Stable Matching Problem

Historically a heterosexual, monogmous marrage

- Match a set of men (husband, applicants) with a set of women (wives, positions) optimally

> **What is "optimal"?**
>
> 1. No mutually beneficial swaps
> 2. Maximize *happiness*

Problem

- Applicants in summer internships
- Grad school
- Residency at Med school

We want self-reinforcing process of matching based on performances

> **An *instability* occess**
>
> 1. Applicant A prefers Postion P' better over current employer P
> 2. Position P prefers Applicant A' better over current Applicant A

Defining the 2 sets

> Refer using Set of open Applicant A , Set of open Position P
>
> $$A = a_1, a_2, ..., a_n$$
>
> $$P = p_1, p_2, ..., p_n$$
>
> Where there are n applicants and n positions

Set of all ordered paris $A \times P$ of form (a, p) represents a found pair

- A **Matching**, S, is a set of ordered pairs each from $A \times P$ s.t each member of A and each member of P appears at most once
- A **Perfect Matching** is where each member appears exactly once (only if cardinality matches)

- Each applicant to rank all position (total ordering) a prefers p to p'
- Postions also ranked

> **Set up Preference List**
>
> 1. Every $a \in A$ ranks $p \in P$
> 2. Every $p \in P$ ranks $a \in A$
> 3. An instability occurs when S contains (a, p) and (a', p') such that a prefers p' and p' prefers a (both ways)
> 4. This is **unstable** because a and p' trade and a' and p are left unpaired

Example: **Instablility**

> **Applicants**
>
> | Preference | 1 | 2 | 3 |
> |---|---|---|---|
> | X | A | B | C |
> | Y | B | A | C |
> | Z | A | B | C |
>
> **Positions**
>
> | Preference | 1 | 2 | 3 |
> |---|---|---|---|
> | A | Y | X | Z |
> | B | X | Y | Z |
> | C | X | Y | Z |
>
> **Pairing with instability**
>
> $$\{(X, C), (Y, B), (Z, A)\}$$
>
> - X prefers B over C and B prefers X over Y

1. Does there exist a stable mathcing for every set of preference list? *(Yes)*
2. Given a set of preference lists, can we efficiently construct a stable matching (if there is one)? *(Need to prove)*

Gale-Shapely

> **Gale-Shapely Psuedocode**
> Initiall all $a \in A$ and $p \in P$ are free
> **while** $\exists a$ who is availiable and hasn't offered to every $p \in P$
>   **do** Choose sicha a postion a
>   **Let** p be the highest ranked in a's preference list to whom a has not yet made an offer
>   **if** p is free
>   **then** (a, p) becomes linked
>   **else** p is currently linked to a'
>   **if** p prefers a' to a
>   **then** a remains free
>   **else** w prefers a to a'
> (a, p) become linked
> a' becomes free
> return the set S of linked pairs

Axioms

1. w remains linked from the point of the first offer
2. The offers given to m get progressively better
3. Sequence of offer offered gets worse
4. Position Optimal

Termination base on number of **offers**

- Terminates after n^2 iterations at most $\therefore O(n^2)$

Proofs for Gale-Shapely

> **Proof for G-S algo terminates after n^2 iterations**
> Each interation consists of position making offer to applicant it has not previously offered to. Count the number of offers. The number of offers always increase by 1 for each iteration. the total number of offers is bounded by n^2 (n applicants and n postions). The loop terminates after n^2

> **Theorem:** *If a position is open at any point in the algorithm, then there muyst be an applicant to which that position has not yet been offered*
> **Proof:**
> Suppose not, suppose that at some point, p is open. But p has already made offers to all applicant. Then all applicants must be matched because G-S is bounded by n^2. But this is a contradiction $\therefore p$ must be matched

> **Theorem:** *The Set S returned at termination is a perfect matching*
> **Proof:**
> Suppose not, suppose the algorithm terminates with a open position. However, this position must have offered to every applicant. If the position made an offer to every applicant, it must be matched to at least 1. $\therefore$ this contradicts the previous theorem.

