Stable Matching & Asymptotic Analysis
Reminders

• Confirm that your GitLab account is set up & you have a repository: https://evolene.cs.williams.edu/cs256-f22/\(<your-userid>/cs256-hw00

  • Assignment 0 due **Wed, September 14 at 10 pm**

• Bill’s office hours:
  • (Today) 11-noon
  • (Tomorrow) 3-4:30 pm
  • (Wednesday) 1:30-3pm

TAs: Max Enis, Andrew Megalaa, Max Kan, Ye Shu

• Meeting this afternoon to figure out TA help schedule
# Stable Matching Problem

**Input.** A set $H$ of $n$ hospitals, a set $D$ of $n$ doctors, and their preferences

<table>
<thead>
<tr>
<th>Hospital Preferences</th>
<th>Doctor Preferences</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1st</strong></td>
<td><strong>2nd</strong></td>
</tr>
<tr>
<td>MA</td>
<td>Aamir</td>
</tr>
<tr>
<td>NH</td>
<td>Aamir</td>
</tr>
<tr>
<td>OH</td>
<td>Chris</td>
</tr>
</tbody>
</table>
Stable Matching Problem

Input. A set $H$ of $n$ hospitals, a set $D$ of $n$ doctors, and their preferences

Goal. Create a matching $M$ that assigns each doctor to a single hospital, and each hospital a single doctor (this is called a perfect matching) s.t. there are no unstable pairs. That is, there is no pair $(h, d) \in H \times D$ where both

- $h$ prefers $d$ to its current match in $M$, and
- $d$ prefers $h$ to its current match in $M$
A First Attempt

Proceed **greedily** in rounds until matched. In each round:

- Each hospital makes an offer to its top available candidate.
- Each doctor *accepts* its top offer (irrevocable contract) and *rejects* any others.

Does anything go wrong? Let’s try it!

<table>
<thead>
<tr>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>MA</td>
<td>Aamir</td>
<td>Chris</td>
</tr>
<tr>
<td>NH</td>
<td>Aamir</td>
<td>Beth</td>
</tr>
<tr>
<td>OH</td>
<td>Chris</td>
<td>Beth</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aamir</td>
<td>OH</td>
<td>NH</td>
</tr>
<tr>
<td>Beth</td>
<td>MA</td>
<td>OH</td>
</tr>
<tr>
<td>Chris</td>
<td>MA</td>
<td>NH</td>
</tr>
</tbody>
</table>
A First Attempt

Proceed greedily in rounds until matched.

- (Round 1) MA → Aamir, NH → Aamir, OH → Chris

<table>
<thead>
<tr>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>MA</td>
<td>Aamir</td>
<td>Chris</td>
</tr>
<tr>
<td>NH</td>
<td>Aamir</td>
<td>Beth</td>
</tr>
<tr>
<td>OH</td>
<td>Chris</td>
<td>Beth</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aamir</td>
<td>OH</td>
<td>NH</td>
</tr>
<tr>
<td>Beth</td>
<td>MA</td>
<td>OH</td>
</tr>
<tr>
<td>Chris</td>
<td>MA</td>
<td>NH</td>
</tr>
</tbody>
</table>
A First Attempt

Proceed greedily in rounds until matched.

- (Round 1) MA → Aamir, NH → Aamir, OH → Chris
- (Round 1) Aamir rejects MA, accepts NH, Chris accepts OH
A First Attempt

Proceed greedily in rounds until matched.

- (Round 1) MA → Aamir, NH → Aamir, OH → Chris
- (Round 1) Aamir rejects MA, accepts NH, Chris accepts OH
- (Round 2) Only Beth and MA left, and must match

<table>
<thead>
<tr>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>MA</td>
<td>Aamir</td>
<td>Chris</td>
</tr>
<tr>
<td>NH</td>
<td>Aamir</td>
<td>Beth</td>
</tr>
<tr>
<td>OH</td>
<td>Chris</td>
<td>Beth</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aamir</td>
<td>MA</td>
<td>NH</td>
</tr>
<tr>
<td>Beth</td>
<td>MA</td>
<td>OH</td>
</tr>
<tr>
<td>Chris</td>
<td>MA</td>
<td>NH</td>
</tr>
</tbody>
</table>
A First Attempt

Proceed greedily in rounds until matched.

