MECHANISM DESIGN

JOHN P DICKERSON AND MARINA KNITTEL

Lecture #7 – 02/14/2022

CMSC498T
Mondays & Wednesdays
2:00pm – 3:15pm

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

THIS CLASS:
MATCHING & MAYBE THE NRMP

OVERVIEW OF THIS LECTURE

Stable marriage problem

- Bipartite, one vertex to one vertex

Stable roommates problem

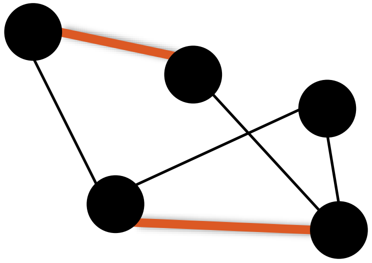
- Not bipartite, one vertex to one vertex

Hospitals/Residents problem

- Bipartite, one vertex to many vertices

MATCHING WITHOUT INCENTIVES

Given a graph G = (V, E), a matching is any set of pairwise non-adjacent edges



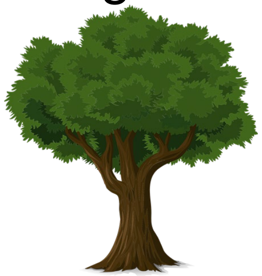
- No two edges share the same vertex
- Classical combinatorial optimization problem

Bipartite matching:

- Bipartite graph G = (U, V, E)
- Max cardinality/weight matching found easily – O(VE) and better
  - E.g., through network flow, Hungarian algorithm, etc

Matching in general graphs:

- Also PTIME via Edmond's algorithm – $O(V^2E)$ and better



STABLE MATCHING PROBLEM

Thanks Prof. Xanda Schofield for the example!

Complete bipartite graph with equal sides:

- *n* horses and *n* jockeys

Each horse has a strict, complete preference ordering over jockeys, and vice versa

Want: a stable matching

Stable matching: No unmatched horse and jockey both prefer each other to their current matches

EXAMPLE PREFERENCE PROFILES

 >  > 

Alice			
Bob			
Eve			

Donkey			
Spirit			
Swiftwind			

EXAMPLE PREFERENCE PROFILES

 >  > 

Alice	Donkey	Spirit	Swiftwind
Bob	Spirit	Donkey	Swiftwind
Eve	Donkey	Spirit	Swiftwind

Donkey	Bob	Alice	Eve
Spirit	Alice	Bob	Eve
Swiftwind	Alice	Bob	Eve

EXAMPLE MATCHING #1

Alice	Donkey	Spirit	Swiftwind
Bob	Spirit	Donkey	Swiftwind
Eve	Donkey	Spirit	Swiftwind

Donkey	Bob	Alice	Eve
Spirit	Alice	Bob	Eve
Swiftwind	Alice	Bob	Eve

Is this a **stable matching?**

EXAMPLE MATCHING #1

Alice	Donkey	Spirit	Swiftwind
Bob	Spirit	Donkey	Swiftwind
Eve	Donkey	Spirit	Swiftwind

Donkey	Bob	Alice	Eve
Spirit	Alice	Bob	Eve
Swiftwind	Alice	Bob	Eve

No.

Alice and Spirit form a **blocking pair.**

EXAMPLE MATCHING #2

Alice	Donkey	Spirit	Swiftwind
Bob	Spirit	Donkey	Swiftwind
Eve	Donkey	Spirit	Swiftwind

Donkey	Bob	Alice	Eve
Spirit	Alice	Bob	Eve
Swiftwind	Alice	Bob	Eve

What about this matching?

EXAMPLE MATCHING #2

Alice	Donkey	Spirit	Swiftwind
Bob	Spirit	Donkey	Swiftwind
Eve	Donkey	Spirit	Swiftwind

Donkey	Bob	Alice	Eve
Spirit	Alice	Bob	Eve
Swiftwind	Alice	Bob	Eve

Yes!

(Swiftwind and Eve are unhappy, but helpless.)

THROWBACK MONDAY: INT. LINEAR PROGRAMS

Can we formulate this as a linear program?

Spoiler: Yes we can

Another spoiler: You're going to do it!

