Algorithm II

1. Stable Matching

WU Xiaokun 吴晓堃

xkun.wu [at] gmail

Problem solving steps

1. **formulate** problem with *enough mathematical precision* that:
   - pose concrete questions,
   - motivate smart solutions;
2. **design** an algorithm for the problem;
3. **analyze** the algorithm to:
   - prove it is *correct*,
   - give a bound on running time (thus establish *efficiency*).

Content

- Stable Matching
- Gale-Shapley Algorithm
- Proof of Correctness
- (Exclusive) Optimality
- Context
- Five Representative Problems

Stable Matching

The story

Consider **job recruiting** procedure:

- several companies would each offer *one* position
- a group of students made applications to *every* company
- these two groups are *mutually acceptable*, but each has preferences

Similar situations: PhD admission, apartment renting, marriage, etc.

The story

Consider **job recruiting** procedure:

- several companies would each offer *one* position
- a group of students made applications to *every* company
- these two groups are *mutually acceptable*, but each has preferences

Similar situations: PhD admission, apartment renting, marriage, etc.

Can anything goes wrong?

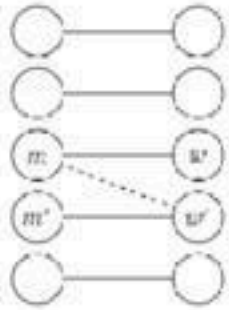
- someone made a side-deal with another company during intern
- some company revoked offer
- circular preference

Everyone likes predictable process

Instability. Consider an "arranged pair" m and w:

- m prefers w' to its current partner w
- w prefers m' to its current partner m

Whenever there's a chance, they break up.



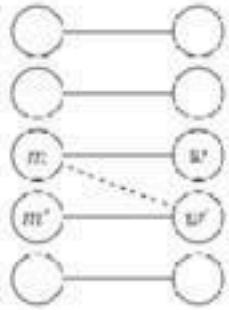
Instability

Everyone likes predictable process

Instability. Consider an "arranged pair" m and w:

- m prefers w' to its current partner w
- w prefers m' to its current partner m

Whenever there's a chance, they break up.



Instability

Stable assignment. Assignment with no instability.

- proceed in a spontaneous way
- individual self-interest prevents side deals

How to formulated this story into a problem *mathematically*?

Stable Matching Problem: Input

Entities. Set $M = \{m_1, m_2, ..., m_n\}$ and $W = \{w_1, w_2, ..., w_n\}$

- each has n entities.

Preference. Each $m \in M$ *ranks* W into R_m, and each $w \in W$ *ranks* M into R_w.

	1st	2nd	3rd
Atlanta	Xavier	Yolanda	Zeus
Boston	Yolanda	Xavier	Zeus
Chicago	Xavier	Yolanda	Zeus

	1st	2nd	3rd
Xavier	Boston	Atlanta	Chicago
Yolanda	Atlanta	Boston	Chicago
Zeus	Atlanta	Boston	Chicago

Matching

Let $M \times W$: set of all possible *ordered pairs* of the form (m, w), where $m \in M$ and $w \in W$.

Matching. a subset of $M \times W$, where each $m \in M$ and $w \in W$ appears in *at most one* pair.

- $m \mapsto i_m, w \mapsto i_w$ are *injective*, where i_m, i_w are partner's index

Matching

Let $M \times W$: set of all possible *ordered pairs* of the form (m, w), where $m \in M$ and $w \in W$.

Matching. a subset of $M \times W$, where each $m \in M$ and $w \in W$ appears in *at most one* pair.

- $m \mapsto i_m, w \mapsto i_w$ are *injective*, where i_m, i_w are partner's index

A matching P is **perfect** if $|P| = |M| = |W| = n$.

	1st	2nd	3rd
Atlanta	Xavier	Yolanda	Zeus
Boston	Yolanda	Xavier	Zeus
Chicago	Xavier	Yolanda	Zeus

	1st	2nd	3rd
Xavier	Boston	Atlanta	Chicago
Yolanda	Atlanta	Boston	Chicago
Zeus	Atlanta	Boston	Chicago

Stable Matching Problem: Output

A **stable matching** is a perfect matching with no instability.

