CSE 421 Section 1

Stable Matching

Administrivia & Introductions

Homework

- Submissions
  - LaTeX (highly encouraged)
    - overleaf.com
    - template and LaTeX guide posted on course website!
  - Word Editor that supports mathematical equations
  - Handwritten neatly and scanned
- All homeworks will be turned in via Gradescope
- Homeworks typically due on Wednesdays at 10pm

Announcements & Reminders

- Section Materials
  - Handouts will be provided in at each section
  - Worksheets and sample solutions will be available on the course calendar later this evening

- HW1
  - Due Wednesday 1/11 @ 10pm

Stable Matching

Stable Matching

Given n riders and n horses with preference lists, how can we find a stable matching so that all riders have horses and all horses have riders?

Perfect Matching:

- Each rider is paired with exactly one horse
- Each horse is paired with exactly one rider

Stability: No ability to exchange partners

Blocking: An unmatched pair r-h is blocking if they both prefer each other to current matches

Stable Matching: perfect matching with no blocking pairs

Gale-Shapley Algorithm

Algorithm to find a stable matching:

```
Initially all r in R and h in H are free
while there is a free r
        Let h be highest on r's list that r has not proposed to
        if h is free
                match (r,h)
        else //h is not free
                Let r' be the current match of h
                if h prefers r to r'
                        unmatch (r',h)
                        match (r,h)
```

Problem 1 – Gale-Shapley

Consider the following stable matching instance:

r_1: h_3, h_1, h_2, h_4
r_2: h_2, h_1, h_4, h_3
r_3: h_2, h_3, h_1, h_4
r_4: h_3, h_4, h_1, h_2

h_1: r_4, r_1, r_3, r_2
h_2: r_1, r_3, r_2, r_4
h_3: r_1, r_3, r_4, r_2
h_4: r_3, r_1, r_2, r_4

a) Run the Gale-Shapley Algorithm with riders proposing on the instance above. When choosing which free rider to propose next, always choose the one with the smallest index (e.g., if r_1 and r_2 are both free, always choose r_1).

Problem 1 – Gale-Shapley

a) Run the Gale-Shapley Algorithm with riders proposing on the instance above. When choosing which free rider to propose next, always choose the one with the smallest index (e.g., if r_1 and r_2 are both free, always choose r_1).

r_1: h_3, h_1, h_2, h_4
r_2: h_2, h_1, h_4, h_3
r_3: h_2, h_3, h_1, h_4
r_4: h_3, h_4, h_1, h_2

h_1: r_4, r_1, r_3, r_2
h_2: r_1, r_3, r_2, r_4
h_3: r_1, r_3, r_4, r_2
h_4: r_3, r_1, r_2, r_4

r_1 chooses h_3 (r_1, h_3)

Problem 1 – Gale-Shapley

a) Run the Gale-Shapley Algorithm with riders proposing on the instance above. When choosing which free rider to propose next, always choose the one with the smallest index (e.g., if r_1 and r_2 are both free, always choose r_1).

r_1: h_3, h_1, h_2, h_4
r_2: h_2, h_1, h_4, h_3
r_3: h_2, h_3, h_1, h_4
r_4: h_3, h_4, h_1, h_2

h_1: r_4, r_1, r_3, r_2
h_2: r_1, r_3, r_2, r_4
h_3: r_1, r_3, r_4, r_2
h_4: r_3, r_1, r_2, r_4

r_1 chooses h_3 — (r_1, h_3)

r_2 chooses h_2 — $(r_1, h_3), (r_2, h_2)$

Problem 1 – Gale-Shapley

a) Run the Gale-Shapley Algorithm with riders proposing on the instance above. When choosing which free rider to propose next, always choose the one with the smallest index (e.g., if r_1 and r_2 are both free, always choose r_1).

r_1: h_3, h_1, h_2, h_4
r_2: h_2, h_1, h_4, h_3
r_3: h_2, h_3, h_1, h_4
r_4: h_3, h_4, h_1, h_2

h_1: r_4, r_1, r_3, r_2
h_2: r_1, r_3, r_2, r_4
h_3: r_1, r_3, r_4, r_2
h_4: r_3, r_1, r_2, r_4

r_1 chooses h_3	(r_1, h_3)
r_2 chooses h_2	$(r_1, h_3), (r_2, h_2)$
r_3 chooses h_2	(r_1, h_3), **(r_2, h_2)**, **(r_3, h_2)**