What are our variables?

x_{hj} **for each horse** $h \in \{1, \dots, n\}$ **and jockey** $j \in \{1, \dots, n\}$

How are they bounded?

$x_{hj} \in \{0, 1\}$, **indicating if the horse and jockey are matched**

How do we ensure everyone is only matched once?

$\sum_{j \in \{1,\dots,n\}} x_{hj} \leq 1$ **for all** $h \in \{1, \dots, n\}$ **(covers horses)**

$\sum_{h \in \{1,\dots,n\}} x_{hj} \leq 1$ **for all** $j \in \{1, \dots, n\}$ **(covers jockeys)**

How do we ensure stability?

$\underbrace{\sum_{j' >_h j} x_{hj'}}_{h \text{ prefers its match to } j \text{ OR}} + \underbrace{\sum_{h' >_j h} x_{h'j}}_{j \text{ prefers its match to } h \text{ OR}} + \underbrace{x_{hj}}_{j \text{ is matched to } h} \geq 1$ **for all** $h, j \in \{1, \dots, n\}$

THROWBACK MONDAY: INT. LINEAR PROGRAMS

Optimize: ***Nothing***

$$\sum_{j\in\{1,\ldots,n\}} x_{hj} \leq 1 \text{ for all } h\in\{1,\ldots,n\}$$

$$\sum_{h\in\{1,\ldots,n\}} x_{hj} \leq 1 \text{ for all } j\in\{1,\ldots,n\}$$

$$\sum_{j'>_h j} x_{hj'} + \sum_{h'>_j h} x_{h'j} + x_{hj} \geq 1 \text{ for all } h,j\in\{1,\ldots,n\}$$

$$x_{hj} \in \{0,1\} \text{ for all } h,j\in\{1,\ldots,n\}$$

What does this give us?

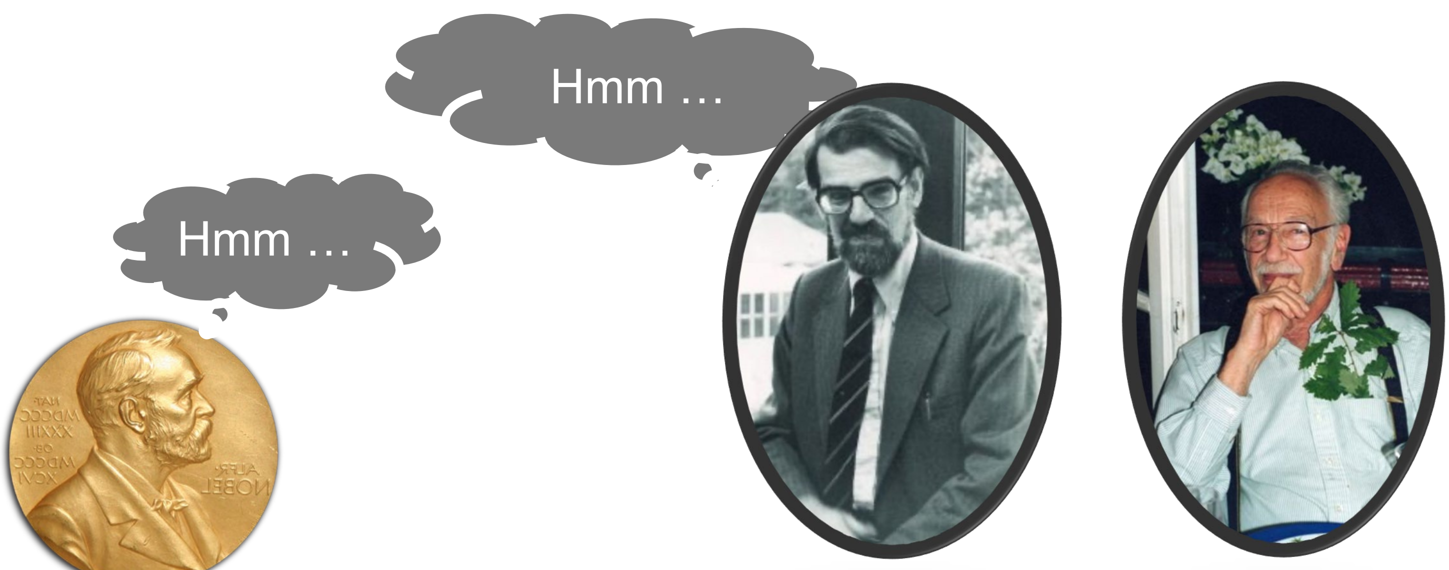
- If there is a stable matching, this finds one
- This might take exponential time!
- Open question: Can there exist no stable matching?

