Stability, Popularity, and Lower Quotas

Meghana Nasre

IIT Madras

CAALM 2019

Chennai Mathematical Institute

Jan 22, 2019

Problem Setup

Problem Setup

- A set of men $\mathcal{A}$ (applicants / students /medical interns)

Problem Setup

- A set of men $\mathcal{A}$ (applicants / students /medical interns)
- A set of women $\mathcal{B}$ (jobs / courses / hospitals)

Problem Setup

- A set of men $\mathcal{A}$ (applicants / students /medical interns)
- A set of women $\mathcal{B}$ (jobs / courses / hospitals)
- Each participant has a preference ordering.

Problem Setup

- A set of men $\mathcal{A}$ (applicants / students /medical interns)
- A set of women $\mathcal{B}$ (jobs / courses / hospitals)
- Each participant has a preference ordering.

a_1:	b_1	b_2		b_1:	a_1	a_2
a_2:	b_1	b_2		b_2:	a_1	a_2

- Here preferences are strict and complete.

Problem Setup

- A set of men $\mathcal{A}$ (applicants / students /medical interns)
- A set of women $\mathcal{B}$ (jobs / courses / hospitals)
- Each participant has a preference ordering.

a_1:	b_1	b_2		b_1:	a_1	a_2
a_2:	b_1	b_2		b_2:	a_1	a_2

- Here preferences are strict and complete.

$\boxed{\text{Goal: Assign men to women optimally.}}$

Problem Setup

- A set of men $\mathcal{A}$ (applicants / students /medical interns)
- A set of women $\mathcal{B}$ (jobs / courses / hospitals)
- Each participant has a preference ordering.

a_1:	b_1	$\boxed{b_2}$	b_1:	a_1	a_2
a_2:	$\boxed{b_1}$	b_2	b_2:	a_1	a_2

- Here preferences are strict and complete.

A possible assignment $M = \{(a_1, b_2), (a_2, b_1)\}$.

Stability – a notion of optimality

a_1:	b_1	$\boxed{b_2}$		b_1:	a_1	a_2
a_2:	$\boxed{b_1}$	b_2		b_2:	a_1	a_2

A pair $(a, b) \in E \setminus M$ blocks M if

- Both a and b prefer each other to their current partner in M.

Stability – a notion of optimality

a_1:	b_1	$\boxed{b_2}$		b_1:	a_1	a_2
a_2:	$\boxed{b_1}$	b_2		b_2:	a_1	a_2

A pair $(a, b) \in E \setminus M$ blocks M if

- Both a and b prefer each other to their current partner in M.
- (a_1, b_1): both a_1 and b_1 wish to deviate – blocking pair.

Stability – a notion of optimality

a_1:	b_1	$\boxed{b_2}$		b_1:	a_1	a_2
a_2:	$\boxed{b_1}$	b_2		b_2:	a_1	a_2

A pair $(a, b) \in E \setminus M$ blocks M if

- Both a and b prefer each other to their current partner in M.
- (a_1, b_1): both a_1 and b_1 wish to deviate – blocking pair.

A matching is stable if no pair wishes to deviate.

Stability – a notion of optimality

a_1:	b_1	$\boxed{b_2}$		b_1:	a_1	a_2
a_2:	$\boxed{b_1}$	b_2		b_2:	a_1	a_2

A pair $(a, b) \in E \setminus M$ blocks M if

- Both a and b prefer each other to their current partner in M.
- (a_1, b_1): both a_1 and b_1 wish to deviate – blocking pair.

A matching is stable if no pair wishes to deviate.

$M' = \{(a_1, b_1), (a_2, b_2)\}$ is a stable.

Stability – a notion of optimality

a_1:	b_1	$\boxed{b_2}$		b_1:	a_1	a_2
a_2:	$\boxed{b_1}$	b_2		b_2:	a_1	a_2

A pair $(a, b) \in E \setminus M$ blocks M if

- Both a and b prefer each other to their current partner in M.
- (a_1, b_1): both a_1 and b_1 wish to deviate – blocking pair.

A matching is stable if no pair wishes to deviate.

$M' = \{(a_1, b_1), (a_2, b_2)\}$ is a stable.

Known Facts:

- Every instance admits a stable matching.
- Stable matching can be computed in linear time.
- All stable matchings are perfect.

Today's talk: Three Variants of the SM problem

Variation #1: Incomplete Lists

Variation #1: Incomplete Lists

- Preferences are strict and can be incomplete.

a_1:	b_1	b_2	b_1:	a_1	a_2
a_2:	b_1		b_2:	a_1	

Variation #1: Incomplete Lists

- Preferences are strict and can be incomplete.

a_1:	b_1	b_2		b_1:	a_1	a_2
a_2:	b_1			b_2:	a_1	

- Does a stable matching exist? Yes!

Variation #1: Incomplete Lists

- Preferences are strict and can be incomplete.

a_1:	b_1	b_2		b_1:	a_1	a_2
a_2:	b_1			b_2:	a_1	

- Does a stable matching exist? Yes! $M = \{(a_1, b_1)\}$.

Variation #1: Incomplete Lists

- Preferences are strict and can be incomplete.

a_1:	b_1	b_2		b_1:	a_1	a_2
a_2:	b_1			b_2:	a_1	

- Does a stable matching exist? Yes! $M = \{(a_1, b_1)\}$.

Known Facts:

Variation #1: Incomplete Lists

- Preferences are strict and can be incomplete.

a_1:	b_1	b_2		b_1:	a_1	a_2
a_2:	b_1			b_2:	a_1	

- Does a stable matching exist? Yes! $M = \{(a_1, b_1)\}$.

Known Facts:

- Every instance admits a stable matching; can be computed in linear time.

Variation #1: Incomplete Lists

- Preferences are strict and can be incomplete.

a_1:	b_1	b_2	b_1:	a_1	a_2
a_2:	b_1		b_2:	a_1	

- Does a stable matching exist? Yes! $M = \{(a_1, b_1)\}$.

Known Facts:

- Every instance admits a stable matching; can be computed in linear time.
- All stable matchings are ~~perfect~~

Variation #1: Incomplete Lists

- Preferences are strict and can be incomplete.

a_1:	b_1	b_2	b_1:	a_1	a_2
a_2:	b_1		b_2:	a_1	

- Does a stable matching exist? Yes! $M = \{(a_1, b_1)\}$.

Known Facts:

- Every instance admits a stable matching; can be computed in linear time.
- All stable matchings are ~~perfect~~ of the same size.

Variation #1: Incomplete Lists

- Preferences are strict and can be incomplete.

a_1:	b_1	b_2		b_1:	a_1	a_2
a_2:	b_1			b_2:	a_1	

- Does a stable matching exist? Yes! $M = \{(a_1, b_1)\}$.