Problem 1 – Gale-Shapley

a) Run the Gale-Shapley Algorithm with riders proposing on the instance above. When choosing which free rider to propose next, always choose the one with the smallest index (e.g., if r_1 and r_2 are both free, always choose r_1).

r_1: h_3, h_1, h_2, h_4
r_2: h_2, h_1, h_4, h_3
r_3: h_2, h_3, h_1, h_4
r_4: h_3, h_4, h_1, h_2

h_1: r_4, r_1, r_3, r_2
h_2: r_1, r_3, r_2, r_4
h_3: r_1, r_3, r_4, r_2
h_4: r_3, r_1, r_2, r_4

r_1 chooses h_3	(r_1, h_3)
r_2 chooses h_2	$(r_1, h_3), (r_2, h_2)$
r_3 chooses h_2	$(r_1, h_3), \cancel{(r_2, h_2)}, (r_3, h_2)$

Problem 1 – Gale-Shapley

a) Run the Gale-Shapley Algorithm with riders proposing on the instance above. When choosing which free rider to propose next, always choose the one with the smallest index (e.g., if r_1 and r_2 are both free, always choose r_1).

r_1: h_3, h_1, h_2, h_4
r_2: h_2, h_1, h_4, h_3
r_3: h_2, h_3, h_1, h_4
r_4: h_3, h_4, h_1, h_2

h_1: r_4, r_1, r_3, r_2
h_2: r_1, r_3, r_2, r_4
h_3: r_1, r_3, r_4, r_2
h_4: r_3, r_1, r_2, r_4

r_1 chooses h_3	(r_1, h_3)
r_2 chooses h_2	$(r_1, h_3), (r_2, h_2)$
r_3 chooses h_2	$(r_1, h_3), \cancel{(r_2, h_2)}, (r_3, h_2)$
r_2 chooses h_1	$(r_1, h_3), (r_2, h_1), (r_3, h_2)$

Problem 1 – Gale-Shapley

a) Run the Gale-Shapley Algorithm with riders proposing on the instance above. When choosing which free rider to propose next, always choose the one with the smallest index (e.g., if r_1 and r_2 are both free, always choose r_1).

r_1: h_3, h_1, h_2, h_4
r_2: h_2, h_1, h_4, h_3
r_3: h_2, h_3, h_1, h_4
r_4: h_3, h_4, h_1, h_2

h_1: r_4, r_1, r_3, r_2
h_2: r_1, r_3, r_2, r_4
h_3: r_1, r_3, r_4, r_2
h_4: r_3, r_1, r_2, r_4

r_1 chooses h_3	(r_1, h_3)
r_2 chooses h_2	$(r_1, h_3), (r_2, h_2)$
r_3 chooses h_2	$(r_1, h_3), \cancel{(r_2, h_2)}, (r_3, h_2)$
r_2 chooses h_1	$(r_1, h_3), (r_2, h_1), (r_3, h_2)$
r_4 chooses h_3	$\mathbf{(r_1, h_3)}, (r_2, h_1), (r_3, h_2), \mathbf{(r_4, h_3)}$

Problem 1 – Gale-Shapley

a) Run the Gale-Shapley Algorithm with riders proposing on the instance above. When choosing which free rider to propose next, always choose the one with the smallest index (e.g., if r_1 and r_2 are both free, always choose r_1).

r_1: h_3, h_1, h_2, h_4
r_2: h_2, h_1, h_4, h_3
r_3: h_2, h_3, h_1, h_4
r_4: h_3, h_4, h_1, h_2

h_1: r_4, r_1, r_3, r_2
h_2: r_1, r_3, r_2, r_4
h_3: r_1, r_3, r_4, r_2
h_4: r_3, r_1, r_2, r_4

r_1 chooses h_3	(r_1, h_3)
r_2 chooses h_2	$(r_1, h_3), (r_2, h_2)$
r_3 chooses h_2	$(r_1, h_3), \cancel{(r_2, h_2)}, (r_3, h_2)$
r_2 chooses h_1	$(r_1, h_3), (r_2, h_1), (r_3, h_2)$
r_4 chooses h_3	$(r_1, h_3), (r_2, h_1), (r_3, h_2), \cancel{(r_4, h_3)}$