SOME QUESTIONS

Does a stable solution to the marriage problem always exist?

Can we compute such a solution efficiently?

Can we compute the best stable solution efficiently?



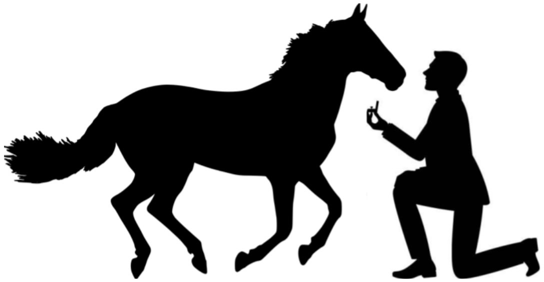
Lloyd Shapley *David Gale*

GALE-SHAPLEY [1962]

Idea: men propose to women

GALE-SHAPLEY [1962]

Idea: jockeys "propose" to horses



1. **Everyone is unmatched**
2. **While some jockey j is unmatched:**
   - $h \coloneqq j$'s most-preferred horse to whom they have not proposed yet
   - If h is also unmatched:
     - h and j are engaged
   - Else if h prefers j to their current match j'
     - h and j are engaged, j' is unmatched
   - Else: h rejects j
3. **Return matched pairs**

RUNNING GS

 >  > 

Alice	Donkey	Spirit	Swiftwind
Bob	Spirit	Donkey	Swiftwind
Eve	Donkey	Spirit	Swiftwind

Donkey	Bob	Alice	Eve
Spirit	Alice	Bob	Eve
Swiftwind	Alice	Bob	Eve

RUNNING GS

 >  > 

Alice	Donkey	Spirit	Swiftwind
Bob	Spirit	Donkey	Swiftwind
Eve	Donkey	Spirit	Swiftwind

Donkey	Bob	Alice	Eve
Spirit	Alice	Bob	Eve
Swiftwind	Alice	Bob	Eve

RUNNING GS

 >  > 

Alice	Donkey	Spirit	Swiftwind
Bob	Spirit	Donkey	Swiftwind
Eve	Donkey	Spirit	Swiftwind

Donkey	Bob	Alice	Eve
Spirit	Alice	Bob	Eve
Swiftwind	Alice	Bob	Eve

RUNNING GS

 >  > 

Alice	Donkey	Spirit	Swiftwind
Bob	Spirit	Donkey	Swiftwind
Eve	Donkey	Spirit	Swiftwind

Donkey	Bob	Alice	Eve
Spirit	Alice	Bob	Eve
Swiftwind	Alice	Bob	Eve

RUNNING GS

 >  > 

Alice	Donkey	Spirit	Swiftwind
Bob	Spirit	Donkey	Swiftwind
Eve	Donkey	Spirit	Swiftwind

Donkey	Bob	Alice	Eve
Spirit	Alice	Bob	Eve
Swiftwind	Alice	Bob	Eve

RUNNING GS

 >  > 

Alice	Donkey	Spirit	Swiftwind
Bob	Spirit	Donkey	Swiftwind
Eve	Donkey	Spirit	Swiftwind

Donkey	Bob	Alice	Eve
Spirit	Alice	Bob	Eve
Swiftwind	Alice	Bob	Eve

RUNNING GS

 >  > 

Alice	Donkey	Spirit	Swiftwind
Bob	Spirit	Donkey	Swiftwind
Eve	Donkey	Spirit	Swiftwind

Donkey	Bob	Alice	Eve
Spirit	Alice	Bob	Eve
Swiftwind	Alice	Bob	Eve

Claim

GS terminates in polynomial time (at most n^2 iterations of the outer loop)

Proof:

- Each iteration, one jockey proposes to someone to whom they have never proposed before
- *n* horses, *n* jockeys → $n \times n$ possible events

(Can tighten a bit to $n(n - 1) + 1$ iterations.)