	1st	2nd	3rd
Atlanta	Xavier	Yolanda	Zeus
Boston	Yolanda	Xavier	Zeus
Chicago	Xavier	Yolanda	Zeus

	1st	2nd	3rd
Xavier	Boston	Atlanta	Chicago
Yolanda	Atlanta	Boston	Chicago
Zeus	Atlanta	Boston	Chicago

Stable Matching Problem: Output

A **stable matching** is a perfect matching with no instability.

	1st	2nd	3rd
Atlanta	Xavier	Yolanda	Zeus
Boston	Yolanda	Xavier	Zeus
Chicago	Xavier	Yolanda	Zeus

	1st	2nd	3rd
Xavier	Boston	Atlanta	Chicago
Yolanda	Atlanta	Boston	Chicago
Zeus	Atlanta	Boston	Chicago

Stable Matching Problem. Given preference lists, find a stable matching (if exists).

	1st	2nd	3rd
Atlanta	Xavier	Yolanda	Zeus
Boston	Yolanda	Xavier	Zeus
Chicago	Xavier	Yolanda	Zeus

	1st	2nd	3rd
Xavier	Boston	Atlanta	Chicago
Yolanda	Atlanta	Boston	Chicago
Zeus	Atlanta	Boston	Chicago

Note: X, Y, Z are not most satisfied, any clue?

Gale-Shapley Algorithm

States

Clearly, each entity has two states in our current formulation: `free`, or `matched`.

- Is it enough? Let's brainstorm a scenario where m *proposes* to each $w \in W$.

States

Clearly, each entity has two states in our current formulation: `free`, or `matched`.

- Is it enough? Let's brainstorm a scenario where m *proposes* to each $w \in W$.

If w got proposed:

- might too *rush* to accept: better ones might come later.
- might too *risky* to reject: this could be the best ever comes.

States

Clearly, each entity has two states in our current formulation: `free`, or `matched`.

- Is it enough? Let's brainstorm a scenario where m *proposes* to each $w \in W$.

If w got proposed:

- might too *rush* to accept: better ones might come later.
- might too *risky* to reject: this could be the best ever comes.

We need a third state: `engaged`.

State transitions

1. Initially, everyone is `free`.
2. Let's say an arbitrary m proposes to each $w \in W$:
    1. in order of preference list R_m.
3. suppose w got proposed, and if:
    1. `free`: engage.
    2. `engaged` to m': check its preference list R_w:
        1. m' is higher: reject m, then m will propose to next one.
        2. m is higher: engage m, which makes m' `free`.
4. When everyone is `engaged`, claim it the final matching.

Gale–Shapley algorithm

INPUT: M, W, R_m, R_w

1. $P = \varnothing$; mark $m \in M$ and $w \in W$ `free`;
2. WHILE some $m \in M$ is `free`
    1. w: highest on R_m that m has not yet proposed;
    2. IF w is `free`
        1. Add (m, w) to P;
    3. ELSE IF w prefers m to current partner m'
        1. Replace (m', w) with (m, w), set m' `free`;
    4. ELSE (Nothing happens.);
3. RETURN P;

Demo: G-S

Proof of Correctness

Termination

Observation 1. Each m proposes in decreasing order of preference (getting worse and worse).

Observation 2. Once w engaged, it's never free again, but "trades up" (getting better and better).

Termination

Observation 1. Each m proposes in decreasing order of preference (getting worse and worse).

Observation 2. Once w engaged, it's never free again, but "trades up" (getting better and better).

Claim. G-S algorithm terminates after at most n^2 iterations.
Pf. One of M proposes to a new candidate in each iteration, and there are at most n^2 possible proposals.

Perfect matching

Claim (injection). G-S algorithm outputs a matching.
Pf. [from observations]

- m proposes only if free $\Rightarrow$ matched to at most 1
- w keeps only the best $\Rightarrow$ matched to at most 1

Perfect matching

Claim (injection). G-S algorithm outputs a matching.
Pf. [from observations]

- m proposes only if free $\Rightarrow$ matched to at most 1
- w keeps only the best $\Rightarrow$ matched to at most 1

Claim (surjection). In G-S matching, all M get matched.
Pf. [by contradiction]

- suppose $m \in M$ is still `free` upon termination.
- at least one $w \in W$ is unmatched.
- so w was never proposed to.
- but m proposed to everyone, contradiction.