Known Facts:

- Every instance admits a stable matching; can be computed in linear time.
- All stable matchings are ~~perfect~~ of the same size.
- Stable matching can be half the size of max. matching.

Variation #1: Incomplete Lists

- Preferences are strict and can be incomplete.

a_1:	b_1	b_2		b_1:	a_1	a_2
a_2:	b_1			b_2:	a_1	

- Does a stable matching exist? Yes! $M = \{(a_1, b_1)\}$.

Known Facts:

- Every instance admits a stable matching; can be computed in linear time.
- All stable matchings are ~~perfect~~ of the same size.
- Stable matching can be half the size of max. matching.

Question: Are there larger optimal matchings?

Variation #2: Incomplete Lists and Ties

Variation #2: Incomplete Lists and Ties

- Preferences can contain ties and can be incomplete.

a_1:	b_1	b_2		b_1:	$(a_1$	$a_2)$
a_2:	b_1			b_2:	a_1	

Variation #2: Incomplete Lists and Ties

- Preferences can contain ties and can be incomplete.

a_1:	b_1	b_2	b_1:	$(a_1$	$a_2)$
a_2:	b_1		b_2:	a_1	

- Redefine blocking pair.
  - A pair $(a, b) \in E \setminus M$ blocks M if
    Both a and b strictly prefer each other to their current partner in M.

Variation #2: Incomplete Lists and Ties

- Preferences can contain ties and can be incomplete.

a_1:	b_1	b_2	b_1:	$(a_1$	$a_2)$
a_2:	b_1		b_2:	a_1	

- Redefine blocking pair.
  - A pair $(a, b) \in E \setminus M$ blocks M if
    Both a and b strictly prefer each other to their current partner in M.
- Does a stable matching exist? Yes!

Variation #2: Incomplete Lists and Ties

- Preferences can contain ties and can be incomplete.

a_1:	b_1	b_2	b_1:	$(a_1$	$a_2)$
a_2:	b_1		b_2:	a_1	

- Redefine blocking pair.
  - A pair $(a, b) \in E \setminus M$ blocks M if
    Both a and b strictly prefer each other to their current partner in M.
- Does a stable matching exist? Yes!
  $M_1 = \{(a_1, b_1)\}$ $\quad$ $M_2 = \{(a_1, b_2), (a_2, b_1)\}$

Variation #2: Incomplete Lists and Ties

- Preferences can contain ties and can be incomplete.

a_1:	b_1	b_2		b_1:	$(a_1$	$a_2)$
a_2:	b_1			b_2:	a_1	

- Redefine blocking pair.
  - A pair $(a, b) \in E \setminus M$ blocks M if
    Both a and b strictly prefer each other to their current partner in M.
- Does a stable matching exist? Yes!
  $M_1 = \{(a_1, b_1)\}$ $\qquad$ $M_2 = \{(a_1, b_2), (a_2, b_1)\}$

Known Facts:

Variation #2: Incomplete Lists and Ties

- Preferences can contain ties and can be incomplete.

a_1:	b_1	b_2		b_1:	$(a_1$	$a_2)$
a_2:	b_1			b_2:	a_1	

- Redefine blocking pair.
  - A pair $(a, b) \in E \setminus M$ blocks M if
    Both a and b strictly prefer each other to their current partner in M.
- Does a stable matching exist? Yes!
  $M_1 = \{(a_1, b_1)\}$ $\quad$ $M_2 = \{(a_1, b_2), (a_2, b_1)\}$

Known Facts:

- Every instance admits a stable matching; can be computed in linear time.

Variation #2: Incomplete Lists and Ties

- Preferences can contain ties and can be incomplete.

a_1:	b_1	b_2		b_1:	$(a_1$	$a_2)$
a_2:	b_1			b_2:	a_1	

- Redefine blocking pair.
  - A pair $(a, b) \in E \setminus M$ blocks M if
    Both a and b strictly prefer each other to their current partner in M.
- Does a stable matching exist? Yes!
  $M_1 = \{(a_1, b_1)\}$ $\quad$ $M_2 = \{(a_1, b_2), (a_2, b_1)\}$

Known Facts:

- Every instance admits a stable matching; can be computed in linear time.
- All stable matchings ~~are of the same size~~ need not be of same size.

Variation #2: Incomplete Lists and Ties

- Preferences can contain ties and can be incomplete.

a_1:	b_1	b_2		b_1:	(a_1	a_2)
a_2:	b_1			b_2:	a_1	

- Redefine blocking pair.
  - A pair $(a, b) \in E \setminus M$ blocks M if
    Both a and b strictly prefer each other to their current partner in M.
- Does a stable matching exist? Yes!
  $M_1 = \{(a_1, b_1)\}$ $\qquad$ $M_2 = \{(a_1, b_2), (a_2, b_1)\}$

Known Facts:

- Every instance admits a stable matching; can be computed in linear time.
- All stable matchings ~~are of the same size~~ need not be of same size.
- A stable matching can be half the size of another stable matching.

Variation #2: Incomplete Lists and Ties

- Preferences can contain ties and can be incomplete.

a_1:	b_1	b_2		b_1:	$(a_1$	$a_2)$
a_2:	b_1			b_2:	a_1	

- Redefine blocking pair.
  - A pair $(a, b) \in E \setminus M$ blocks M if
    Both a and b strictly prefer each other to their current partner in M.
- Does a stable matching exist? Yes!
  $M_1 = \{(a_1, b_1)\}$ $\qquad$ $M_2 = \{(a_1, b_2), (a_2, b_1)\}$

Known Facts:

- Every instance admits a stable matching; can be computed in linear time.
- All stable matchings ~~are of the same size~~ need not be of same size.
- A stable matching can be half the size of another stable matching.

Question: How to compute largest size stable matching?

Variation #3: Lower Quotas

Variation #3: Lower Quotas

- Preferences are strict and can be incomplete.
- Some vertices must be matched – lower quota vertices.

Variation #3: Lower Quotas

- Preferences are strict and can be incomplete.
- Some vertices must be matched – lower quota vertices.

a_1:	b_1	b_2		b_1:	a_1	a_2
$\underline{a_2}$:	b_1			b_2:	a_1	

Variation #3: Lower Quotas

- Preferences are strict and can be incomplete.
- Some vertices must be matched – lower quota vertices.

a_1:	b_1	b_2		b_1:	a_1	a_2
$\underline{a_2}$:	b_1			b_2:	a_1	

- Does a stable and feasible matching exist? Not necessarily.

Variation #3: Lower Quotas

- Preferences are strict and can be incomplete.
- Some vertices must be matched – lower quota vertices.

---

a_1:	b_1	b_2		b_1:	a_1	a_2
$\underline{a_2}$:	b_1			b_2:	a_1	

- Does a stable and feasible matching exist? Not necessarily.
  - $M_1 = \{(a_1, b_1)\}$ Stable but not feasible.
  - $M_1 = \{(a_2, b_1), (a_1, b_2)\}$ Feasible but not stable.

