A	~~1~~	2	3
B	~~1~~	~~2~~	3
C	~~2~~	1	3

Candidates

1	C	A	B
2	A	B	C
3	A	C	B

	Day 1	Day 2	Day 3	Day 4	Day 5
1	A , ~~B~~	A	~~A~~ , C	C	C
2	C	B , ~~C~~	B	A , ~~B~~	A
3					B

The Propose and Reject Algorithm

Each Day:

1. Each job **proposes** to its favorite candidate that hasn't rejected it
2. Each candidate rejects all but their favorite job
(which they "put on a string")
3. Each rejected job crosses out rejecting candidate from its list

Stop when all candidates have a job on a string.

What can we prove about this algorithm?

Does it terminate?

... produce a matching?

...... produce a *stable* matching?

Who does "better": jobs or candidates?

Termination

Does the algorithm always terminate?

Some important observations:

- Every day, each job is offered to one candidate
- On any non-terminating day, some candidate did not get an offer
  $\Rightarrow$ So some candidate got more than one offer
  Why? *Pigeonhole Principle*

On every non-terminating day, a job **crosses** an item off its list

Total size of lists? n jobs, n-length list n^2 items

Terminates in $\leq n^2$ steps!

Candidate Jobs Over Time – Ideas

Improvement Lemma: It just gets better for candidates

More precise statement: If on day t candidate c has a job j on a string, any job j' on candidate c's string for any day $t' > t$ is at least as good as j.

Recall our earlier example:

	Day 1	Day 2	Day 3	Day 4	Day 5
1	A , ~~B~~	A	~~A~~ , C	C	C
2	C	B , ~~C~~	B	A , ~~B~~	A
3					B

Candidate 2 has job B on string on day 2; job A on day 4.

Mapping to statement: $c = 2$, $j = B$, $t = 2$, $j' = A$, $t' = 4$.

Does candidate 2 prefer B or A?

Improvement Lemma says j' ... is at least as good as j: prefers A.

Can candidate 2 have A on a string at a later time?
$\Rightarrow$ Yes – "at least as good" includes same

Let's prove the Improvement Lemma...

Improvement Lemma – Slight Restatement and Proof

Improvement Lemma: Let candidate c have job j on a string at time t. Then for all $n \in \mathbb{N}$, if c has job j' on a string at time $t+n$, then j' is at least as good (to c) as j.

Proof: We proceed this by induction on n.

Base Case ($n = 0$): At time t, candidate c has job j on a string.

Induction Hypothesis: Assume the lemma holds for $n = k$, so if candidate c has job j' on a string at time $t+k$ then j' is at least as good as j.

Inductive Step: We prove the lemma holds at $n = k+1$: if candidate c holds job j'' on a string at time $t+k+1$ then j'' is at least as good as j.

Job j' and possibly other jobs make offers to candidate c at time $t+k+1$, and c selects its highest-ranked job j'' to put on a string.

The highest-ranked job is at least as good as each offer made, so j'' is at least as good as j'.

By the induction hypothesis we know that j' is ranked at least as high as j, so j'' is at least as good as j' which is at least as good as j.

Therefore j'' is at least as good as j, which completes the proof. $\square$

The P&R Algorithm Finds a Full Matching

Theorem: At the completion of the algorithm, every job is matched.

Proof: Assume for the sake of contradiction that some job j is not matched to a candidate at the end.

As long as a job is unmatched and there are candidates remaining in its list, the algorithm continues, so job j must have been rejected by all n candidates.

Let c be any candidate – since they rejected j's offer, they must have ended matched to a job they preferred (by the Improvement Lemma).

So *all* n candidates end up matched to a job.

A job can't be matched to more than one candidate, since while a candidate has it on a string it will not be offered to any other candidate.

So *all* n candidates end up matched to n different jobs, so every job has a matched candidate.

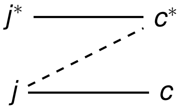
This contradicts our assumption that j was not matched, which completes the proof. □

The Matching Found By The P&R Algorithm is Stable

Theorem: The matching given by the Propose-and-Reject algorithm is stable.

Proof: Let j be any job, and let c be the candidate it is matched to by the algorithm. We show that j cannot be part of a rogue pair.

Say there is a candidate c^* (matched to j^*) that j prefers.



j prefers c^* to c.

Since j prefers c^* to c, it must have offered a job to c^* before c.

At some point, c^* rejected j's offer (since j moved on to c).

By the Improvement Lemma, c^* prefers its final job to j.

Therefore, (j, c^*) cannot be a rogue pair.

This is true for *any* job j, and *any* candidate c^* that it prefers, so there are no rogue pairs in the matching. □

How Do Jobs and Candidates Fare?

Definition: A **matching is *x*-optimal** if *x*'s partner
is its best partner in any stable matching.

Definition: A **matching is *x*-pessimal** if *x*'s partner
is its worst partner in any stable matching.

Definition: A **matching is job optimal** if it is *x*-optimal for **all** jobs *x*.

... and so on for job pessimal, candidate optimal, candidate pessimal.

Attention check!

The optimal partner for a job must be first in its preference list.
True or False? False!

Example: Every job lists candidate 1 first – all candidates list job A last
Can job A match to candidate 1 in any stable matching?