Problem 1 – Gale-Shapley

a) Run the Gale-Shapley Algorithm with riders proposing on the instance above. When choosing which free rider to propose next, always choose the one with the smallest index (e.g., if r_1 and r_2 are both free, always choose r_1).

r_1: h_3, h_1, h_2, h_4
r_2: h_2, h_1, h_4, h_3
r_3: h_2, h_3, h_1, h_4
r_4: h_3, h_4, h_1, h_2

h_1: r_4, r_1, r_3, r_2
h_2: r_1, r_3, r_2, r_4
h_3: r_1, r_3, r_4, r_2
h_4: r_3, r_1, r_2, r_4

r_1 chooses h_3	(r_1, h_3)
r_2 chooses h_2	$(r_1, h_3), (r_2, h_2)$
r_3 chooses h_2	$(r_1, h_3), \cancel{(r_2, h_2)}, (r_3, h_2)$
r_2 chooses h_1	$(r_1, h_3), (r_2, h_1), (r_3, h_2)$
r_4 chooses h_3	$(r_1, h_3), (r_2, h_1), (r_3, h_2), \cancel{(r_4, h_3)}$
r_4 chooses h_4	$(r_1, h_3), (r_2, h_1), (r_3, h_2), (r_4, h_4)$

Problem 1 – Gale-Shapley

a) Run the Gale-Shapley Algorithm with riders proposing on the instance above. When choosing which free rider to propose next, always choose the one with the smallest index (e.g., if r_1 and r_2 are both free, always choose r_1).

r_1: h_3, h_1, h_2, h_4
r_2: h_2, h_1, h_4, h_3
r_3: h_2, h_3, h_1, h_4
r_4: h_3, h_4, h_1, h_2

h_1: r_4, r_1, r_3, r_2
h_2: r_1, r_3, r_2, r_4
h_3: r_1, r_3, r_4, r_2
h_4: r_3, r_1, r_2, r_4

r_1 chooses h_3	(r_1, h_3)
r_2 chooses h_2	$(r_1, h_3), (r_2, h_2)$
r_3 chooses h_2	$(r_1, h_3), \cancel{(r_2, h_2)}, (r_3, h_2)$
r_2 chooses h_1	$(r_1, h_3), (r_2, h_1), (r_3, h_2)$
r_4 chooses h_3	$(r_1, h_3), (r_2, h_1), (r_3, h_2), \cancel{(r_4, h_3)}$
r_4 chooses h_4	$(r_1, h_3), (r_2, h_1), (r_3, h_2), (r_4, h_4)$

$(r_1, h_3), (r_2, h_1), (r_3, h_2), (r_4, h_4)$

Problem 1 – Gale-Shapley

b) Run the Gale-Shapley Algorithm with riders proposing on the same instance. But now, when choosing which free rider to propose next, always choose the one with the largest index. Do you get the same result?

r_1: h_3, h_1, h_2, h_4
r_2: h_2, h_1, h_4, h_3
r_3: h_2, h_3, h_1, h_4
r_4: h_3, h_4, h_1, h_2

h_1: r_4, r_1, r_3, r_2
h_2: r_1, r_3, r_2, r_4
h_3: r_1, r_3, r_4, r_2
h_4: r_3, r_1, r_2, r_4

c) Now run the algorithm with horses proposing, breaking ties by taking the free horse with the smallest index. Do you get the same result?

Work on parts b and c of this problem with the people around you, and then we'll go over it together!