Perfect matching

Claim (injection). G-S algorithm outputs a matching.
Pf. [from observations]

- m proposes only if free $\Rightarrow$ matched to at most 1
- w keeps only the best $\Rightarrow$ matched to at most 1

Claim (surjection). In G-S matching, all M get matched.
Pf. [by contradiction]

- suppose $m \in M$ is still `free` upon termination.
- at least one $w \in W$ is unmatched.
- so w was never proposed to.
- but m proposed to everyone, contradiction.

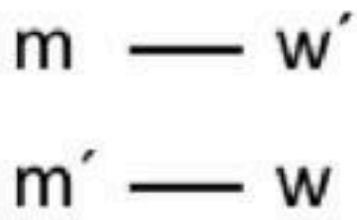
Claim (bijection). G-S algorithm outputs a perfect matching.
Pf. [by counting]

Stability

Claim. [Gale–Shapley 1962] G-S algorithm outputs a stable matching P^*.

Pf. Consider a pair $(m, w) \notin P^*$:

- if m never proposed to w
  - m prefers its G-S partner w' to w
- otherwise, m proposed to w
  - w must rejected m in the end
  - w prefers its G-S partner m' to m



In either case, current matching is more stable.

Quiz: Uniqueness of G-S

Do all executions of Gale–Shapley lead to the same stable matching?

- No, because the algorithm is *nondeterministic*.
- No, because an instance can have several stable matchings.
- Yes, because each instance has a unique stable matching.
- Yes, even though an instance can have several stable matchings and the algorithm is nondeterministic.

Quiz: Uniqueness of G-S

Do all executions of Gale–Shapley lead to the same stable matching?

- No, because the algorithm is *nondeterministic*.
- No, because an instance can have several stable matchings.
- Yes, because each instance has a unique stable matching.
- Yes, even though an instance can have several stable matchings and the algorithm is nondeterministic.

- Nondeterministic? Yes.
- Multiple stable matchings? Yes.

We only show the matching will not change, but is it optimal? How to define optimal?

(Exclusive) Optimality

Completely Clashed Preferences

Consider the following preferences:

	1st	2nd		1st	2nd
m	w	w'	w	m'	m
m'	w'	w	w'	m	m'

- $\{(m, w), (m', w')\}$ is stable:
  - both men are happy, so neither would leave their assigned partner.
- $\{(m', w), (m, w')\}$ is also stable:
  - (complementary) neither women would leave their assigned partner.

Completely Clashed Preferences

Consider the following preferences:

	1st	2nd		1st	2nd
m	w	w'	w	m'	m
m'	w'	w	w'	m	m'

- $\{(m, w), (m', w')\}$ is stable:
  - both men are happy, so neither would leave their assigned partner.
- $\{(m', w), (m, w')\}$ is also stable:
  - (complementary) neither women would leave their assigned partner.

It's possible for an instance to have more than one stable matching.

- Even possible to say they are equally good?

Valid partner

Def. We say m is a **valid partner** of w, if there *exists any* stable matching that contains the pair (m, w).

Valid partner

Def. We say m is a **valid partner** of w, if there *exists any* stable matching that contains the pair (m, w).

	1st	2nd	3rd		1st	2nd	3rd
A	X	Y	Z	**X**	B	A	C
B	Y	X	Z	**Y**	A	B	C
C	X	Y	Z	**Z**	A	B	C

- Both X and Y are valid partners for A.
- Both X and Y are valid partners for B.
- Z is the only valid partner for C

Can you see the stable matchings?

Best-valid assignment

Def. w is the **best-valid partner** of m: if (m, w) is *valid*, and no one else has a higher rank than w is also valid.

- $best(m)$: denote the best-valid partner of m.

Best-valid assignment

Def. w is the **best-valid partner** of m: if (m, w) is *valid*, and no one else has a higher rank than w is also valid.

- $best(m)$: denote the best-valid partner of m.

M-optimal assignment. S^* denote the set of pairs $\{(m, best(m)) : m \in M\}$.

- Is it a (perfect) matching?
- Is it stable?

Best-valid assignment

Def. w is the **best-valid partner** of m: if (m, w) is *valid*, and no one else has a higher rank than w is also valid.

- $best(m)$: denote the best-valid partner of m.