Variation #3: Lower Quotas

- Preferences are strict and can be incomplete.
- Some vertices must be matched – lower quota vertices.

---

a_1:	b_1	b_2		b_1:	a_1	a_2
$\underline{a_2}$:	b_1			b_2:	a_1	

- Does a stable and feasible matching exist? Not necessarily.
  - $M_1 = \{(a_1, b_1)\}$ Stable but not feasible.
  - $M_1 = \{(a_2, b_1), (a_1, b_2)\}$ Feasible but not stable.

Known Fact:

Variation #3: Lower Quotas

- Preferences are strict and can be incomplete.
- Some vertices must be matched – lower quota vertices.

a_1:	b_1	b_2		b_1:	a_1	a_2
a_2:	b_1			b_2:	a_1	

- Does a stable and feasible matching exist? Not necessarily.
  - $M_1 = \{(a_1, b_1)\}$ Stable but not feasible.
  - $M_1 = \{(a_2, b_1), (a_1, b_2)\}$ Feasible but not stable.

Known Fact:

- In linear time we can check if an instance admits a feasible and stable matching.

Variation #3: Lower Quotas

- Preferences are strict and can be incomplete.
- Some vertices must be matched – lower quota vertices.

a_1:	b_1	b_2	b_1:	a_1	a_2
$\underline{a_2}$:	b_1		b_2:	a_1	

- Does a stable and feasible matching exist? Not necessarily.
  - $M_1 = \{(a_1, b_1)\}$ Stable but not feasible.
  - $M_1 = \{(a_2, b_1), (a_1, b_2)\}$ Feasible but not stable.

Known Fact:

- In linear time we can check if an instance admits a feasible and stable matching.

Question: How to compute optimal feasible matching?

Classical Model: Strict and Complete lists

Computing a stable matching

Gale and Shapley 1962

a_1:	b_1	b_2	b_1:	a_1	a_2
a_2:	b_1	b_2	b_2:	a_1	a_2

Computing a stable matching

Gale and Shapley 1962

a_1:	b_1	b_2	b_1:	a_1	a_2
a_2:	b_1	b_2	b_2:	a_1	a_2

Gale and Shapley Algo.

- Men propose.
- Women accept / reject.

Computing a stable matching

Gale and Shapley 1962

$\boldsymbol{a_1}$:	$\boldsymbol{b_1}$	b_2		$\boldsymbol{b_1}$:	$\boldsymbol{a_1}$	a_2
a_2:	b_1	b_2		b_2:	a_1	a_2

$\boxed{\text{Gale and Shapley Algo.}}$

- Men propose.
- Women accept / reject.

- $a_1 \rightarrow b_1$

Computing a stable matching

Gale and Shapley 1962

a_1:	$\boxed{b_1}$	b_2	b_1:	$\boxed{a_1}$	a_2
a_2:	b_1	b_2	b_2:	a_1	a_2

Gale and Shapley Algo.

- Men propose.
- Women accept / reject.

- $a_1 \rightarrow b_1$ accept.

Computing a stable matching

Gale and Shapley 1962

a_1:	$\boxed{b_1}$	b_2		b_1:	$\boxed{a_1}$	a_2
a_2:	b_1	b_2		b_2:	a_1	a_2

Gale and Shapley Algo.

- Men propose.
- Women accept / reject.

- $a_1 \rightarrow b_1$ accept.
- $a_2 \rightarrow b_1$

Computing a stable matching

Gale and Shapley 1962

a_1:	$\boxed{b_1}$	b_2	b_1:	$\boxed{a_1}$	a_2
a_2:	b_1	b_2	b_2:	a_1	a_2

Gale and Shapley Algo.

- Men propose.
- Women accept / reject.

- $a_1 \rightarrow b_1$ accept.
- $a_2 \rightarrow b_1$ reject.

Computing a stable matching

Gale and Shapley 1962

a_1:	$\boxed{b_1}$	b_2			b_1:	$\boxed{a_1}$	a_2
$\boldsymbol{a_2}$:	b_1	$\boldsymbol{b_2}$			$\boldsymbol{b_2}$:	a_1	$\boldsymbol{a_2}$

Gale and Shapley Algo.

- Men propose.
- Women accept / reject.

- $a_1 \to b_1$ accept.
- $a_2 \to b_1$ reject.
- $a_2 \to b_2$ accept.

Computing a stable matching

Gale and Shapley 1962

a_1:	$\boxed{b_1}$	b_2	b_1:	$\boxed{a_1}$	a_2
a_2:	b_1	$\boxed{b_2}$	b_2:	a_1	$\boxed{a_2}$

Gale and Shapley Algo.

- Men propose.
- Women accept / reject.

- $a_1 \rightarrow b_1$ accept.
- $a_2 \rightarrow b_1$ reject.
- $a_2 \rightarrow b_2$ accept.

Computing a stable matching

Gale and Shapley 1962

a_1:	$\boxed{b_1}$	b_2	b_1:	$\boxed{a_1}$	a_2
a_2:	b_1	$\boxed{b_2}$	b_2:	a_1	$\boxed{a_2}$

Gale and Shapley Algo.

- Men propose.
- Women accept / reject.

- $a_1 \to b_1$ accept.
- $a_2 \to b_1$ reject.
- $a_2 \to b_2$ accept.

$\boxed{M_s = \{(a_1, b_1), (a_2, b_2)\}.}$

- Order of proposals does not matter.
- The side which proposes does matter.

Models : Recap

Model	Details	Goal	
Classical setting	strict and complete list	Compute a stable matching	✓
Variation #1	strict and incomplete list	Compute a **larger optimal** matching	
Variation #2	strict and tied list	Compute a **largest stable** matching	
Variation #3	strict and incomplete list; lower quotas	Compute a **feasible optimal** matching	

Variation #2: Incomplete Lists and Ties

Variation #2: Incomplete Lists and Ties

Assume ties only on $\mathcal{B}$ side.

a_1:	b_1	b_2		b_1:	$(a_1$	$a_2)$
a_2:	b_1			b_2:	a_1	

Variation #2: Incomplete Lists and Ties

Assume ties only on $\mathcal{B}$ side.

a_1:	b_1	b_2		b_1:	$(a_1$	$a_2)$
a_2:	b_1			b_2:	a_1	

Recall:

- Multiple stable matchings of <u>different</u> sizes.

Variation #2: Incomplete Lists and Ties

Assume ties only on $\mathcal{B}$ side.

a_1:	b_1	b_2		b_1:	$(a_1$	$a_2)$
a_2:	b_1			b_2:	a_1	

Recall:

- Multiple stable matchings of different sizes.

  $$M_1 = \{(a_1, b_1)\} \qquad M_2 = \{(a_1, b_2), (a_2, b_1)\}$$

- Compute largest size stable matching

Variation #2: Incomplete Lists and Ties

Assume ties only on $\mathcal{B}$ side.

a_1:	b_1	b_2	b_1:	$(a_1$	$a_2)$
a_2:	b_1		b_2:	a_1	

Recall:

- Multiple stable matchings of different sizes.

