Stable Matching Intro

Note 4

Goal: We have n candidates and n jobs; each candidate and job has a preference list. We want to pair up candidates and jobs such that there is a stable matching.

Definitions:

- *Stable Matching Instance*: A set of jobs and candidates and their respective preference lists
- *Matching*: Disjoint set of (c_i, j_i) pairs that are matched together
- *Rogue Couple*: a pair (c, j) *not* in the matching that prefers each other over their current matchings (c, j') and (c', j)
- *Stable Matching*: matching with no rogue couples
- *Optimal candidate for job*: highest ranked candidate for a job in any *stable* matching
- *Optimal job for candidate*: highest ranked job for a candidate in any *stable* matching
- *Job optimal*: All jobs get their optimal candidates; a candidate optimal matching is defined similarly
- *Candidate pessimal*: All candidates get their pessimal jobs (i.e. lowest ranked job for each candidate in any *stable* matching); a job pessimal matching is defined similarly

Propose and reject algorithm: Each day,

1. *Morning*: Jobs propose to the top candidate who have not rejected them; note that a job will propose to the same candidate as the previous day if they were not rejected
2. *Afternoon*: Candidates say "maybe" to the best job offer so far, keeping them *in hand* or *on a string*, and say "no" to every other offer
3. *Evening*: Jobs cross off the candidates that have rejected them

The process halts when no rejections happen; all candidates then accept their current offer. Note that the algorithm only produces one stable matching, so there can be other stable matchings not produced by the algorithm.

Improvement lemma: every candidate will only say maybe to better job offers as time goes on. Similarly, every job will only propose to worse candidates as time goes on.

As a result, the propose and reject algorithm always produces a stable matching that is job optimal and candidate pessimal.

Remember, a stable matching *instance* is only defined as the set of jobs, candidates, and preference lists. It does not include the matching itself, nor any algorithm.

1 Stable Matching

Note 4

Consider the set of jobs $J = \{1, 2, 3\}$ and the set of candidates $C = \{A, B, C\}$ with the following preferences.

Jobs	Candidates
1	A > B > C
2	B > A > C
3	A > B > C

Candidates	Jobs
A	2 > 1 > 3
B	1 > 3 > 2
C	1 > 2 > 3

Run the traditional propose-and-reject algorithm on this example. How many days does it take and what is the resulting pairing? (Show your work.)

Solution:

The algorithm takes 5 days to produce a matching. The resulting pairing is as follows. The circles indicate the job that a candidate picked on a given day (and rejected the rest).

$$\{(A,2),(B,1),(C,3)\}.$$

Candidate	Day 1	Day 2	Day 3	Day 4	Day 5
A	(1),3	(1)	1,(2)	(2)	(2)
B	(2)	2,(3)	(3)	(1),3	(1)
C					(3)

2 Propose-and-Reject Proofs

Note 4

Prove the following statements about the traditional propose-and-reject algorithm.

(a) In any execution of the algorithm, if a candidate receives a proposal on day i, then they receive some proposal on every day thereafter until termination.

(b) In any execution of the algorithm, if a candidate receives no proposal on day i, then they receive no proposal on any previous day j, $1 \leq j < i$.

(c) In any execution of the algorithm, there is at least one candidate who only receives a single proposal. (Hint: use the parts above!)

(d) There does not exist a stable matching instance for n jobs and n candidates for $n > 1$, such that in a stable matching algorithm with jobs proposing, every job ends up with its least preferred candidate.

Solution:

(a) The idea is to induct on the number of days passed so far.
Base case: Candidate C receives a proposal on day i
Inductive Step: Assume C receives a proposal on day $j \geq i$ from job J. We want to show they

will also get a proposal on day $j+1$. There are two cases: C prefers J to all other offers, or C prefers some job J' to J. In the first case J proposes to C on day $j+1$ and in the second J' proposes to C on day $j+1$ so C receives at least one proposal on day $j+1$.

(b) One way is to use a proof by contradiction. Assume that a candidate receives no proposal on day i but did receive a proposal on some previous day j, $1 \leq j < i$. By the previous part, since the candidate received a proposal on day j, they must receive at least one proposal on every day after j. But $i > j$, so the candidate must have received a proposal on day i, contradicting our original assumption that they did not.

(c) Let's say the algorithm takes k days. This means that every candidate must have received a proposal on day k. However, this also means that there is at least one candidate C who does not receive a proposal on day $k-1$–if this were not the case, the algorithm would have already terminated on day $k-1$. Then from part (b), since C did not receive a proposal on day $k-1$, they didn't receive a proposal on any day before k. Furthermore, we know they got exactly one proposal on day k, since the algorithm terminated on that day. Thus, we have that C receives exactly one proposal throughout the entire run of the algorithm.

(d) If such an instance existed, it would mean that at the end of the algorithm, every job would have proposed to every candidate on its list and has been rejected $n-1$ times. This also means that every candidate got proposed to by every single one of the jobs, and at the very end must have rejected $n-1$ jobs in total (to end up with only one preferred job at the last day). We know this is impossible though, as we learned above that at least one candidate receives a single proposal. Thus, there must be at least one candidate who is not proposed to until the very last day.