M-optimal assignment. S^* denote the set of pairs $\{(m, best(m)) : m \in M\}$.

- Is it a (perfect) matching?
- Is it stable?

Claim. Every executions of G-S yield S^*.

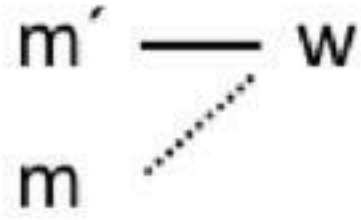
- Remember rules in G-S algorithm: only M propose.

M-optimality

Claim. Every executions of G-S yield S^*.

Pf. suppose one of M matched non-best-valid in S.

1. M propose in decreasing order of preference.
   - rejected by best-valid partner.
2. consider the *first moment* such rejection happened
   - Let m: *first* got rejected,
   - Let w: *first valid* that rejected m,
     - must be: $w = best(m)$.
3. Let m': engaged to w when w rejected m,
   - $\star$: w prefers m' to m.

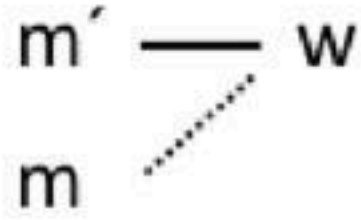


M-optimality (cont.)

Claim. Every executions of G-S yield S^*.

Pf. suppose one of M matched non-best-valid in S.

1. M propose in decreasing order of preference.
2. consider the *first moment* such rejection happened
3. Let m': engaged to w when w rejected m,
   - $\star$: w prefers m' to m.



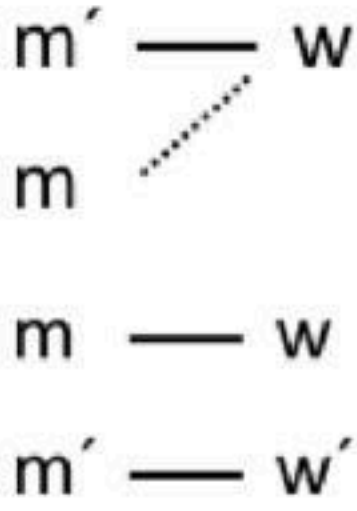
M-optimality (cont.)

Claim. Every executions of G-S yield S^*.

Pf. suppose one of M matched non-best-valid in S.

1. M propose in decreasing order of preference.
2. consider the *first moment* such rejection happened
3. Let m': engaged to w when w rejected m,
   - $\star$: w prefers m' to m.

1. (m, w) is valid $\Rightarrow$ exists stable match S' contains it.
   1. Let w' be the partner of m' in S',
2. m' not rejected by any valid partner *at the moment*.
   1. especially, m' not yet rejected by w',
3. m' not yet proposed to w', but engaged,
   1. $\star$: m' prefers w to w'.
4. (m', w) is an instability in S', a contradiction.



Exclusiveness of Optimality

M-optimality come at the expense of the other side.

- In S^*, each $w \in W$ got the *worst possible* partner.

Exclusiveness of Optimality

M-optimality come at the expense of the other side.

- In S^*, each $w \in W$ got the *worst possible* partner.

Def. m is the **worst-valid partner** of w: if (m, w) is *valid*, and no one else has a lower rank than m is also valid.

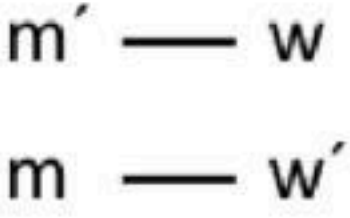
- $worst(w)$: denote the worst-valid partner of w.

W-pessimal

Claim. In S^*, each $w \in W$ is paired with $worst(w)$.

Pf. suppose $(m, w) \in S^*$, but $m \neq worst(w)$.

1. $\exists m', S : (m', w) \in S$,
   1. w ranks m' even lower,
   2. $\star$: w prefers m to m'.
2. Let w': $(m, w') \in S$,
   1. By M-optimality, $w = best(m)$
   2. $\star$: m prefers w to w'.
3. (m, w) is an instability in S^*, a contradiction.



Is it fair?

When preferences clash completely:

- *proposing* side got best possible stable matching;
- the other side got worst possible stable matching.

Someone is destined to end up unhappy.

Is it fair?