  $M_1 = \{(a_1, b_1)\}$ $\qquad$ $M_2 = \{(a_1, b_2), (a_2, b_1)\}$

- Compute largest size stable matching

NP-hard even for restricted setting.

Variation #2: Incomplete Lists and Ties

Assume ties only on $\mathcal{B}$ side.

a_1:	b_1	b_2	b_1:	$(a_1$	$a_2)$
a_2:	b_1		b_2:	a_1	

Recall:

- Multiple stable matchings of different sizes.

  $M_1 = \{(a_1, b_1)\}$ $\quad$ $M_2 = \{(a_1, b_2), (a_2, b_1)\}$

- $\boxed{\text{Compute largest size stable matching}}$

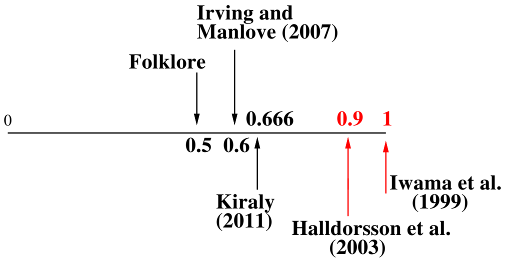
NP-hard even for restricted setting.

Naive method:

- Break ties arbitrarily.
- Run GS algo.

Variation #2: Incomplete Lists and Ties

Assume ties only on $\mathcal{B}$ side.



Variation #2: Incomplete Lists and Ties

Király 2011

Assume ties only on $\mathcal{B}$ side.

Király's Algorithm

- Break ties arbitrarily.
- Execute GS algo.
- Unmatched $\mathcal{A}$'s propose again with increased priority.

Variation #2: Incomplete Lists and Ties

Király 2011

Assume ties only on $\mathcal{B}$ side.

Király's Algorithm

- Break ties arbitrarily.
- Execute GS algo.
- Unmatched $\mathcal{A}$'s propose again with increased priority.
  - b uses increased priority for breaking ties.

Variation #2: Incomplete Lists and Ties

Király 2011

Assume ties only on $\mathcal{B}$ side.

Király's Algorithm

- Break ties arbitrarily.
- Execute GS algo.
- Unmatched $\mathcal{A}$'s propose again with increased priority.
  - b uses increased priority for breaking ties.
  - Stability is not violated.

Variation #2: Incomplete Lists and Ties

Király 2011

a_1:	b_1	b_2	b_1:	$(a_1$	$a_2)$
a_2:	b_1		b_2:	a_1	

Variation #2: Incomplete Lists and Ties

Király 2011

a_1:	b_1	b_2	b_1:	a_1	a_2
a_2:	b_1		b_2:	a_1	

$\boxed{\text{Király's Algo.}}$

- Break ties arbitrarily.
- Run GS algo.
- Unmatched $\mathcal{A}$s propose with increased priority.

Variation #2: Incomplete Lists and Ties

Király 2011

a_1:	$\boldsymbol{b_1}$	b_2		b_1:	$\boldsymbol{a_1}$	a_2
a_2:	b_1			b_2:	a_1	

Király's Algo.

- Break ties arbitrarily.
- Run GS algo.
- Unmatched $\mathcal{A}$s propose with increased priority.

- $a_1 \rightarrow b_1$

Variation #2: Incomplete Lists and Ties

Király 2011

a_1:	$\boxed{b_1}$	b_2		b_1:	$\boxed{a_1}$	a_2
a_2:	b_1			b_2:	a_1	

Király's Algo.

- Break ties arbitrarily.
- Run GS algo.
- Unmatched $\mathcal{A}$s propose with increased priority.

- $a_1 \rightarrow b_1$ accept.

Variation #2: Incomplete Lists and Ties

Király 2011

a_1:	$\boxed{b_1}$	b_2	b_1:	$\boxed{a_1}$	$\boldsymbol{a_2}$
$\boldsymbol{a_2}$:	$\boldsymbol{b_1}$		b_2:	a_1	

Király's Algo.

- Break ties arbitrarily.
- Run GS algo.
- Unmatched $\mathcal{A}$s propose with increased priority.

- $a_1 \rightarrow b_1$ accept.
- $a_2 \rightarrow b_1$

Variation #2: Incomplete Lists and Ties

Király 2011

a_1:	$\boxed{b_1}$	b_2		b_1:	$\boxed{a_1}$	a_2
a_2:	b_1			b_2:	a_1	

Király's Algo.

- Break ties arbitrarily.
- Run GS algo.
- Unmatched $\mathcal{A}$s propose with increased priority.

- $a_1 \to b_1$ accept.
- $a_2 \to b_1$ reject.

Variation #2: Incomplete Lists and Ties

Király 2011

a_1:	$\boxed{b_1}$	b_2		b_1:	$\boxed{a_1}$	$\boldsymbol{a_2}$
$\boldsymbol{a_2}$:	$\boldsymbol{b_1}$			b_2:	a_1	

Király's Algo.

- Break ties arbitrarily.
- Run GS algo.
- Unmatched $\mathcal{A}$s propose with increased priority.

- $a_1 \to b_1$ accept.
- $a_2 \to b_1$ reject.
- $a_2^* \to b_1$ accept; recall ties originally.

Variation #2: Incomplete Lists and Ties

Király 2011

a_1:	b_1	b_2	b_1:	a_1	$\boxed{a_2}$
a_2:	$\boxed{b_1}$		b_2:	a_1	

Király's Algo.

- Break ties arbitrarily.
- Run GS algo.
- Unmatched $\mathcal{A}$s propose with increased priority.

- $a_1 \rightarrow b_1$ accept.
- $a_2 \rightarrow b_1$ reject.
- $a_2^* \rightarrow b_1$ accept; recall ties originally.

Variation #2: Incomplete Lists and Ties

Király 2011

$\boldsymbol{a_1}$:	b_1	$\boldsymbol{b_2}$		b_1:	a_1	$\boxed{a_2}$
a_2:	$\boxed{b_1}$			$\boldsymbol{b_2}$:	$\boldsymbol{a_1}$	

Király's Algo.

- Break ties arbitrarily.
- Run GS algo.
- Unmatched $\mathcal{A}$s propose with increased priority.

- $a_1 \rightarrow b_1$ accept.
- $a_2 \rightarrow b_1$ reject.
- $a_2^* \rightarrow b_1$ accept; recall ties originally.
- $a_1 \rightarrow b_2$ accept.

