CS3000: Algorithms & Data
Paul Hand

Lecture 3:
- Stable Matching: Gale-Shapley Algorithm
- Asymptotic Analysis

Jan 14, 2019

Stable Matching Problem

- Many job candidates (eg. doctors). Many jobs (eg. residency programs). You are to assign candidates to jobs. How should you do it?

Stable Matching problem – What makes an output good?

No candidate-job pair prefers each other over what they have.

Stable Matching Problem

- Many job candidates (eg. doctors). Many jobs (eg. residency programs). You are to assign candidates to jobs. How should you do it?

A matching is **stable** if it has no **instabilities**

An **instability** is

- $(c, j) \in M$, j' unmatched, and $c: j' \succ j$.
- $(c, j) \in M$, c' unmatched, and $j: c' \succ c$
- $(c, j) \in M$ & $(c', j') \in M$ but $c: j' \succ j$, $j': c \succ c'$

Stable Matching – Questions

- For any set of preferences, does a stable matching exist?
- Can there be more than one stable matching? yes
- How can you find one if it exists?

If no, what do we want alg to do

Gale-Shapley algorimthm.

Gale-Shapley Algorithm

- Let M be empty
- While (some job j is unmatched):
  - If (j has offered a job to everyone): break
  - Else: let c be the highest-ranked candidate to which j has not yet offered a job
  - j makes an offer to c:
    - If (c is unmatched):
      - c accepts, add (c,j) to M
    - ElseIf (c is matched to j′ & c: j′ > j):
      - c rejects, do nothing
    - ElseIf (c is matched to j′ & c: j > j′):
      - c accepts, remove (c,j′) from M and add (c,j) to M
- Output M

Gale-Shapley Algorithm

```
• Let M be empty
• While (some job j is unmatched):
   • If (j has offered a job to everyone): break
   • Else: let c be the highest-ranked candidate
     to which j has not yet offered a job
   • j makes an offer to c:
      • If (c is unmatched):
         • c accepts, add (c,j) to M
      • ElseIf (c is matched to j′ & c: j′ > j):
         • c rejects, do nothing
      • ElseIf (c is matched to j′ & c: j > j′):
         • c accepts, remove (c,j′) from M and
           add (c,j) to M
• Output M
```

What matching does the algorithm give this data for jobs (j1 and j2) and candidates (c1 and c2)?

	1st	2nd
j1	c1	c2
j2	c2	c1

	1st	2nd
c1	j2	j1
c2	j1	j2

Gale-Shapley Algorithm

- Questions about the Gale-Shapley Algorithm:
  - Will this algorithm terminate? After how long?
  - Does it output a perfect matching?
  - Does it output a stable matching?
  - How do we implement this algorithm efficiently?

Observations about GS

```
• Let M be empty
• While (some job j is unmatched):
  • If (j has offered a job to everyone): break
  • Else: let c be the highest-ranked candidate
    to which j has not yet offered a job
  • j makes an offer to c:
    • If (c is unmatched):
      • c accepts, add (c,j) to M
    • ElseIf (c is matched to j' & c: j' > j):
      • c rejects, do nothing
    • ElseIf (c is matched to j' & c: j > j'):
      • c accepts, remove (c,j') from M and
        add (c,j) to M
• Output M
```

- At all steps, the state of the algorithm is matching.
- Jobs make offers in descending order
- Candidates that get a job never become unemployed
- Candidates accept offers in ascending order

Does the GS algorithm terminate?

```
• Let M be empty
• While (some job j is unmatched):
   • If (j has offered a job to everyone): break
   • Else: let c be the highest-ranked candidate
     to which j has not yet offered a job
   • j makes an offer to c:
      • If (c is unmatched):
         • c accepts, add (c,j) to M
      • ElseIf (c is matched to j′ & c: j′ > j):
         • c rejects, do nothing
      • ElseIf (c is matched to j′ & c: j > j′):
         • c accepts, remove (c,j′) from M and
           add (c,j) to M
• Output M
```

- **Claim:** The GS algorithm terminates after n^2 ← no more than iterations of the main loop

Job offer made at every itteration.

only n^2 possible offers

none repeated.