Claim

GS results in a perfect matching

Proof by contradiction:

- Suppose BWOC that j is unmatched at termination
- n horses, n jockeys → h is unmatched, too
- Once a horse is proposed to, they are matched and never unmatched; they only swap partners. Thus, nobody proposed to h
- j proposed to everyone (by def. of GS): ><

Claim

GS results in a stable matching (i.e., there are no blocking pairs)

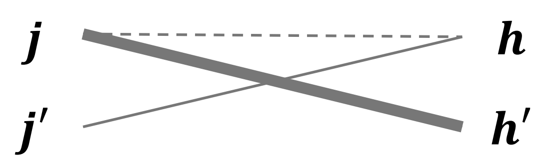
Proof by contradiction (1):

- Assume j and h form a blocking pair

Case #1: j never proposed to h

- GS: jockeys propose in order of preferences
- j prefers current match $h' > h$ (since it proposed to $\mathbf{h}'$ and not $\mathbf{h}$)
- → j and h are not blocking

j and h are not matched
j and h' are matched
j' and h are matched



j prefers h to h'
h prefers j to j'

js proposals:
…
h'
…
h
…

Claim

GS results in a stable matching (i.e., there are no blocking pairs)

Proof by contradiction (2):

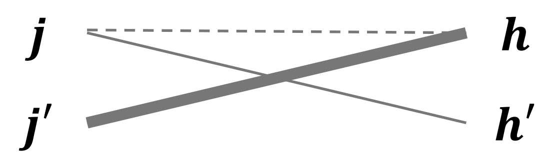
Case #2: j proposed to h

- h rejected j at some point
- *GS:* horses only reject for better jockeys
- h prefers current partner $j' > j$
- → j and h are not blocking

Case #1 and #2 exhaust space.

><

j and h are not matched
j and h' are matched
j' and h are matched



j prefers h to h'
h prefers j to j'

h proposers:	h rejections
…	…
j	…
…	…
j'	j
…	…

RECAP: SOME QUESTIONS

Does a stable solution to the marriage problem always exist?

Can we compute such a solution efficiently?

Can we compute the best stable solution efficiently?

We'll look at a specific notion of "the best" – optimality with respect to one side of the market

HORSE/JOCKEY OPTIMALITY/PESSIMALITY

Let $\mathcal{S}$ be the set of stable matchings

j is a valid partner of h (and vice versa) if there exists some stable matching S in $\mathcal{S}$ where they are paired

A matching is jockey optimal (resp. horse optimal) if each jockey (resp. horse) receives their *best* valid partner

- Is this a perfect matching? Stable?

A matching is jockey pessimal (resp. horse pessimal) if each jockey (resp. horse) receives their *worst* valid partner

Claim

GS – with the jockey proposing – results in a jockey-optimal matching

Proof by contradiction (1):

- Jockey propose in order → at least one jockey was rejected by a valid partner
- Let $\boldsymbol{j}$ and $\boldsymbol{h}$ be the first such reject in S

$\boldsymbol{j}$ and $\boldsymbol{h}$ are not matched in S, they are valid partners

$\boldsymbol{j}$ $\boldsymbol{h}$

S is stable

Claim

GS – with the jockey proposing – results in a jockey-optimal matching

Proof by contradiction (1):

- Jockey propose in order → at least one jockey was rejected by a valid partner
- Let $\boldsymbol{j}$ and $\boldsymbol{h}$ be the first such reject in S
- Let S' be a stable matching with $\boldsymbol{j}$, $\boldsymbol{h}$ paired
  (S' exists by def. of valid)

$\boldsymbol{j}$ and $\boldsymbol{h}$ are not matched in S, they are valid partners
$\boldsymbol{j}$ and $\boldsymbol{h}$ are matched in S'

$\boldsymbol{j} \overset{S'}{\text{- - - - - - - - - -}} \boldsymbol{h}$

S is stable
S' is stable

Claim

GS – with the jockey proposing – results in a jockey-optimal matching

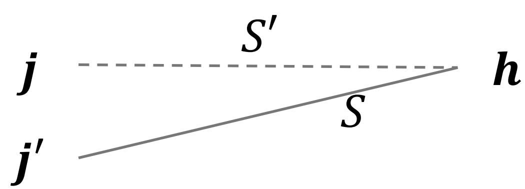
Proof by contradiction (1):