Variation #2: Incomplete Lists and Ties

Király 2011

a_1:	b_1	$\boxed{b_2}$	b_1:	a_1	$\boxed{a_2}$
a_2:	$\boxed{b_1}$		b_2:	$\boxed{a_1}$	

Király's Algo.

- Break ties arbitrarily.
- Run GS algo.
- Unmatched $\mathcal{A}$s propose with increased priority.

- $a_1 \to b_1$ accept.
- $a_2 \to b_1$ reject.
- $a_2^* \to b_1$ accept; recall ties originally.
- $a_1 \to b_2$ accept.

$\boxed{M = \{(a_1, b_2), (a_2, b_1)\}.}$

Variation #2: Incomplete Lists and Ties

Király 2011

a_1:	b_1	$\boxed{b_2}$		b_1:	a_1	$\boxed{a_2}$
a_2:	$\boxed{b_1}$			b_2:	$\boxed{a_1}$	

Király's Algo.

- Break ties arbitrarily.
- Run GS algo.
- Unmatched $\mathcal{A}$s propose with increased priority.

- $a_1 \rightarrow b_1$ accept.
- $a_2 \rightarrow b_1$ reject.
- $a_2^* \rightarrow b_1$ accept; recall ties originally.
- $a_1 \rightarrow b_2$ accept.

$\boxed{M = \{(a_1, b_2), (a_2, b_1)\}.}$

Goal: Argue about the size of the matching.

Variation #2: Incomplete Lists and Ties

Király 2011

a_1:	b_1	$\boxed{b_2}$			b_1:	a_1	$\boxed{a_2}$
a_2:	$\boxed{b_1}$				b_2:	$\boxed{a_1}$	

Király's Algo.

- Break ties arbitrarily.
- Run GS algo.
- Unmatched $\mathcal{A}$s propose with increased priority.

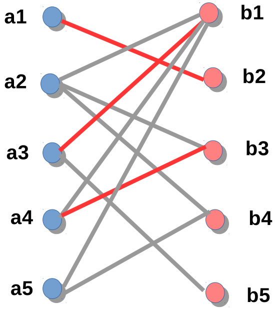
- $a_1 \to b_1$ accept.
- $a_2 \to b_1$ reject.
- $a_2^* \to b_1$ accept; recall ties originally.
- $a_1 \to b_2$ accept.

$\boxed{M = \{(a_1, b_2), (a_2, b_1)\}.}$

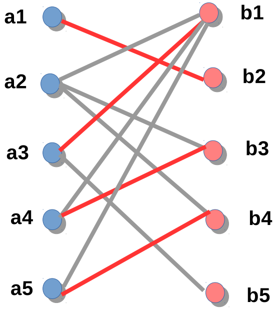
Goal: Argue about the size of the matching.

- Show no short aug. paths.

Matchings and aug. paths: a detour

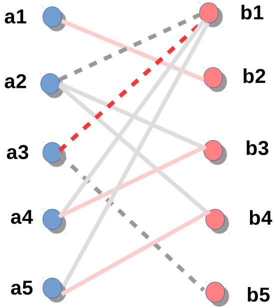


Matchings and aug. paths: a detour



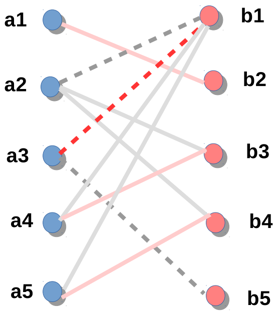
- Is this the largest sized matching?

Matchings and aug. paths: a detour



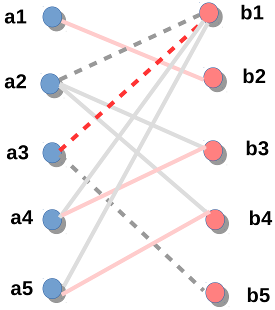
- Is this the largest sized matching?
- a_2, b_1, a_3, b_5 – alternating path with both end points free.

Matchings and aug. paths: a detour



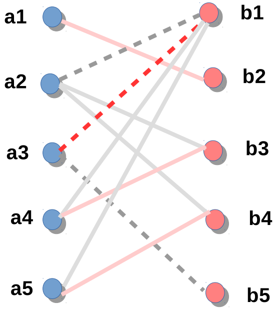
- Is this the largest sized matching?
- a_2, b_1, a_3, b_5 – alternating path with both end points free.
- Aug. paths: odd number of edges (1, 3, 5, ..., 2k+1)

Matchings and aug. paths: a detour



- Is this the largest sized matching?
- a_2, b_1, a_3, b_5 – alternating path with both end points free.
- Aug. paths: odd number of edges
  (1, 3, 5, ..., 2k+1)
- No one length aug. path $\rightarrow$ maximal

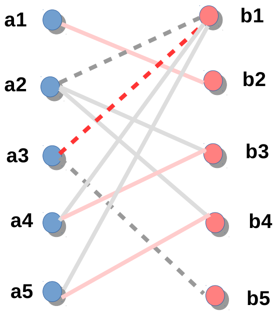
Matchings and aug. paths: a detour



- Is this the largest sized matching?
- a_2, b_1, a_3, b_5 – alternating path with both end points free.
- Aug. paths: odd number of edges (1, 3, 5, ..., 2k+1)
- No one length aug. path $\rightarrow$ maximal
- No short aug. path, closer to max. matching.

- Matching M' without 1 and 3 length aug. paths.

Matchings and aug. paths: a detour



- Is this the largest sized matching?
- a_2, b_1, a_3, b_5 – alternating path with both end points free.
- Aug. paths: odd number of edges (1, 3, 5, ..., 2k+1)
- No one length aug. path $\rightarrow$ maximal
- No short aug. path, closer to max. matching.

- Matching M' without 1 and 3 length aug. paths.
- $|M'| \geq \frac{2}{3}|M^*|$.

$\boxed{\textbf{Back to Király's algorithm}}$

Recall Király's algorithm

- Break ties arbitrarily.
- Execute GS algo.
- Unmatched $\mathcal{A}$'s propose again with increased priority.

Recall Király's algorithm

- Break ties arbitrarily.
- Execute GS algo.
- Unmatched $\mathcal{A}$'s propose again with increased priority.
  - b uses increased priority for breaking ties.

Recall Király's algorithm

- Break ties arbitrarily.
- Execute GS algo.
- Unmatched $\mathcal{A}$'s propose again with increased priority.
  - b uses increased priority for breaking ties.
  - Stability is not violated.

Recall Király's algorithm

- Break ties arbitrarily.
- Execute GS algo.
- Unmatched $\mathcal{A}$'s propose again with increased priority.
    - b uses increased priority for breaking ties.
    - Stability is not violated.
- ~~Need to argue about the size of the output.~~

Recall Király's algorithm

- Break ties arbitrarily.
- Execute GS algo.
- Unmatched $\mathcal{A}$'s propose again with increased priority.
  - b uses increased priority for breaking ties.
  - Stability is not violated.
- ~~Need to argue about the size of the output.~~
- Show that there are no short (1 and 3 length) aug. paths.