Is the output a perfect matching?

```
• Let M be empty
• While (some job j is unmatched):
    • If (j has offered a job to everyone): break
    • Else: let c be the highest-ranked candidate
      to which j has not yet offered a job
    • j makes an offer to c:
        • If (c is unmatched):
            • c accepts, add (c,j) to M
        • ElseIf (c is matched to j' & c: j' > j):
            • c rejects, do nothing
        • ElseIf (c is matched to j' & c: j > j'):
            • c accepts, remove (c,j') from M and
              add (c,j) to M
• Output M
```

- **Claim:** The GS algorithm outputs a perfect matching (all jobs are matched and all candidates are matched).

Contradiction:

Assume matching not perfect.

Case I: Job j has no candidate
As alg gives matching, at most n-1 candidates have a job candidate c is unmatched.
Because alg terminated, j offered job to c.
So c is matched therefore,
CONTRADICTION.

Case II: If cand c has no job.
There is job without candidate.

See case 1.

Is the output a stable matching?

```
• Let M be empty
• While (some job j is unmatched):
  • If (j has offered a job to everyone): break
  • Else: let c be the highest-ranked candidate
    to which j has not yet offered a job
  • j makes an offer to c:
    • If (c is unmatched):
      • c accepts, add (c,j) to M
    • ElseIf (c is matched to j' & c: j' > j):
      • c rejects, do nothing
    • ElseIf (c is matched to j' & c: j > j'):
      • c accepts, remove (c,j') from M and
        add (c,j) to M
• Output M
```

An instability is

- ~~$(c, j) \in M$, j' unmatched, and $c: j' > j$~~
- ~~$(c, j) \in M$, c' unmatched, and $j: c' > c$~~
- $(c, j) \in M$ & $(c', j') \in M$ but $c: j' > j$, $j': c > c'$

matching is perfect.

- **Claim:** The GS algorithm outputs a stable matching.
- Proof by contradiction:
  Suppose there is an instability (c, j)

(c, j) $\quad c: j' > j$

(c', j') $\quad j': c > c'$

j' offered job to c before d

Because c's job quality only increases, c's job is always at least as good as j'

But c ended up with $j < j'$

✗

Running time of GS?

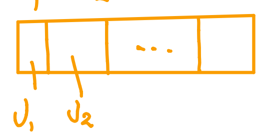
```
• Let M be empty
• While (some job j is unmatched):
  • If (j has offered a job to everyone): break
  • Else: let c be the highest-ranked candidate
    to which j has not yet offered a job
  • j makes an offer to c:
    • If (c is unmatched):
      • c accepts, add (c,j) to M
    • ElseIf (c is matched to j' & c: j' > j):
      • c rejects, do nothing
    • ElseIf (c is matched to j' & c: j > j'):
      • c accepts, remove (c,j') from M and
        add (c,j) to M
• Output M
```

n^2 iteration

Decide if c prefers j to j'

Store C's preferance

$C: j_1 > j_2 > \dots > j_n$



May take n steps to find.

total $\approx n^2 \times n \simeq n^3$

- **Running Time:**
  - A straightforward implementation requires $\approx n^3$ operations, $\approx n^2$ space

memory

Better data structure

```
• Let M be empty
• While (some job j is unmatched):
    • If (j has offered a job to everyone): break
    • Else: let c be the highest-ranked candidate
      to which j has not yet offered a job
    • j makes an offer to c:
        • If (c is unmatched):
            • c accepts, add (c,j) to M
        • ElseIf (c is matched to j' & c: j' > j):
            • c rejects, do nothing
        • ElseIf (c is matched to j' & c: j > j'):
            • c accepts, remove (c,j') from M and
              add (c,j) to M
• Output M
```

	1st	2nd	3rd	4th	5th
Alice	CH	MGH	BW	MTA	BID
Bob	BID	BW	MTA	MGH	CH
Clara	BW	BID	MTA	CH	MGH
Dorit	MGH	CH	MTA	BID	BW
Ernie	MTA	BW	CH	BID	MGH