- Jockey propose in order → at least one jockey was rejected by a valid partner
- Let $\boldsymbol{j}$ and $\boldsymbol{h}$ be the first such reject in S
- Let S' be a stable matching with $\boldsymbol{j}$, $\boldsymbol{h}$ paired
  - (S' exists by def. of valid)
- $\boldsymbol{j}$ is rejected in S because $\boldsymbol{h}$ chose some $\boldsymbol{j}' > \boldsymbol{j}$

$\boldsymbol{j}$ and $\boldsymbol{h}$ are not matched in S, they are valid partners

$\boldsymbol{j}$ and $\boldsymbol{h}$ are matched in S'

$\boldsymbol{j}'$ and $\boldsymbol{h}$ are matched in S



S is stable

S' is stable

Claim

GS – with the jockey proposing – results in a jockey-optimal matching

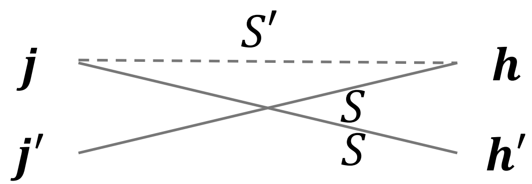
Proof by contradiction (2):

- Let $\boldsymbol{h}'$ be match of $\boldsymbol{j}'$ in S'

$\boldsymbol{j}$ and $\boldsymbol{h}$ are not matched in S, they are valid partners

$\boldsymbol{j}$ and $\boldsymbol{h}$ are matched in S'

$\boldsymbol{j}'$ and $\boldsymbol{h}$ are matched in S



S is stable

S' is stable

Claim

GS – with the jockey proposing – results in a jockey-optimal matching

Proof by contradiction (2):

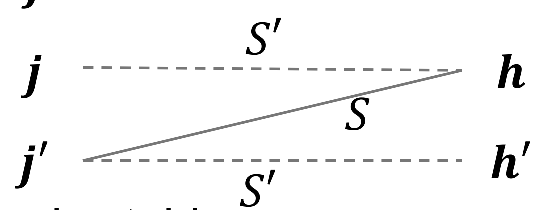
- Let h' be match of j' in S'
- j' was not rejected by valid partner in S before j was rejected by h (by assump.) → *j' prefers h to h'*

j and h are not matched in S, they are valid partners

j and h are matched in S'

j' and h are matched in S

j' and h' are matched in S'



S is stable

S' is stable

Can't come before h since j' and h' are valid partners

j' proposals:
…
h
…
h' ✖
…

Claim

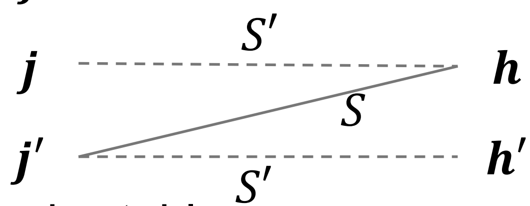
GS – with the jockey proposing – results in a jockey-optimal matching

Proof by contradiction (2):

- Let h' be match of j' in S'
- j' was not rejected by valid partner in S before j was rejected by h (by assump.)
  → *j' prefers h to h'*
- Know h prefers j' over j, their jockey in S'
  → j' and h form a blocking

pair in S' ><

j and h are not matched in S, they are valid partners
j and h are matched in S'
j' and h are matched in S
j' and h' are matched in S'



S is stable
S' is stable

h proposers:
…
j
…
j' ✓
…

j' proposals:
…
h
…
h' ✗
…

RECAP: SOME QUESTIONS

Does a stable solution to the marriage problem always exist?

Can we compute such a solution efficiently?

Can we compute the best stable solution efficiently? *

For one side of the market. What about the other side?