Subtlety here: Best partner in any stable matching.

Question: Is there always a job or candidate optimal matching?
j-optimal for each job *j* simultaneously? Unclear – let's figure it out!

Exploring Optimality With Examples

Example 1

Jobs

A	1	2
B	1	2

Candidates

1	A	B
2	B	A

Matching 1

(A,1)
(B,2)

Any rogue pairs?

No rogues!

A gets top choice
2 gets top choice

Matching 2

(A,2)
(B,1)

Any rogue pairs?

Rogue: (A,1)

A prefers 1 to 2
1 prefers A to B

Is Matching 1 optimal for B, who didn't get its top choice? Yes!

Unique stable matching, so optimal for B. Optimal for A, 1, and 2.

Also job-optimal, candidate-optimal, job-pessimal, and candidate-pessimal

Exploring Optimality With Examples
Example 2

Jobs		
A	1	2
B	2	1

Candidates		
1	B	A
2	A	B

Matching 1

(A,1)
(B,2)

Any rogue pairs?

No rogues!

Neither A nor B can improve

Matching 2

(A,2)
(B,1)

Any rogue pairs?

No rogues!

Neither 1 nor 2 can improve

Which is optimal for A? Matching 1
Which is optimal for B? Matching 1
Which is optimal for 1? Matching 2
Which is optimal for 2? Matching 2

Matching 1 is job-optimal

Matching 2 is candidate-optimal

Job-Optimality of Propose and Reject

Theorem: Propose and Reject produces a job-optimal pairing.

Proof Sketch: Let S be the matching produced by the P&R algorithm, and assume for the sake of contradiction that S is not job-optimal.

P&R produces a stable matching, so no job can stop making offers before it offers to its job-optimal candidate.
$\Rightarrow$ In P&R, every job makes an offer to its optimal candidate

Since S is not job-optimal, at least one job-optimal candidate rejects

Let t be the first time a job-optimal candidate rejects an offer
$\Rightarrow$ At time t, job j offers to job-optimal c^* and is rejected

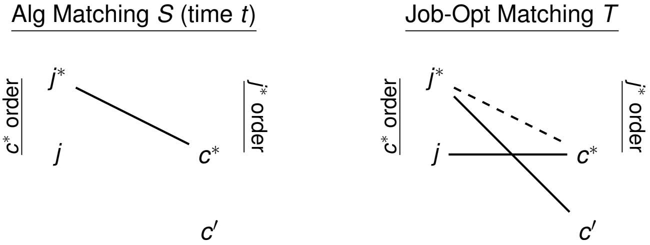
We could really use a picture...

> *I have a truly marvelous picture to demonstrate this proof that this slide is too small to contain. – Fermat, 1637*

Or maybe he would have said that if he knew what a “slide” is...

Job-Optimality of Propose and Reject – Picture It!

Let t be the first time a job-optimal candidate rejects an offer
$\Rightarrow$ At time t, job j offers to job-optimal c^* and is rejected



c^* is j's optimal match

c^* rejects j at time t, so must accept some job j^* that they like better

j^* is matched to some job, say c' in job-optimal matching

Before time t, j^* hasn't rejected its optimal match c' but offered to c^*
$\Rightarrow$ c' is lower on j^*'s list than c^*

(j^*, c^*) is a rogue pair in T, so T is not stable. Contradiction! □

Well-Ordering Principle

Let t be the first time a job-optimal candidate rejects an offer

...

Before time t, j^* hasn't rejected its optimal match

If something "goes bad" there must be a first time the bad thing happens

Related to induction...

- Think of induction for "the algorithm doesn't make a bad move at step k"
- At some step the induction breaks
- Identifying that step and using "OK until then" is vital!

Read more about it in the notes...

Job-Optimal and Candidate-Pessimal

Theorem: If a stable matching is job-optimal, then it is candidate-pessimal.

Proof Sketch: Let S be a job-optimal stable matching that pairs job j with candidate c, and assume for the sake of contradiction that there is a stable matching T that is worse for c.

In T: c matches to a worse candidate j^*

In T: j can't match *better* than c since S is job-optimal – matches to worse c^*

But now (j, c) is a rogue pair in T, so T is not stable.

Contradiction! □

Candidate Optimality

Propose and Reject is job-optimal – what if we want candidate-optimal?

Needed two different sets to match
$\Rightarrow$ *Was anything special about either?* No! Completely symmetric

In reality: Propose and reject is *proposer-optimal*

Solution: Swap roles – candidates propose to jobs
$\Rightarrow$ *Now candidate-optimal!*

Residency Matching

The method was used to match residents to hospitals.

Until 1990s: Hospital optimal

Then: Resident optimal

Now: Placing couples together, other real-world complications

Takeaways

Analysis of cool algorithm with interesting goal: stability.

Stability seems like a good idea – is it possible?

- Two-set instance: Yes
- One-set instance: No

Can we *find* a stable matching?

- Yes! Propose and Reject algorithm
- Basic idea: Over time things get better for candidates, worse for jobs
- Eventually reaches a balance

... and we can (and did) prove it always finds a stable matching

Beyond stability – several stable solutions – which is better?
$\Rightarrow$ For jobs? For candidates?