- (Round 1) MA → Aamir, NH → Aamir, OH → Chris
- (Round 1) Aamir rejects MA, accepts NH, Chris accepts OH
- (Round 2) Only Beth and MA left, and must match

Is this a stable matching?

<table>
<thead>
<tr>
<th></th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>MA</td>
<td>Aamir</td>
<td>Chris</td>
<td>Beth</td>
</tr>
<tr>
<td>NH</td>
<td>Aamir</td>
<td>Beth</td>
<td>Chris</td>
</tr>
<tr>
<td>OH</td>
<td>Chris</td>
<td>Beth</td>
<td>Aamir</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aamir</td>
<td>OH</td>
<td>NH</td>
<td>MA</td>
</tr>
<tr>
<td>Beth</td>
<td>MA</td>
<td>OH</td>
<td>NH</td>
</tr>
<tr>
<td>Chris</td>
<td>MA</td>
<td>NH</td>
<td>OH</td>
</tr>
</tbody>
</table>
A First Attempt

Proceed greedily in rounds until matched.

- (Round 1) MA → Aamir, NH → Aamir, OH → Chris
- (Round 1) Aamir rejects MA, accepts NH, Chris accepts OH
- (Round 2) Only Beth and MA left, and must match

Is this a stable matching?

- Unstable pair: (MA, Chris). What could have avoided it?

<table>
<thead>
<tr>
<th></th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>MA</td>
<td>Aamir</td>
<td>Chris</td>
<td>Beth</td>
</tr>
<tr>
<td>NH</td>
<td>Aamir</td>
<td>Beth</td>
<td>Chris</td>
</tr>
<tr>
<td>OH</td>
<td>Chris</td>
<td>Beth</td>
<td>Aamir</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aamir</td>
<td>OH</td>
<td>NH</td>
<td>MA</td>
</tr>
<tr>
<td>Beth</td>
<td>MA</td>
<td>OH</td>
<td>NH</td>
</tr>
<tr>
<td>Chris</td>
<td>MA</td>
<td>NH</td>
<td>OH</td>
</tr>
</tbody>
</table>
False Starts are a Problem

- We want to prove: a **stable matching** always exists
- One way:
  - Give an algorithm to find a stable matching
  - Prove that it is always successful
  - *Constructive method*

Gale-Shapely Deferred Acceptance Algorithm*
Propose-Reject Algorithm

Initialize each doctor $d$ and hospital $h$ as Free

while there is a free doctor who hasn’t proposed to every hospital do

Choose a free doctor $d$

$h \leftarrow$ first hospital on $d$’s list to whom $d$ has not yet proposed

if $h$ is Free then

$d$ and $h$ are Matched

else if $h$ prefers $d$ to its current match $d'$ then

$d$ and $h$ are Matched and $d'$ is Free

else

$h$ rejects $d$ and remains Free

end if

end while
Observations

(Write these down, we’ll use them later)

**Observation 1.** A doctor proposes at most $n$ times, to $n$ different hospitals.
Propose-Reject Algorithm

Initialize each doctor $d$ and hospital $h$ as Free

while there is a free doctor who hasn’t proposed to every hospital do

Choose a free doctor $d$

$h \leftarrow$ first hospital on $d$’s list to whom $d$ has not yet proposed

if $h$ is Free then

$d$ and $h$ are Matched

else if $h$ prefers $d$ to its current match $d'$ then

$d$ and $h$ are Matched and $d'$ is Free

else

$h$ rejects $d$ and remains Free

end if

end while

Doctors only propose to hospitals that they have not yet proposed to
Observations

(Write these down, we’ll use them later)

**Observation 1.** A doctor proposes at most $n$ times, to $n$ different hospitals.

**Observation 2.** Once a hospital is matched, it never becomes unmatched, it only “trades up”.
Propose-Reject Algorithm

Initialize each doctor $d$ and hospital $h$ as $Free$

while there is a free doctor who hasn’t proposed to every hospital do

Choose a free doctor $d$

$h \leftarrow$ first hospital on $d$’s list to whom $d$ has not yet proposed

if $h$ is $Free$ then

$d$ and $h$ are $Matched$

else if $h$ prefers $d$ to its current match $d'$ then

$d$ and $h$ are $Matched$ and $d'$ is $Free$

else

$h$ rejects $d$ and remains $Free$

end if

end while
Observations

(Write these down, we’ll use them later)

**Observation 1.** A doctor proposes at most $n$ times, to $n$ different hospitals.

**Observation 2.** Once a hospital is matched, it never becomes unmatched, it only “trades up”.

Now let’s make and prove some claims about the algorithm.
Algorithm Analysis

Claim 1. The propose-reject algorithm terminates after at most $n^2$ iterations of the \textbf{while} loop.