Recall Király's algorithm

- Break ties arbitrarily.
- Execute GS algo.
- Unmatched $\mathcal{A}$'s propose again with increased priority.
  - b uses increased priority for breaking ties.
  - Stability is not violated.
- ~~Need to argue about the size of the output.~~
- Show that there are no short (1 and 3 length) aug. paths.

Some observations:

Recall Király's algorithm

- Break ties arbitrarily.
- Execute GS algo.
- Unmatched $\mathcal{A}$'s propose again with increased priority.
  - b uses increased priority for breaking ties.
  - Stability is not violated.
- ~~Need to argue about the size of the output.~~
- Show that there are no short (1 and 3 length) aug. paths.

Some observations:

- If a woman b is unmatched at the end of algo., she never got a proposal.

Recall Király's algorithm

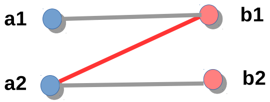
- Break ties arbitrarily.
- Execute GS algo.
- Unmatched $\mathcal{A}$'s propose again with increased priority.
  - b uses increased priority for breaking ties.
  - Stability is not violated.
- ~~Need to argue about the size of the output.~~
- Show that there are no short (1 and 3 length) aug. paths.

Some observations:

- If a woman b is unmatched at the end of algo., she never got a proposal.
- If a man a is unmatched at the end of algo., he got increased priority.

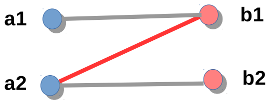
Output of Király's algorithm

Suppose there exists a 3 length aug. path w.r.t. M_{algo}.



Output of Király's algorithm

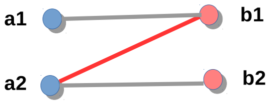
Suppose there exists a 3 length aug. path w.r.t. M_{algo}.



- a_2 never proposed to b_2 ($\because$ b_2 is unmatched after algo) $\rightarrow$ a_2 did not get increased priority.

Output of Király's algorithm

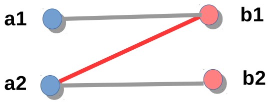
Suppose there exists a 3 length aug. path w.r.t. M_{algo}.



- a_2 never proposed to b_2 ($\because$ b_2 is unmatched after algo)
  $\rightarrow$ a_2 did not get increased priority.
  $\rightarrow$ $\boxed{a_2 \text{ strictly prefers } b_1 \text{ over } b_2.}$

Output of Király's algorithm

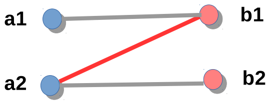
Suppose there exists a 3 length aug. path w.r.t. M_{algo}.



- a_2 never proposed to b_2 ($\because$ b_2 is unmatched after algo)
  $\rightarrow$ a_2 did not get increased priority.
  $\rightarrow$ $\boxed{a_2 \text{ strictly prefers } b_1 \text{ over } b_2.}$
- a_1 is unmatched at the end of algo.
  $\rightarrow$ a_1 must have got high priority.

Output of Király's algorithm

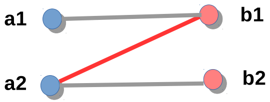
Suppose there exists a 3 length aug. path w.r.t. M_{algo}.



- a_2 never proposed to b_2 ($\because$ b_2 is unmatched after algo)
  $\rightarrow$ a_2 did not get increased priority.
  $\rightarrow$ $\boxed{a_2 \text{ strictly prefers } b_1 \text{ over } b_2.}$
- a_1 is unmatched at the end of algo.
  $\rightarrow$ a_1 must have got high priority.
  $\rightarrow$ $\boxed{b_1 \text{ strictly prefers } a_2 \text{ over } a_1.}$

Output of Király's algorithm

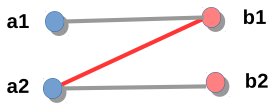
Suppose there exists a 3 length aug. path w.r.t. M_{algo}.



- a_2 never proposed to b_2 ($\because$ b_2 is unmatched after algo)
  $\rightarrow$ a_2 did not get increased priority.
  $\rightarrow$ $\boxed{a_2 \text{ strictly prefers } b_1 \text{ over } b_2.}$
- a_1 is unmatched at the end of algo.
  $\rightarrow$ a_1 must have got high priority.
  $\rightarrow$ $\boxed{b_1 \text{ strictly prefers } a_2 \text{ over } a_1.}$
- (a_2, b_1) is a blocking pair w.r.t. M^*.

Output of Király's algorithm

Suppose there exists a 3 length aug. path w.r.t. M_{algo}.

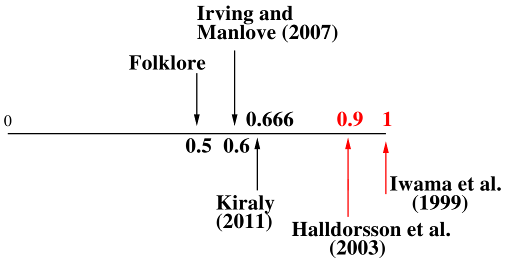


- a_2 never proposed to b_2 ($\because$ b_2 is unmatched after algo)
  $\rightarrow$ a_2 did not get increased priority.
  $\rightarrow$ $\boxed{a_2 \text{ strictly prefers } b_1 \text{ over } b_2.}$
- a_1 is unmatched at the end of algo.
  $\rightarrow$ a_1 must have got high priority.
  $\rightarrow$ $\boxed{b_1 \text{ strictly prefers } a_2 \text{ over } a_1.}$
- (a_2, b_1) is a blocking pair w.r.t. M^*.
- contradicts stability of M^*.

There are no 3 length aug. paths w.r.t. M_{algo}.
Thus, $|M_{algo}| \geq \frac{2}{3}|M^*|$.

Variation #2: Incomplete Lists and Ties

Assume ties only on $\mathcal{B}$ side.



Main takeaways

- A simple extension of GS algo.
- Extension to capacitated case (hospital residents).
- Extension to case of ties on both sides.

Models : Recap

Model	Details	Goal	
Classical setting	strict and complete list	Compute a stable matching	✓
Variation #1	strict and incomplete list	Compute a **larger optimal** matching	
Variation #2	strict and tied list	Compute a **largest stable** matching	✓
Variation #3	strict and incomplete list; lower quotas	Compute a **feasible optimal** matching	

Variation #1 & Variation #3

Popularity – an alternative to stability

Gärdenfors 1975

Compare two matchings by votes of participants.