→

	MGH	BW	BID	MTA	CH
Alice	2nd	3rd	5th	4th	1st
Bob	4th	2nd	1st	3rd	5th
Clara	5th	1st	2nd	3rd	4th
Dorit	1st	5th	4th	3rd	2nd
Ernie	5th	2nd	4th	1st	3rd

- **Running Time:**
  - A careful implementation requires $\approx n^2$ operations, $\approx n^2$ space

cost of n lookup to determine

Does Clara prefer MGH OR CH?

cost of 2 mem lookups to determine

Notes for instructor
Students may ignore
because they are repeated
elsewhere

Is this algorithm fair?

No

consider the example:

$c_1 : j_1 > j_2$ $\quad$ $j_1 : c_2 > c_1$

c_2 $j_2 > j_1$ $\quad$ $j_2 : c_1 > c_2$

According to the preferences, there will always be a paired partner that is unhappy.

So if the jobs are happy the candidates are unhappy and vice versa.

The algo prioritises a certain groups preference.

IT DOESNT TREAT JOBS AND CANDIDATES THE SAME.

Proofs:

Termination:

Each loop makes ~~at~~ ~~most~~ one new offer.
Only n^2 total possible offers

Perfect Matching:

Suppose a job is unmatched.
- Job offer was made to all candidates
- All candidates have a job
- So some candidate is matched with this job

Contradiction

Suppose a candidate is unmatched.
- Some job is unmatched. Contradiction

Stability:

As matching is perfect, only possible instability is $(c, j) \in M$, $(c', j') \in M$ and $c : j' > j$, $j' : c > c'$

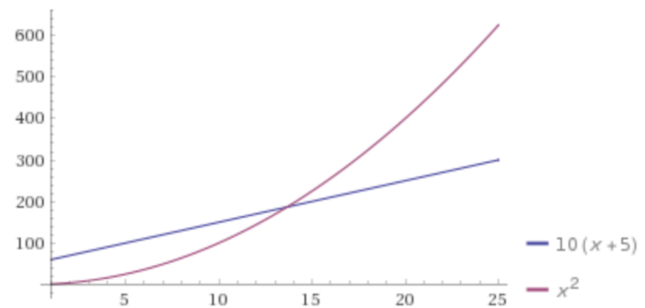
At some point, j' offered to c. c had a job at least as good as j'. c has a job at least as good as j'. Contradiction.

Asymptotic Analysis

Analyzing run time of algorithms

- Predicting the wall-clock time of an algorithm is basically impossible.
  - What machine will actually run the algorithm?
  - Impossible to exactly count “operations”?
  - Which data will it be applied to?

- What do we do instead?
  - We compare how the algorithm scales with lots of data.



Common computational complexity rates (and what they mean in time)

	linear n	$n \log_2 n$	quadratic n^2	cubic n^3	exponential 1.5^n	exponential 2^n	factorial $n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1{,}000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10{,}000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100{,}000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1{,}000{,}000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

1 000 000 steps per second

Efficiency matters more than problem size.

Common computational complexity rates (and what they mean in time)

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1{,}000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10{,}000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100{,}000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1{,}000{,}000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

n^2 ← things to do.

$$\binom{n}{2} = \frac{n(n-1)}{2} \approx \frac{n^2}{2}$$

Activity:

Suppose 1 million write an essay for a standardized test each year. You have code that takes two essays as input and outputs if there is plagiarism. You want to determine if there is any plagiarism by comparing all possible pairs of essays. Roughly how long will it take?

Common computational complexity rates (and what they mean in time)

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1{,}000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10{,}000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100{,}000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1{,}000{,}000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

2^{50} options ← n

Activity:

Suppose someone's password was an arbitrary sequence of 50 bits. Someone wants to hack it by trying all possible passwords. Roughly how long will this take?

