CSE 421
Introduction to Algorithms

Lecture 1: Intro & Stable Matching

https://cs.washington.edu/421
Instructor

Paul Beame [he/him]
beame@cs
Specialty: Complexity and Applications
https://homes.cs.washington.edu/~beame
Office: CSE 668
A Dedicated Team of TAs

Daniel Gao  Raymond Guo  Samarjit Kaushik  Kyle Mumma  Edward Qin  Robert Stevens

Glenn Sun  Aman Thukral  Tom Tian  Maxwell Wang  Ben Zhang  Muru Zhang

See https://cs.washington.edu/421/staff.html to learn more about their backgrounds and interests!
Getting Started (Your TODO List)

• Make sure you are on Ed (a.k.a. EdStem)!
  • Check your inbox – and maybe your SPAM filter – for an invitation

• Attend your first Quiz Section tomorrow!

• Homework 1 will be out tonight
  • You will have enough to start on it after section tomorrow
  • Start thinking about it right away after that

• Get all the credit you deserve: Sign up for CSE 493Z

• Attend lecture and participate
  • Students who participate do better on average
Coursework

• 8 homework assignments roughly (due Wednesdays)
  • Typically 1 mechanical problem
    3 long-form problems
  • See the Homework Guide linked on the course website
  • Start early to reduce amount of time you need to concentrate on them
    • Use your brain’s background processing
  • OK to talk with fellow students but solution write-up must be your own
    • See syllabus https://cs.washington.edu/421/syllabus.html
  • Use of outside resources for solutions forbidden (see syllabus)
    • Generative AI does worse than almost anyone in the class would on their own...
Late Problem Days

• Late days per problem rather than for the whole assignment
  • Each problem is a separate Gradescope submission
  • Max 2 late days per problem; limit on total # of late problem days
  • You should submit anything that is done as soon as you are finished with it
  • See the syllabus for details
Exam dates

Midterm: Wednesday Nov 8 (possibly evening to give you more time for the same problems)

Final Exam: Standard exam time and place: Monday Dec 11, 2:30-4:20 here

Grading scheme

• Homework  55%
• Midterm    15-20%
• Final Exam 25-30%
Textbook

Kleinberg-Tardos: Algorithm Design

- International Edition just as good
- Plus supplements on website

- Worth reading
  - Good for reading sequentially and learning how to think like an algorithm designer
  - Not as good for random access

- Not required
  - All required content will be on slides in lectures and quiz section
Introduction to Algorithms

• Basic techniques for the design and analysis of algorithms.
  • Develop a toolkit of ways to find efficient algorithms to solve problems
  • Prove that the algorithms are correct
  • Analyze their efficiency properties
  • Communicate these algorithms and their properties to others
On efficiency

• Originally, efficiency was important for many reasons but partly because computers were weak

• Now we have powerful computers but
  • Data has grown to be enormous
    • We need *even more* efficient algorithms at this scale
  • Computation has an *energy cost* and represents a significant part of society’s total energy use
    • Efficient computing is essential to reducing that cost
  • Additional power is of little help for inefficient (e.g. brute force) solutions
Introduction to Algorithms

- Stable Matching
Matching Medical Residents to Hospitals

**Goal:** Given a set of preferences among hospitals and medical school residents (graduating medical students), design a *self-reinforcing* admissions process.

**Unstable pair:** applicant $x$ and hospital $y$ are *unstable* if:
- $x$ prefers $y$ to their assigned hospital.
- $y$ prefers $x$ to one of its admitted residents.

**Stable assignment.** Assignment with no unstable pairs.
- Natural and desirable condition.
- Individual self-interest will prevent any applicant/hospital side deal from being made.
Simpler: Stable Matching Problem

**Goal:** Given two groups of \( n \) people each, find a "suitable" matching.

- Participants rate members from opposite group.
- Each person lists members from the other group in order of preference from best to worst.

<table>
<thead>
<tr>
<th>Group P Preference Profile</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
</tr>
<tr>
<td>Y</td>
</tr>
<tr>
<td>Z</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group R Preference Profile</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>C</td>
</tr>
</tbody>
</table>
Stable Matching Problem

**Perfect matching:** everyone is matched to precisely one person from the other group.

**Stability:** self-reinforcing, i.e. no incentive to undermine assignment by joint action.
- For a matching $M$, an unmatched pair $p-r$ from different groups is *unstable* if $p$ and $r$ prefer each other to current partners.
- Unstable pair $p-r$ could each improve by ignoring the assignment.