Proof. The proof directly analyzes the structure of the algorithm.

1. A doctor proposes during each iteration of the \textbf{while} loop
Propose-Reject Algorithm

Initialize each doctor \( d \) and hospital \( h \) as \( \text{Free} \)

\[ \text{while} \ \text{there is a free doctor who hasn’t proposed to every hospital} \ \text{do} \]

Choose a free doctor \( d \)

\[ h \leftarrow \text{first hospital on } d\text{’s list to whom } d \text{ has not yet proposed} \]

if \( h \) is \( \text{Free} \) then

\( d \) and \( h \) are \( \text{Matched} \)

else if \( h \) prefers \( d \) to its current match \( d' \) then

\( d \) and \( h \) are \( \text{Matched} \) and \( d' \) is \( \text{Free} \)

else

\( h \) rejects \( d \) and remains \( \text{Free} \)

end if

end while
Algorithm Analysis

Claim 1. The propose-reject algorithm terminates after at most $n^2$ iterations of the while loop.

Proof. The proof directly analyzes the structure of the algorithm.

1. A doctor proposes during each iteration of the while loop

2. Since there are $n$ doctors and each can propose to at most $n$ different hospitals, the while loop can execute at most $n^2$ times.
Claim 2. The propose-reject algorithm returns a perfect matching.

Proof. The proof is by contradiction. Suppose the algorithm yields an imperfect matching.

1. Since we do not allow many-to-one relationships, there must be both a doctor \( d \) and a hospital \( h \) who are unmatched.

2. By Observation 2, \( h \) was never proposed to by anyone, which includes \( d \).

3. But if \( d \) is still free, then, by the while loop condition, \( d \) must have proposed to every hospital, including \( h \). This is a contradiction.
Algorithm Analysis

Claim 3. The perfect matching yielded by the algorithm is *stable*.

Proof. The proof is by contradiction. Suppose the algorithm yields an unstable perfect matching.

1. Then there exist two pairs \((d_1, h_1)\) and \((d_2, h_2)\) such that \(d_1\) and \(h_2\) prefer each other to their current assignment. In other words, the rankings look something like:

   \[
   d_1 : \ldots, h_2, \ldots, h_1, \ldots \quad \text{and} \quad h_2 : \ldots, d_1, \ldots, d_2, \ldots.
   \]

2. Since \(d_1\) ranks \(h_2\) higher than \(h_1\), \(d_1\) proposed to \(h_2\) sometime before proposing to \(h_1\).

3. But by **Observation 2**, \(h_2\) only ever trades up, so \(d_2\) must be ranked higher than \(d_1\). This is a contradiction.
What Have We Shown?

So far we have analyzed the algorithm in a couple of ways:

- We proved key properties about its output
  - It yields perfect matchings (Claim 2)
  - It yields stable matchings (Claim 3)
- We showed that the \texttt{while} loop executes at most $n^2$ times (Claim 1)
  - \textbf{Question}: Does this mean the algorithm is $O(n^2)$?
What Have We Shown?

We’ve specified the algorithm using a powerful and abstract pseudocode.

• Our pseudocode ignores data representation

We can reason about correctness, but not efficiency.

• Efficiency comes when we add the data structures!
Representing the Input

**Idea:** Order the doctors arbitrarily from 1 to $n$. Similarly, arbitrarily order the hospitals from 1 to $n$. A ranking list for doctors is an $n \times n$ matrix $D$, where position $D(i, j)$ gives the $j^{th}$ favorite hospital for doctor $i$. Similarly, construct matrix $H$ for hospitals.