3 Be a Judge

Note 4

For each of the following statements, indicate whether the statement is True or False and justify your answer with a short 2-3 line explanation:

(a) In a stable matching instance, if job J and candidate C each put each other at the top of their respective preference lists, then J must be paired with C in every stable pairing.

(b) In a stable matching instance with at least two jobs and two candidates, if job J and candidate C each put each other at the bottom of their respective preference lists, then J cannot be paired with C in any stable pairing.

(c) For every $n > 1$, there is a stable matching instance for n jobs and n candidates which has an **unstable** pairing where **every** unmatched job-candidate pair is a rogue couple.

Solution:

(a) **True**: We give a simple proof by contradiction. Assume that J and C put each other at the top of their respective preference lists, but J and C are not paired with each other in some stable pairing S. Thus, S includes the pairings (J, C'), (J', C), for some job J' and candidate C'. However, J prefers C over its partner in S, since C is at the top of J's preference list.

Similarly C prefers J over her current job. Thus (J,C) form a rogue couple in S, so S is not stable. We have arrived at a contradiction that S exists where C and J are not paired.

Therefore if job J and candidate C put each other at the top of their respective preference lists, then J must be paired with C in every stable pairing.

(b) **False**: The key here is to realize that this is possible if job J and candidate C are at the bottom of everybody else's preference list as well. For example, a two job, two candidate instance where the first job is best for both candidates, and the first candidate is best for both jobs has its only stable pairing where the first job and first candidate are paired (by part (b)), and the second job and second candidate are paired. The second job and second candidate are each other's least preferred option.

(c) **True**: The key idea to this solution is that we want a set of preferences for which J_i and C_i like each other the least and make a pairing J_i and C_i together. In this matching M each unmatched couple is a rogue couple. In particular, an instance can be constructed by forming preferences where job i's (candidate i's) least favorite candidate is candidate i (job i) and an *arbitrary* ordering on all the others. Thus, for any job i and candidate j, where $i \neq j$, then job i prefers candidate j to i (its partner in M) and candidate j prefers job i to j (its partner in M.) That, is every pair i and j is a rogue couple.

An example for two jobs and two candidates, is the instance:

Jobs		
1	B	A
2	A	B

Candidates		
A	2	1
B	1	2

The pairing $P = \{(A,1),(B,2)\}$ is the pairing where each pair of jobs and candidates that are not in P, $(A,2)$ and $(B,1)$, are rogue couples.

4 Stable Matching III

Note 4

(a) True or False?

(i) If a candidate accidentally rejects a job they prefer on a given day, then the algorithm still always ends with a matching.

(ii) The Propose-and-Reject Algorithm never produces a candidate-optimal matching.

(iii) If the same job is last on the preference list of every candidate, the job must end up with its least preferred candidate.

(b) As you've seen from lecture, the jobs-proposing Propose-and-Reject Algorithm produces an employer-optimal stable matching. Let's see if the candidate have any way of improving their standing. Suppose exactly one of the candidates has the power to arbitrarily reject one proposal, regardless of which job they have on their string (if any). Construct an example that illustrates the following: for any $n \geq 2$, there exists a stable matching instance for which

using this power helps **every** candidate, i.e. every candidate gets a better job than they would have gotten under the jobs-proposing Propose-and-Reject Algorithm.

Solution:

(a) (i) False, consider the case:

Jobs		
1	A	B
2	B	A

Candidates		
A	1	2
B	2	1

Using SMA, the matching will be: $(A,1),(B,2)$. If candidate A rejects job 1 despite having no other jobs on their string, job 1 will propose to candidate B and also get rejected. This leaves both candidate A and job 1 partnerless. In this case, the accidental rejection prevents a matching from being produced at all.

(ii) False. Suppose that all jobs have a different first choice. Also supposed that the job proposing is each candidate's first choice. In this case, the algorithm would end after the first day with both jobs and candidates ending with their top pick. In this case, the result is candidate-optimal.

(iii) False, consider the following case where jobs are numbers and candidates are letters:

Jobs		
1	A	B
2	B	A

Candidates		
A	2	1
B	2	1

Job 1 is last on every candidate's list, however, $\{(A,1),(B,2)\}$ is a stable pairing where job 1 got its top choice candidate.

(b) Without loss of generality, assume that candidate 1 is the candidate with this special power. Now, assume the preference lists are ordered as follows:

Job	Preferences	Candidate	Preferences
1	$1>2>\cdots>n-1>n$	1	$n>n-1>\cdots>2>1$
2	$2>3>\cdots>n-1>n>1$	2	$1>n>n-1>\cdots>2$
3	$3>4>\cdots>n>1>2$	3	$2>1>n>n-1>\cdots>3$
	$\cdots$		$\cdots$
n	$n>1>2>\cdots>n-1$	n	$n-1>n-2>\cdots>1>n$

If the Propose-and-Reject Algorithm was run with these preference lists, then each candidate would be stuck with their least-preferred job. However, let's say candidate 1 rejects job 1 on the first day, even though they have nobody on their string. Then, job 1 will be forced to propose to its second option, candidate 2, and they will accept because the job is their first choice. Now, job 2 has no partner and will propose to candidate 3, who will accept, leaving job 3 without a partner. This process continues until job n proposes to candidate 1. One rejection has led all candidates to get their best choice instead of their worst choice!