**Stable matching:** perfect matching with no unstable pairs.

**Stable matching problem:** Given the preference lists of $n$ people from each of two groups, find a stable matching between the two groups if one exists.
Stable Matching Problem

Q: Is matching \((X,C), (Y,B), (Z,A)\) stable?

Group P Preference Profile

<table>
<thead>
<tr>
<th></th>
<th>1st favorite</th>
<th>2nd least favorite</th>
<th>3rd least favorite</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>Y</td>
<td>B</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>Z</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
</tbody>
</table>

Group R Preference Profile

<table>
<thead>
<tr>
<th></th>
<th>1st favorite</th>
<th>2nd favorite</th>
<th>3rd favorite</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Y</td>
<td>X</td>
<td>Z</td>
</tr>
<tr>
<td>B</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
</tr>
<tr>
<td>C</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
</tr>
</tbody>
</table>

X would prefer A, A would prefer X, not the only case.
Stable Matching Problem

Q: Is matching \((X, C), (Y, B), (Z, A)\) stable?

A: No. B and X prefer each other.
Stable Matching Problem

**Q:** Is matching \((X,A), (Y,B), (Z,C)\) stable?

**Group P Preference Profile**

<table>
<thead>
<tr>
<th></th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>Y</td>
<td>B</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>Z</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
</tbody>
</table>

**Group R Preference Profile**

<table>
<thead>
<tr>
<th></th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Y</td>
<td>X</td>
<td>Z</td>
</tr>
<tr>
<td>B</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
</tr>
<tr>
<td>C</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
</tr>
</tbody>
</table>
Stable Matching Problem

Q: Is matching \((X,A), (Y,B), (Z,C)\) stable?
A: Yes
Variant: Stable Roommate Problem

Q. Do stable matchings always exist?
A. Not obvious a priori.

Stable roommate problem:
- $2n$ people; each person ranks others from 1 to $2n - 1$.
- Assign roommate pairs so that no unstable pairs.

Observation: Stable matchings do not always exist for stable roommate problem.

<table>
<thead>
<tr>
<th></th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>B</td>
<td>C</td>
<td>A</td>
<td>D</td>
</tr>
<tr>
<td>C</td>
<td>A</td>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>D</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
</tbody>
</table>

$(A,B), (C,D) \Rightarrow B-C$ unstable
$(A,C), (B,D) \Rightarrow A-B$ unstable
$(A,D), (B,C) \Rightarrow A-C$ unstable
Propose-And-Reject Algorithm

Propose-and-reject algorithm: [Gale-Shapley 1962]

Intuitive method that guarantees to find a stable matching.

• Members of one group \( P \) make proposals, the other group \( R \) receives proposals

```
Initialize each person to be free.
while (some \( p \) in \( P \) is free) {
    Choose some free \( p \) in \( P \)
    \( r = 1^{st} \) person on \( p \)'s preference list to whom \( p \) has not yet proposed
    if (\( r \) is free)
        tentatively match (\( p, r \)) // \( p \) and \( r \) both engaged, no longer free
    else if (\( r \) prefers \( p \) to current tentative match \( p' \))
        replace (\( p', r \)) by (\( p, r \)) // \( p \) now engaged, \( p' \) now free
    else
        \( r \) rejects \( p \)
} 
```
Proof of Correctness: Termination (not obvious from the code)

Observation 1: Members of \( P \) propose in decreasing order of preference.

Claim: The Gale-Shapley Algorithm terminates after at most \( n^2 \) iterations.

Proof: Proposals are never repeated (by Observation 1) and there are only \( n^2 \) possible proposals.

It could be nearly that bad...

General form of this example will take \( n(n - 1) + 1 \) proposals.
Proof of Correctness: Perfection

Observation 2: Once a member of $R$ is matched, they never become free; they only "trade up."

Claim: Everyone gets matched.

Proof:

- After some $p$ proposes to the last person on their list, all the $r$ in $R$ have been proposed to by someone (by $p$ at least).
- By Observation 2, every $r$ in $R$ is matched at that point.
- Since $|P| = |R|$ every $p$ in $P$ is also matched.
Proof of Correctness: Stability

Claim: No unstable pairs in the final Gale-Shapley matching $M$

Proof: Consider a pair $p-r$ not matched by $M$

Case 1: $p$ never proposed to $r$.
\[\Rightarrow p \text{ prefers } M\text{-partner to } r.\]
\[\Rightarrow p-r \text{ is not unstable for } M.\]

Case 2: $p$ proposed to $r$.
\[\Rightarrow r \text{ rejected } p \text{ (right away or later when trading up)}\]
\[\Rightarrow r \text{ prefers } M\text{-partner to } p.\]
\[\Rightarrow p-r \text{ is not unstable for } M.\]
Summary

Stable matching problem: Given $n$ people in each of two groups, and their preferences, find a stable matching if one exists.