> **Theorem:** *The Set$S$ of matches returned by the G-S algo is a stable matching*
> **Proof:**
>
> 1. Suppose not, suppose $\exists(a, p) \in S$ and $(a', p') \in S$ s.t. a prefers p' to p and p' prefers a to a'.
> 2. During the algo p' offer to a' most recently
> 3. There are 2 cases
>
> - **Case 1:** p' offered to a previously
>   - Then a would have drop p for p'. Since that is not the case, this is a **Contradiction**
> - **Case 2:** p' has not offered to a previously
>   - Then p' prefers a' over a **(Contradiction)**
>
> 4. $\therefore$ G-S is stable

Practice

> **Prove:** If all position have the same preference list and all applicant have the same preference list, then only a single stable mathcing exists
>
> 1. Suppose not, let the applicant rank i is match to some applicant rank j, $i \neq j$
> 2. Assume $i < j$
> 3. $\exists$ another mismatch i' and j', where $j' < i'$
>
> | j **pref** | i **pref** |
> | --- | --- |
> | $\vdots$ | $\vdots$ |
> | i | j' |
> | i' | j |
> | x | x' |
> | $\vdots$ | $\vdots$ |

> Notice that i prefer j' over j
> Notice that j' prefer i over i'
>
> 4. This will cause them to swap
> 5. This property will apply to every single mismatch until there only exist 1 matching

Implementation

1. How do we **efficiently** implement Gale-Shapely
2. If multiple stable matching exist, which one does Gale-Shapely return

If the while loop of the pseudocode executes N^2 times, we will seek an implementation with $O(n^2)$ complexity

Data Structures

1. Queue of availible positions
2. 2 arrays of length `n`
    - position-match index by position
    - applicant-match index by applicant
    - invariant `(a,p)`
        - `pos_match[p] = a`
        - `app_match[a] = p`
3. one or more array for progress
    - `count[p] = # offers p has made`
    - `count[p] + 1 = ranking of next offer`

Matching Process

1. Dequeue an open positions, `p`. Make offer to the applicant who ranks at `count[p] + 1` in `p`'s preference list
2. The applicant accepts if (1) `app_match[a] < lower on app_pref[a]` (2) `app_match[a] == NULL`
3. if the applicant accepts, any dropped position is unpaired and goes back into the queue
4. if the applicant declines `p` goes back into the queue
5. `count[p]` is incremented

Representing Preference lists

1. For position - ordered list, most to least preferred
2. For applicant
    - map from "key" (position #) to the rank

- "inverted" pref list

Understanding the solution

For a given instnace of stable mathcing problem, there may be more than one stable mathcing

Q: Is Gale-Shapely **deterministic**? that is, do all executions of Gale-Shapely yielf the same matching? If so which one

Valid match - position p is a valid match for applicant a if $\exists$ some stable matching in which a is matched with p s.t. (a, p)

Optimality

- **Position optimal** - every position received best valid match
- **Applicant optimal** - every applicant received best valid match

> **Claims**
>
> 1. All executions of Gale-Shapley (as written above) yield a position-optimal assignment, which is stable.
> 2. All executions of Gale Shapely (as written above) yields an applicant pessimal assignment.
>
> **Proof for Claim 1**
>
> 1. Suppose not, suppose G-S ends with some position p that does not have it best valid match
> 2. p must have made an offer to its best valid match
> 3. Consider the first time that any position p is rejected by some valid applicant a
> 4. a movies to some other preferable postion p'
> 5. p' is preferred over p on a's pref list
> 6. Since a is valid with p, $\exists$ a S' s.t (a, p) and (a', p') where $a \neq a'$
> 7. But since the rejection that p experience is the first executed rejection of S, p' connot have been rejected by a
> 8. Since p' offers in order of pref p' prefers a tp a'
> 9. But $(a, p') \notin S'$, $\therefore$ contradiction and there is an instablility in S'

Variations on Stable Matching

- multi-dimensional pref list
- joint optimisation
- roommate: $2N$ people to be paired with one another
- unequal assigments

Closing Thoughts: Steps in algo design

1. Formulate problem
2. Design algo to solve
3. Prove that algo is correct
4. Give bound on algorithms running time