Asymptotic Order Of Growth

- **"Big-Oh" Notation:** $f(n) = O\big(g(n)\big)$ if there exists $c \in (0, \infty)$ and $n_0 \in \mathbb{N}$ such that $f(n) \leq c \cdot g(n)$ for every $n \geq n_0$.
  - Asymptotic version of $f(n) \leq g(n)$
  - Roughly equivalent to $\lim_{n \to \infty} \frac{f(n)}{g(n)} < \infty$

Asymptotic Order Of Growth

- **"Big-Oh" Notation:** $f(n) = O\big(g(n)\big)$ if there exists $c \in (0, \infty)$ and $n_0 \in \mathbb{N}$ such that $f(n) \leq c \cdot g(n)$ for every $n \geq n_0$.
  - Asymptotic version of $f(n) \leq g(n)$
  - Roughly equivalent to $\lim_{n \to \infty} \frac{f(n)}{g(n)} < \infty$

- Activity: Which of these statements are true?
  - $3n^2 + n = O(n^2)$
  - $n^3 = O(n^2)$
  - $10n^4 = O(n^5)$
  - $\log_2 n = O(\log_{16} n)$
  - $n \log_2(n^2) = O(n \log_2 n)$

Big-Oh Rules

- **Constant factors can be ignored**
  - $\forall C > 0 \quad Cn = O(n)$
- **Smaller exponents are Big-Oh of larger exponents**
  - $\forall a > b \quad n^b = O(n^a)$
- **Any logarithm is Big-Oh of any polynomial**
  - $\forall a, \varepsilon > 0 \quad \log_2^a n = O(n^\varepsilon)$
- **Any polynomial is Big-Oh of any exponential**
  - $\forall\ a > 0, b > 1 \quad n^a = O(b^n)$
- **Lower order terms can be dropped**
  - $n^2 + n^{3/2} + n = O(n^2)$

Asymptotic Order Of Growth

- **"Big-Omega" Notation:** $f(n) = \Omega\big(g(n)\big)$ if there exists $c \in (0, \infty)$ and $n_0 \in \mathbb{N}$ s.t. $f(n) \geq c \cdot g(n)$ for every $n \geq n_0$.
  - Asymptotic version of $f(n) \geq g(n)$
  - Roughly equivalent to $\lim_{n \to \infty} \frac{f(n)}{g(n)} > 0$
- **"Big-Theta" Notation:** $f(n) = \Theta\big(g(n)\big)$ if there exists $c_1 \leq c_2 \in (0, \infty)$ and $n_0 \in \mathbb{N}$ such that $c_2 \cdot g(n) \geq f(n) \geq c_1 \cdot g(n)$ for every $n \geq n_0$.
  - Asymptotic version of $f(n) = g(n)$
  - Roughly equivalent to $\lim_{n \to \infty} \frac{f(n)}{g(n)} \in (0, \infty)$

Asymptotic Running Times

- **We usually write running time as a Big-Theta**
  - Exact time per operation doesn't appear
  - Constant factors do not appear
  - Lower order terms do not appear

- **Examples:**
  - $30 \log_2 n + 45 = \Theta(\log n)$
  - $Cn \log_2 2n = \Theta(n \log n)$
  - $\sum_{i=1}^{n} i = \Theta(n^2)$

Asymptotic Order Of Growth

- **"Little-Oh" Notation:** $f(n) = o\big(g(n)\big)$ if for every $c > 0$ there exists $n_0 \in \mathbb{N}$ s.t. $f(n) < c \cdot g(n)$ for every $n \geq n_0$.
  - Asymptotic version of $f(n) < g(n)$
  - Roughly equivalent to $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$
- **"Little-Omega" Notation:** $f(n) = \omega\big(g(n)\big)$ if for every $c > 0$ there exists $n_0 \in \mathbb{N}$ such that $f(n) > c \cdot g(n)$ for every $n \geq n_0$.
  - Asymptotic version of $f(n) > g(n)$
  - Roughly equivalent to $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$

Activity

- Fill in the blank with the strongest statement that applies $(O, \Omega, \Theta, o, \omega)$:
  - $15\, n \log_2 n$ = ____ $(\log_2 \sqrt{n})$
  - $n^2 =$_____ $(5\, n^2 + n)$
  - $100n =$ _____ $(5\, n^2 + n)$
  - $3^{\log_2 n} = 2^{\log_3 n}$