When preferences clash completely:

- *proposing* side got best possible stable matching;
- the other side got worst possible stable matching.

Someone is destined to end up unhappy.

The lessen also applies to real life:

- become attractive: reach a higher rank on other people's preference list
- be active: make sure you, instead of your competitors, achieve optimality

Context

Extensions

We made many assumptions in the problem formulation.

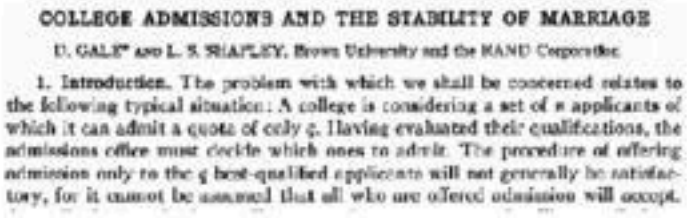
- Some agents declare others as unacceptable.
- Some companies have more than one position.
- Unequal number of positions and students.

2012 Nobel Prize in Economics

Lloyd Shapley. Stable matching theory and Gale–Shapley algorithm.

- original applications: college admissions and opposite-sex marriage.

Alvin Roth. Applied Gale–Shapley to matching med-school students with hospitals, students with schools, and organ donors with patients.



COLLEGE ADMISSIONS AND THE STABILITY OF MARRIAGE

D. GALE* AND L. S. SHAPLEY, Brown University and the RAND Corporation

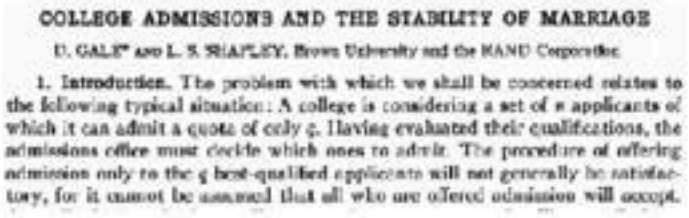
1. Introduction. The problem with which we shall be concerned relates to the following typical situation: A college is considering a set of n applicants of which it can admit a quota of only q. Having evaluated their qualifications, the admissions office must decide which ones to admit. The procedure of offering admission only to the q best-qualified applicants will not generally be satisfactory, for it cannot be assumed that all who are offered admission will accept.

2012 Nobel Prize in Economics

Lloyd Shapley. Stable matching theory and Gale–Shapley algorithm.

- original applications: college admissions and opposite-sex marriage.

Alvin Roth. Applied Gale–Shapley to matching med-school students with hospitals, students with schools, and organ donors with patients.



COLLEGE ADMISSIONS AND THE STABILITY OF MARRIAGE

D. GALE* AND L. S. SHAPLEY, Brown University and the RAND Corporation

1. Introduction. The problem with which we shall be concerned relates to the following typical situation: A college is considering a set of n applicants of which it can admit a quota of only q. Having evaluated their qualifications, the admissions office must decide which ones to admit. The procedure of offering admission only to the q best-qualified applicants will not generally be satisfactory, for it cannot be assumed that all who are offered admission will accept.

While talking algorithms, *it's not just about computer science*.

Five Representative Problems

Recap: Problem solving steps

1. **formulate** problem with *enough mathematical precision* that:
   - pose concrete questions,
   - motivate smart solutions;
2. **design** an algorithm for the problem;
3. **analyze** the algorithm to:
   - prove it is *correct*,
   - give a bound on running time (thus establish *efficiency*).

Milestones

The course is structured by fundamental design techniques.

- learning design patterns helps building your own **knowledge database**
  - organize your KB into categories

Milestones

The course is structured by fundamental design techniques.

- learning design patterns helps building your own **knowledge database**
  - organize your KB into categories
- *subtle changes* in the statement of a problem can have an *enormous effect* on its computational difficulty.

Milestones

The course is structured by fundamental design techniques.

- learning design patterns helps building your own **knowledge database**
  - organize your KB into categories
- *subtle changes* in the statement of a problem can have an *enormous effect* on its computational difficulty.

- **Interval Scheduling**: Greedy algorithms.
- **Weighted Interval Scheduling**: Dynamic programming.
- **Bipartite Matching**: Network flow.
- **Independent Set**: NP-complete.
- **Competitive Facility Location**: PSPACE-complete.