Popularity – an alternative to stability

Gärdenfors 1975

Compare two matchings by votes of participants.

a_1:	b_1	b_2	b_1:	a_1	$\boxed{a_2}$
a_2:	$\boxed{b_1}$		b_2:	a_1	

- $M_s = \{(a_1, b_1)\}$
- $M = \{(a_2, b_1)\}$

Popularity – an alternative to stability

Gärdenfors 1975

Compare two matchings by votes of participants.

a_1: b_1 b_2
a_2: $\boxed{b_1}$

b_1: a_1 $\boxed{a_2}$
b_2: a_1

- $M_s = \{(a_1, b_1)\}$
- $M = \{(a_2, b_1)\}$

	M_s	M
a_1	✓	
a_2		✓
b_1	✓	

Popularity – an alternative to stability

Gärdenfors 1975

Compare two matchings by votes of participants.

a_1:	b_1	b_2		b_1:	a_1	$\boxed{a_2}$
a_2:	$\boxed{b_1}$			b_2:	a_1	

- $M_s = \{(a_1, b_1)\}$
- $M = \{(a_2, b_1)\}$

	M_s	M
a_1	✓	
a_2		✓
b_1	✓	

- M_s beats M w.r.t. popularity.
- Popular Matching:
  One which cannot be beaten!

Popularity – an alternative to stability

Gärdenfors 1975

Compare two matchings by votes of participants.

a_1: b_1 b_2
a_2: $\boxed{b_1}$

b_1: a_1 $\boxed{a_2}$
b_2: a_1

- $M_s = \{(a_1, b_1)\}$
- $M = \{(a_2, b_1)\}$

	M_s	M
a_1	✓	
a_2		✓
b_1	✓	

- M_s beats M w.r.t. popularity.
- Popular Matching:
  One which cannot be beaten!
- Q: Does a popular matching exist?

Popularity – an alternative to stability

Gärdenfors 1975

Compare two matchings by votes of participants.

a_1:	b_1	$\boxed{b_2}$	b_1:	a_1	$\boxed{a_2}$
a_2:	$\boxed{b_1}$		b_2:	$\boxed{a_1}$	

- $M_s = \{(a_1, b_1)\}$
- $M' = \{(a_1, b_2), (a_2, b_1)\}$

Popularity – an alternative to stability

Gärdenfors 1975

Compare two matchings by votes of participants.

a_1: b_1 $\boxed{b_2}$
a_2: $\boxed{b_1}$

b_1: a_1 $\boxed{a_2}$
b_2: $\boxed{a_1}$

- $M_s = \{(a_1, b_1)\}$
- $M' = \{(a_1, b_2), (a_2, b_1)\}$

	M_s	M
a_1	✓	
a_2		✓
b_1	✓	
b_2		✓

Popularity – an alternative to stability

Gärdenfors 1975

Compare two matchings by votes of participants.

a_1: b_1 $\boxed{b_2}$
a_2: $\boxed{b_1}$

b_1: a_1 $\boxed{a_2}$
b_2: $\boxed{a_1}$

- $M_s = \{(a_1, b_1)\}$
- $M' = \{(a_1, b_2), (a_2, b_1)\}$

	M_s	M
a_1	✓	
a_2		✓
b_1	✓	
b_2		✓

- Neither one beats each other.

Popularity – an alternative to stability

Gärdenfors 1975

Compare two matchings by votes of participants.

a_1: b_1 $\boxed{b_2}$
a_2: $\boxed{b_1}$

b_1: a_1 $\boxed{a_2}$
b_2: $\boxed{a_1}$

- $M_s = \{(a_1, b_1)\}$
- $M' = \{(a_1, b_2), (a_2, b_1)\}$

	M_s	M
a_1	✓	
a_2		✓
b_1	✓	
b_2		✓

- Neither one beats each other.
- Does not prove popularity.

Popularity – an alternative to stability

Gärdenfors 1975

Compare two matchings by votes of participants.

a_1: b_1 $\boxed{b_2}$

a_2: $\boxed{b_1}$

b_1: a_1 $\boxed{a_2}$

b_2: $\boxed{a_1}$

- $M_s = \{(a_1, b_1)\}$
- $M' = \{(a_1, b_2), (a_2, b_1)\}$

	M_s	M
a_1	✓	
a_2		✓
b_1	✓	
b_2		✓

- Neither one beats each other.
- Does not prove popularity.
- Q: Does a popular matching exist?

Popularity – an alternative to stability

Gärdenfors 1975

Compare two matchings by votes of participants.

a_1: b_1 $\boxed{b_2}$
a_2: $\boxed{b_1}$

b_1: a_1 $\boxed{a_2}$
b_2: $\boxed{a_1}$

- $M_s = \{(a_1, b_1)\}$
- $M' = \{(a_1, b_2), (a_2, b_1)\}$

	M_s	M
a_1	✓	
a_2		✓
b_1	✓	
b_2		✓

- Neither one beats each other.
- Does not prove popularity.
- Q: Does a popular matching exist?
  Yes! A stable matching is popular.

Variation #1: Incomplete Lists

- Preferences are strict and can be incomplete.

a_1:	b_1	b_2		b_1:	a_1	a_2
a_2:	b_1			b_2:	a_1	

- Does a stable matching exist? Yes! $M = \{(a_1, b_1)\}$.

Question: Are there larger optimal matchings?

Variation #1: Incomplete Lists

- Preferences are strict and can be incomplete.

a_1:	b_1	b_2	b_1:	a_1	a_2
a_2:	b_1		b_2:	a_1	

- Does a stable matching exist? Yes! $M = \{(a_1, b_1)\}$.

Question: Are there larger optimal matchings?

Yes! $M' = \{(a_1, b_2), (a_2, b_1)\}$ is popular.

Variation #1: Incomplete Lists

Kavitha 2012

a_1:	b_1	b_2		b_1:	a_1	a_2
a_2:	b_1			b_2:	a_1	

Goal: Compute Largest sized Popular matching.

Variation #1: Incomplete Lists

Kavitha 2012

a_1:	b_1	b_2	b_1:	a_1	a_2
a_2:	b_1		b_2:	a_1	

Goal: Compute Largest sized Popular matching.

- Run GS algo. Unmatched $\mathcal{A}$'s propose with increased priority.

Variation #1: Incomplete Lists

Kavitha 2012

a_1:	b_1	b_2	b_1:	a_1	a_2
a_2:	b_1		b_2:	a_1	

Goal: Compute Largest sized Popular matching.

- Run GS algo. Unmatched $\mathcal{A}$'s propose with increased priority.
- Stability may be violated.

Variation #1: Incomplete Lists

Kavitha 2012

a_1:	b_1	b_2	b_1:	a_1	a_2
a_2:	b_1		b_2:	a_1	

Goal: Compute Largest sized Popular matching.

- Run GS algo. Unmatched $\mathcal{A}$'s propose with increased priority.
- Stability may be violated.
- Guarantees on output :
  - M_{algo} is max. sized popular.
  - $|M_{algo}| \geq |M_s|$ and $|M_{algo}| \geq \frac{2}{3}|M^*|$.
  - Linear time algo.

