Stable Matching Problem

- *n* candidates and *n* jobs.
- Each job has a ranked preference list of candidates.
- Each candidate has a ranked preference list of jobs.

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.

Objectives

Produce a matching that one cannot improve upon!

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

Definition: A **rogue couple** j, c^* for a pairing S:
j and c^* prefer each other to their partners in S

A stable matching??

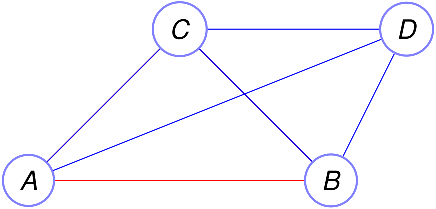
Given a set of preferences.

Is there a stable matching?

How does one find it?

Consider a single type version: stable roommates.

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



The Propose and Reject Algorithm.

Each Day:

1. Each job **proposes** to its favorite candidate on its list.
2. Each candidate rejects all but their favorite proposer (whom they put on a string.)
3. Rejected job crosses rejecting candidate off its list.

Stop when each job gets exactly one proposal (candidate).

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 on its list.
2. Each candidate rejects all but their favorite proposer (whom they put on a string.)
3. Rejected job crosses rejecting candidate off its list.

Stop when each job gets exactly one proposal.

What can we prove about it?
Does this terminate?

...produce a matching?

....a stable matching?

Who does "better": jobs or candidates?

Termination.

Every non-terminated day a job **crossed** an item off the list.

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

Terminates in $\leq n^2$ steps!

It gets better every day for candidates.

Improvement Lemma: It just gets better for candidates
If on day t a 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.

Example: Candidate "1" has job "C" on string on day 5.

1 has job "A" on string on day 7.

Does 1 prefer "C" or "A"?

c - '1', j - 'C', j' - 'A', $t = 5$, $t' = 7$.

Improvement Lemma says 1 prefers 'A'.

Day 10: Can 1 have "A" on a string? Yes.

1 prefers day 10 job as much as day 7 job. Here, $j = j'$.

Why is lemma true?

Proof Idea: Candidate can always keep the previous job on the string.

Improvement Lemma

Improvement Lemma: It just gets better for candidates.

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

Proof:
$P(k)$- - "job on c's string is at least as good as j on day $t+k$"

$P(0)$– true. Candidate has j on string.

Assume $P(k)$. Let j' be job **on string** on day $t+k$.

On day $t+k+1$, job j' still on string.
Candidate c can choose j', or do better with another job, j''

That is, $j' \geq j$ by induction hypothesis.
And j'' is better than j' by algorithm.
$\implies$ Candidate does at least as well as with j.

$P(k) \implies P(k+1)$.
And by principle of induction, lemma holds for every day after t. □

Matching when done.

Lemma: Every job is matched at end.

Proof:
If not, a job j must have been rejected n times.

Every candidate has been proposed to by j,
and Improvement lemma

$\Longrightarrow$ each candidate has a job on a string.

and each job is on at most one string.

n candidates and n jobs. Same number of each.

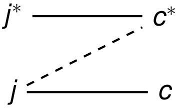
$\Longrightarrow$ j must be on some candidate's string!

Contradiction. □

Matching is Stable.

Lemma: There is no rogue couple for the matching formed by Propose-and-Reject algorithm.

Proof:
Assume there is a rogue couple; (j, c^*)



j prefers c^* to c.

c^* prefers j to j^*.

Job j proposes to c^* before proposing to c.

So c^* rejected j (since he moved on)

By improvement lemma, c^* prefers j^* to j.

Contradiction! □

Question: Proof of Job Propose and Reject a stable pairing uses?

(A) Contradiction.

(B) Uses the improvement lemma.

(C) Induction.

(D) The algorithm description.

(A), (B), (C), (D).

Good for jobs? candidates?

Is the Job-Proposes better for jobs? for candidates?

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

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

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.

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

Subtlety here: Best partner in any stable matching.
As well as you can be in a globally stable solution!

Question: Is there a job or candidate optimal matching?
Is it possible:
j-optimal pairing different from the j'-optimal matching!
Yes? No?

Understanding Optimality: by example.

A:	1,2	1:	A,B
B:	1,2	2:	B,A

Consider pairing: $(A,1),(B,2)$.

Stable? Yes.

Optimal for *B*?

Notice: only one stable pairing. If (A,2) are pair, (A,1) is rogue couple.

So this is the best *B* can do in a stable pairing.

So optimal for *B*.

Also optimal for *A*, 1 and 2. Also pessimal for *A*,*B*,1 and 2.

A:	1,2	1:	B,A
B:	2,1	2:	A,B

Pairing *S*: $(A,1),(B,2)$. Stable? Yes.

Pairing *T*: $(A,2),(B,1)$. Also Stable.

Which is optimal for *A*? *S* Which is optimal for *B*? *S*

Which is optimal for 1? *T* Which is optimal for 2? *T*

Pessimality?

Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

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

Proof:
Assume not: some job is not paired with its optimal candidate.

Let t be first day some job j gets rejected by its optimal candidate c.

There is a stable pairing S where j and c are paired.

j^* - knocks j off of c's string on day t $\implies$ c prefers j^* to j

By choice of t, j^* likes c at least as much as its optimal candidate.

$\implies$ j^* prefers c to its partner c^* in S.

(j^*, c) – Rogue couple for S.
So S is not a stable pairing. Contradiction. $\square$

Notes: S - stable. $(j^*, c^*) \in S$. But (j^*, c) is rogue couple!

Used Well-Ordering principle.

How about for candidates?

Theorem: Job Propose and Reject produces candidate-pessimal pairing.

T – pairing produced by JPR.
S – worse stable pairing for candidate c.
In T, (c, j) is pair.
In S, (c, j^*) is pair.
c prefers j to j^*.
T is job optimal, so j prefers c to its partner in S.
(c, j) is Rogue couple for S
S is not stable.
Contradiction. □

Quick Questions.

How does one make it better for candidates?

Propose and Reject - stable matching algorithm. One side proposes.

Jobs Propose $\implies$ job optimal.

Candidates propose. $\implies$ optimal for candidates.

Residency Matching..

The method was used to match residents to hospitals.

Hospital optimal....

..until 1990's...Resident optimal.

Another variation: couples.

Takeaways.

Analysis of cool algorithm with interesting goal: stability.

"Economic": different utilities.

Definition of optimality: best utility in stable world.

Action gives better results for individuals but gives instability.

Induction over steps of algorithm.

Proofs carefully use definition:
Stability:
Improvement Lemma plus every day the job gets to choose.
Optimality proof:
Job Optimality:
contradiction of the existence of a better *stable* pairing.
that is, no rogue couple by improvement, job choice, and well ordering principle. Candidate Pessimality:
contradiction plus job optimality implies better pairing.