Problem 1 – Gale-Shapley

b) Run the Gale-Shapley Algorithm with riders proposing on the same instance. But now, when choosing which free rider to propose next, always choose the one with the largest index. Do you get the same result?

r_1: h_3, h_1, h_2, h_4
r_2: h_2, h_1, h_4, h_3
r_3: h_2, h_3, h_1, h_4
r_4: h_3, h_4, h_1, h_2

h_1: r_4, r_1, r_3, r_2
h_2: r_1, r_3, r_2, r_4
h_3: r_1, r_3, r_4, r_2
h_4: r_3, r_1, r_2, r_4

Problem 1 – Gale-Shapley

c) Now run the algorithm with horses proposing, breaking ties by taking the free horse with the smallest index. Do you get the same result?

r_1: h_3, h_1, h_2, h_4
r_2: h_2, h_1, h_4, h_3
r_3: h_2, h_3, h_1, h_4
r_4: h_3, h_4, h_1, h_2

h_1: r_4, r_1, r_3, r_2
h_2: r_1, r_3, r_2, r_4
h_3: r_1, r_3, r_4, r_2
h_4: r_3, r_1, r_2, r_4

Proof or Counterexample?

Prove or Disprove?

Often, you will be given a statement, and then asked to either prove or disprove it. This can be stressful! How do you know which you should start with?

The best way to begin, especially when you don’t know if the claim is even true, is to try to understand it better by producing some examples. This has two main benefits that will help, whether you end up proving or disproving the claim:

1) You get a better understanding of the statement so now you have a clear method of approach, OR
2) You find a counterexample, which allows you to easily write a quick proof that the statement is false!

Problem 2 – A Quick Proof

Is it possible to have a stable matching instance with more than 2 stable matchings? If so, give an instance and at least 3 stable matchings. If not, prove that every instance has at most 2 stable matchings.

Work on this problem with the people around you, and then we'll go over it together!

Problem 2 – A Quick Proof

Is it possible to have a stable matching instance with more than 2 stable matchings? If so, give an instance and at least 3 stable matchings. If not, prove that every instance has at most 2 stable matchings.

Induction

Induction

- You will be writing lots of induction proofs in this class in order to prove that your algorithms work the way you say they will.

- The style requirements for proofs in this class are less stringent than the style requirements from 311
    - there is a **style guide** doc on the course website (here) about how 421 proofs are different than what you did in 311

Induction Template

Let $P(n)$ be "(whatever you're trying to prove)".
We show $P(n)$ holds for all n by induction on n.

Base Case: Show $P(b)$ is true.

Inductive Hypothesis: Suppose $P(k)$ holds for an arbitrary $k \geq b$

Inductive Step: Show $P(k+1)$ (i.e. get $P(k) \rightarrow P(k+1)$)

Conclusion: Therefore, $P(n)$ holds for all n by the principle of induction.

Problem 3 – Induction Review

Consider the following claim:

Let $P(n)$ be "Every tree with at least n nodes has at least two nodes of degree-one."

a) What is the correct "skeleton" of the inductive step (i.e., the right things to assume and the right target)?

b) Prove the claim by induction.

Problem 3 – Induction Review

Consider the following claim:

Let $P(n)$ be "Every tree with at least n nodes has at least two nodes of degree-one."

a) What is the correct "skeleton" of the inductive step (i.e., the right things to assume and the right target)?

Work on this problem with the people around you, and then we'll go over it together!

Problem 3 – Induction Review

Consider the following claim:

Let $P(n)$ be “Every tree with at least n nodes has at least two nodes of degree-one.”

a) What is the correct “skeleton” of the inductive step (i.e., the right things to assume and the right target)?

KEY Induction Concept

It might be really tempting to structure the inductive step of this problem as something like, “start with an arbitrary tree T of size k nodes, and then add a node to it, making tree T' with $k + 1$ nodes.”

This is a ***BAD*** idea! Then we’d have to cover every possible way to add on a node (and prove that we had actually dealt with every possible case), making the overall proof way more complicated and unwieldly.

Instead, we ***ALWAYS*** want to start with the bigger thing (in this case, with the arbitrary tree T' of size $k + 1$) and find the smaller thing inside of it.

Problem 3 – Induction Review

Consider the following claim:

Let $P(n)$ be "Every tree with at least n nodes has at least two nodes of degree-one."

b) Prove the claim by induction.

Work on this problem with the people around you, and then we'll go over it together!

Problem 3 – Induction Review

b) Prove the claim by induction.

That's All, Folks!

Thanks for coming to section this week!
Any questions?