Variation #1: Incomplete Lists

Kavitha 2012

a_1:	b_1	b_2		b_1:	a_1	a_2
a_2:	b_1			b_2:	a_1	

Goal: Compute Largest sized Popular matching.

- Run GS algo. Unmatched $\mathcal{A}$'s propose with increased priority.
- Stability may be violated.
- Guarantees on output :
  - M_{algo} is max. sized popular.
  - $|M_{algo}| \geq |M_s|$ and $|M_{algo}| \geq \frac{2}{3}|M^*|$.
  - Linear time algo.

Goal: Compute a max. card. matching that is *popular*.

Variation #1: Incomplete Lists

Kavitha 2012

a_1:	b_1	b_2			b_1:	a_1	a_2
a_2:	b_1				b_2:	a_1	

Goal: Compute Largest sized Popular matching.

- Run GS algo. Unmatched $\mathcal{A}$'s propose with increased priority.
- Stability may be violated.
- Guarantees on output :
  - M_{algo} is max. sized popular.
  - $|M_{algo}| \geq |M_s|$ and $|M_{algo}| \geq \frac{2}{3}|M^*|$.
  - Linear time algo.

Goal: Compute a max. card. matching that is *popular*.

- Run GS algo. Unmatched $\mathcal{A}$'s propose with increased priority n times.

Variation #1: Incomplete Lists

Kavitha 2012

a_1:	b_1	b_2		b_1:	a_1	a_2
a_2:	b_1			b_2:	a_1	

Goal: Compute Largest sized Popular matching.

- Run GS algo. Unmatched $\mathcal{A}$'s propose with increased priority.
- Stability may be violated.
- Guarantees on output :
  - M_{algo} is max. sized popular.
  - $|M_{algo}| \geq |M_s|$ and $|M_{algo}| \geq \frac{2}{3}|M^*|$.
  - Linear time algo.

Goal: Compute a max. card. matching that is *popular*.

- Run GS algo. Unmatched $\mathcal{A}$'s propose with increased priority n times.
- Stability may be violated.

Variation #1: Incomplete Lists

Kavitha 2012

a_1:	b_1	b_2		b_1:	a_1	a_2
a_2:	b_1			b_2:	a_1	

Goal: Compute Largest sized Popular matching.

- Run GS algo. Unmatched $\mathcal{A}$'s propose with increased priority.
- Stability may be violated.
- Guarantees on output :
  - M_{algo} is max. sized popular.
  - $|M_{algo}| \geq |M_s|$ and $|M_{algo}| \geq \frac{2}{3}|M^*|$.
  - Linear time algo.

Goal: Compute a max. card. matching that is *popular*.

- Run GS algo. Unmatched $\mathcal{A}$'s propose with increased priority n times.
- Stability may be violated.
- Guarantees on output :
  - M_{algo} is max. cardinality matching.
  - M_{algo} is popular amongst max. card. matchings.
  - Running time: $O(nm)$.

Variation #3: Lower Quotas

- Preferences are strict and can be incomplete.
- Some vertices must be matched – lower quota vertices.

a_1:	b_1	b_2		b_1:	a_1	a_2
$\underline{a_2}$:	b_1			b_2:	a_1	

- Does a stable and feasible matching exist? Not necessarily.

Question: How to compute optimal feasible matching?

Variation #3: Lower Quotas

- Preferences are strict and can be incomplete.
- Some vertices must be matched – lower quota vertices.

a_1:	b_1	b_2		b_1:	a_1	a_2
$\underline{a_2}$:	b_1			b_2:	a_1	

- Does a stable and feasible matching exist? Not necessarily.

Question: How to compute optimal feasible matching?

Yes! $M' = \{(a_2, b_1), (a_1, b_2)\}$ Feasible, not stable, but popular.

Variation #3: Lower Quotas

N., Nimbhorkar 2017

- Preferences are strict and can be incomplete.
- Some vertices must be matched – lower quota vertices.

a_1:	b_1	b_2		b_1:	a_1	a_2
$\underline{a_2}$:	b_1			b_2:	a_1	

Goal: Compute a feasible matching that is popular.

Variation #3: Lower Quotas

N., Nimbhorkar 2017

- Preferences are strict and can be incomplete.
- Some vertices must be matched – lower quota vertices.

---

a_1:	b_1	b_2		b_1:	a_1	a_2
$\underline{a_2}$:	b_1			b_2:	a_1	

Goal: Compute a feasible matching that is popular.

- Run GS algo. Unmatched $\mathcal{A}$'s propose with increased priority. Deficient $\mathcal{A}$'s propose as long as they are deficient.

Variation #3: Lower Quotas

N., Nimbhorkar 2017

- Preferences are strict and can be incomplete.
- Some vertices must be matched – lower quota vertices.

---

a_1:	b_1	b_2		b_1:	a_1	a_2
$\underline{a_2}$:	b_1			b_2:	a_1	

Goal: Compute a feasible matching that is popular.

- Run GS algo. Unmatched $\mathcal{A}$'s propose with increased priority. Deficient $\mathcal{A}$'s propose as long as they are deficient.
- Stability will be violated.

Variation #3: Lower Quotas

N., Nimbhorkar 2017

- Preferences are strict and can be incomplete.
- Some vertices must be matched – lower quota vertices.

---

a_1:	b_1	b_2	b_1:	a_1	a_2
$\underline{a_2}$:	b_1		b_2:	a_1	

Goal: Compute a feasible matching that is popular.

- Run GS algo. Unmatched $\mathcal{A}$'s propose with increased priority. Deficient $\mathcal{A}$'s propose as long as they are deficient.
- Stability will be violated.
- Guarantees on output :
  - M_{algo} is feasible.
  - M_{algo} is max. sized popular. amongst feasible matchings.
  - Running time: $O(nm)$.

Models : Summary

Model	Details	Goal	
Classical setting	strict and complete list	Compute a stable matching	✓
Variation #1	strict and incomplete list	Compute a **larger optimal** matching	✓
Variation #2	strict and tied list	Compute a **largest stable** matching	✓
Variation #3	strict and incomplete list; lower quotas	Compute a **feasible optimal** matching	✓

To summarize..

A simple extension of GS algo.

- Each case requires different proof techniques and several non-trivial details.
- All algorithms can be written as a reduction to a suitable SM instance.
- Works in the presence of capacities (upper quotas).

To summarize..

A simple extension of GS algo.

- Each case requires different proof techniques and several non-trivial details.
- All algorithms can be written as a reduction to a suitable SM instance.
- Works in the presence of capacities (upper quotas).

To summarize..

A simple extension of GS algo.

- Each case requires different proof techniques and several non-trivial details.
- All algorithms can be written as a reduction to a suitable SM instance.
- Works in the presence of capacities (upper quotas).

Thank You!