Gale-Shapley algorithm: Guarantees to find a stable matching for any problem instance.

Q: How do we implement GS algorithm efficiently?

Q: If there are multiple stable matchings, which one does GS find?
Implementation for Stable Matching Algorithms

- Input size
  - \( N = 2n^2 \) words
  - \( 2n \) people each with a preference list of length \( n \)
  - \( 2n^2 \log n \) bits
    - specifying an ordering for each preference list takes \( n \log n \) bits

- Brute force algorithm
  - Try all \( n! \) possible matchings
  - Do any of them work?

- Gale-Shapley Algorithm
  - \( n^2 \) iterations, each costing constant time as follows ...
Propose-And-Reject Algorithm

Propose-and-reject algorithm: [Gale-Shapley 1962]

Intuitive method that guarantees to find a stable matching.

- Members of one group $P$ make proposals, the other group $R$ receives proposals.

```plaintext
Initialize each person to be free.
while (some p in P is free) {
    Choose some free p in P
    r = 1st person on p's preference list to whom p has not yet proposed
    if (r is free)
        tentatively match (p,r)  // p and r both engaged, no longer free
    else if (r prefers p to current tentative match p')
        replace (p',r) by (p,r)  // p now engaged, p' now free
    else
        r rejects p
}
```
Efficient Implementation

How do we get the an $O(n^2)$ time implementation?

**Input:** Representing members of the two groups $P$ and $R$ and their preferences:
- Assume elements of $P$ (proposers) are numbered $1, \ldots, n$.
- Assume elements of $R$ (receivers) are numbered $1', \ldots, n'$.
- For each proposer, a list/array of the $n$ receivers, ordered by preference.
- For each receiver, a list/array of the $n$ proposers, ordered by preference.

**The matching:**
- Maintain two arrays $\text{match}[p]$, and $\text{match}'[r]$.
  - set entry to 0 if free
  - if $p$ matched to $r$ then $\text{match}[p]=r$ and $\text{match}'[r]=p$

**Making proposals:**
- Maintain a list of free proposers, e.g., in a queue.
- Maintain an array $\text{count}[p]$ that counts the number of proposals already made by proposer $p$. 
Efficient Implementation

Rejecting/accepting proposals:

- Does receiver $r$ prefer proposer $p$ to proposer $p'$?
  - Using original preference list would be slow
- For each receiver, create $inverse$ of preference list of proposers.
- Constant time access for each query after $O(n)$ preprocessing per receiver. $O(n^2)$ total preprocessing cost.

- Proposer 3 or proposer 6?

<table>
<thead>
<tr>
<th>$r$</th>
<th>Proposer 1st</th>
<th>Proposer 2nd</th>
<th>Proposer 3rd</th>
<th>Proposer 4th</th>
<th>Proposer 5th</th>
<th>Proposer 6th</th>
<th>Proposer 7th</th>
<th>Proposer 8th</th>
</tr>
</thead>
<tbody>
<tr>
<td>pref</td>
<td>8</td>
<td>3</td>
<td>7</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>

**Example:**

<table>
<thead>
<tr>
<th>$r$</th>
<th>Proposer 1st</th>
<th>Proposer 2nd</th>
<th>Proposer 3rd</th>
<th>Proposer 4th</th>
<th>Proposer 5th</th>
<th>Proposer 6th</th>
<th>Proposer 7th</th>
<th>Proposer 8th</th>
</tr>
</thead>
<tbody>
<tr>
<td>inverse</td>
<td>4th</td>
<td>8th</td>
<td>2nd</td>
<td>5th</td>
<td>6th</td>
<td>7th</td>
<td>3rd</td>
<td>1st</td>
</tr>
</tbody>
</table>

```plaintext
for i = 1 to n
    inverse[pref[i]] = i
```

$r$ prefers proposer 3 to 6

Propose-And-Reject Algorithm

Propose-and-reject algorithm: [Gale-Shapley 1962]

Intuitive method that guarantees to find a stable matching.

- Members of one group $P$ make proposals, the other group $R$ receives proposals

```plaintext
Initialize each person to be free.
while (some p in P is free and hasn't proposed to everyone in R) {
    Choose some such free p in P who hasn't proposed to everyone in R
    r = 1st person on p's preference list to whom p has not yet proposed
    if (r is free)
        tentatively match (p,r) // p and r both engaged, no longer free
    else if (r prefers p to current tentative match p')
        replace (p',r) by (p,r) // p now engaged, p' now free
    else
        r rejects p
}