Aamir (1), Beth (2), Chris (3)  
MA (1), NH (2), OH (3)

<table>
<thead>
<tr>
<th></th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aamir</td>
<td>OH</td>
<td>NH</td>
<td>MA</td>
</tr>
<tr>
<td>Beth</td>
<td>MA</td>
<td>OH</td>
<td>NH</td>
</tr>
<tr>
<td>Chris</td>
<td>MA</td>
<td>NH</td>
<td>OH</td>
</tr>
</tbody>
</table>

Doctor 1 (Aamir) ranks Hospital 3 (OH) first
Doctor 3 (Chris) ranks Hospital 2 (NH) second
Identifying Free Doctors

**Idea:** Use a queue! A doubly-linked list allows enqueuing and dequeuing in $O(1)$ time.

- Each doctor that is free is stored in the queue.
- Matching a doctor means dequeuing them.
- Unmatching means putting the doctor back into the queue.
Identifying Next Proposal

For each doctor, we need to know the highest ranked hospital that they have not yet proposed to.

Idea: A particular doctor’s preferences are represented by a row in the matrix $D$. A given doctor $i$ will propose in preference order, i.e., from left to right across row $i$.

For each doctor, maintain a counter that is incremented after each proposal. The counter for doctor $i$ is the index into the preference array at row $i$ of $D$. 
Tracking Matches

We need to know which doctor is matched to which hospital (and vice versa). Since matchings are symmetric, we only need to keep track of one direction.

**Idea**: Keep track of each hospital's match using an array of length $n$. Call this array $matched$.

$matched(i) = j$ means that hospital $i$ is matched to doctor $j$

$matched(i) = -1$ means that hospital $i$ is unmatched
Tracking Hospital Preferences

We need to know if a hospital $h$ prefers its current partner to the doctor who just proposed to it.

**Idea:** Create what is called an inverted index of the $H$ matrix (hospital preference matrix), which we will call $R$ ($R$ for ranks). For a given hospital, $R$ doesn’t store it’s preference list; instead, $R$ stores the rank (1 to $n$) of each doctor. So to compare a hospital $h$’s ranking of two doctors, $i$ and $j$, we can check $R(h, i)$ and compare it to $R(h, j)$.

We can build the inverted index in $O(n^2)$ time by consulting $H$ (a one-time setup cost), and with it, we compare two doctors rankings in $O(1)$ time.
Inverted Index Example

- Let’s use our running example where we’ve numbered our 3 hospitals and 3 doctors as follows:

  **Doctors** 1: Aamir, 2: Beth, 3: Chris  
  **Hospitals** 1: MA, 2: NH, and 3: OH

- In our hospital preference table (left), each row specifies a hospital’s preferences for doctors in descending order. So in a given hospital row, the first column is the hospital’s first choice, the second column second...

- In our inverted index (right), each row specifies a hospital’s ranks for doctors, indexed using the doctors’ numbers. So in a given hospital row, the first column is the ranking of the first doctor, the second column is the ranking of the second doctor...

  - \( R(i, j) \) stores the hospital i’s ranking (ranging from 1…n) for doctor j

### Hospital Preferences (visual)

<table>
<thead>
<tr>
<th></th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>MA</td>
<td>Aamir</td>
<td>Chris</td>
<td>Beth</td>
</tr>
<tr>
<td>NH</td>
<td>Aamir</td>
<td>Beth</td>
<td>Chris</td>
</tr>
<tr>
<td>OH</td>
<td>Chris</td>
<td>Beth</td>
<td>Aamir</td>
</tr>
</tbody>
</table>

### Inverted Index \( R \)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Inverted Index Example

- We can query the inverted index in $O(1)$ to check if a hospital prefers one doctor to another.
- Suppose we wanted to check whether NH prefers Chris or Aamir:
  - NH is hospital 2, Chris is doctor 3, and Aamir is doctor 1
  - $R(2,3)$ stores NH's ranking for Chris, and $R(2,1)$ stores NH's ranking for Aamir:
    
    -> $R(2,3) = 3$, while $R(2,1) = 1$, so Aamir is ranked higher!

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

**Doctors** 1: Aamir, 2: Beth, 3: Chris, and **Hospitals** 1: MA, 2: NH, and 3: OH.
Propose-Reject Algorithm

Initialize each doctor $d$ and hospital $h$ as Free

while there is a free doctor who hasn’t proposed to every hospital do

Choose a free doctor $d$

$h \leftarrow$ first hospital on $d$’s list to whom $d$ has not yet proposed

if $h$ is Free then

d and $h$ are Matched

else if $h$ prefers $d$ to its current match $d'$ then

d and $h$ are Matched and $d'$ is Free

else

$h$ rejects $d$ and remains Free

end if

end while