Claim

GS – with the jockey proposing – results in a horse-pessimal matching

Proof by contradiction:

- $\boldsymbol{j}$ and $\boldsymbol{h}$ matched in S, $\boldsymbol{j}$ is not worst valid

$\boldsymbol{j}$ and $\boldsymbol{h}$ are matched in S
$\boldsymbol{j}'$ and $\boldsymbol{h}$ are matched in S'
$\boldsymbol{j}$ and $\boldsymbol{h}'$ are matched in S'

$$\boldsymbol{j} \;\overset{S}{\overline{\qquad\qquad\qquad}}\; \boldsymbol{h}$$

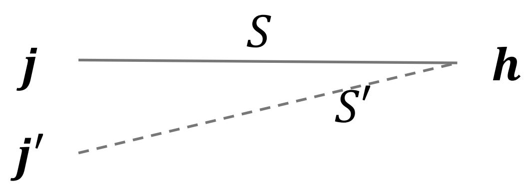
Claim

GS – with the jockey proposing – results in a horse-pessimal matching

Proof by contradiction:

- $\boldsymbol{j}$ and $\boldsymbol{h}$ matched in S, $\boldsymbol{j}$ is not worst valid
- → exists stable S' with $\boldsymbol{h}$ paired to $\boldsymbol{j}'$, where $\boldsymbol{h}$ prefers to $\boldsymbol{j}$ to $\boldsymbol{j}'$

$\boldsymbol{j}$ and $\boldsymbol{h}$ are matched in S
$\boldsymbol{j}'$ and $\boldsymbol{h}$ are matched in S'
$\boldsymbol{j}$ and $\boldsymbol{h}'$ are matched in S'



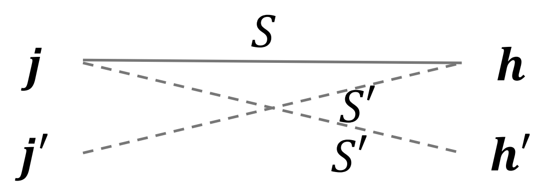
Claim

GS – with the jockey proposing – results in a horse-pessimal matching

Proof by contradiction:

- $\boldsymbol{j}$ and $\boldsymbol{h}$ matched in S, $\boldsymbol{j}$ is not worst valid
- → exists stable S' with $\boldsymbol{h}$ paired to $\boldsymbol{j}'$, where $\boldsymbol{h}$ prefers to $\boldsymbol{j}$ to $\boldsymbol{j}'$
- Let $\boldsymbol{h}'$ be partner of $\boldsymbol{j}$ in S'

$\boldsymbol{j}$ and $\boldsymbol{h}$ are matched in S
$\boldsymbol{j}'$ and $\boldsymbol{h}$ are matched in S'
$\boldsymbol{j}$ and $\boldsymbol{h}'$ are matched in S'



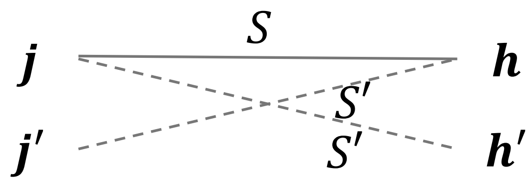
Claim

GS – with the jockey proposing – results in a horse-pessimal matching

Proof by contradiction:

- $\boldsymbol{j}$ and $\boldsymbol{h}$ matched in S, $\boldsymbol{j}$ is not worst valid
- → exists stable S' with $\boldsymbol{h}$ paired to $\boldsymbol{j}'$, where $\boldsymbol{h}$ prefers to $\boldsymbol{j}$ to $\boldsymbol{j}'$
- Let $\boldsymbol{h}'$ be partner of $\boldsymbol{j}$ in S'
- $\boldsymbol{j}$ prefers to $\boldsymbol{h}$ to $\boldsymbol{h}'$ (by jockey-optimality of S)

$\boldsymbol{j}$ and $\boldsymbol{h}$ are matched in S
$\boldsymbol{j}'$ and $\boldsymbol{h}$ are matched in S'
$\boldsymbol{j}$ and $\boldsymbol{h}'$ are matched in S'



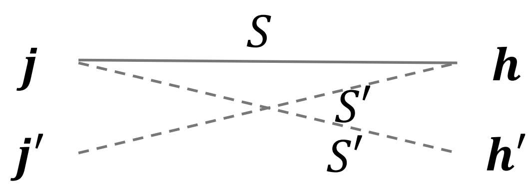
Claim

GS – with the jockey proposing – results in a horse-pessimal matching

Proof by contradiction:

- $\boldsymbol{j}$ and $\boldsymbol{h}$ matched in S, $\boldsymbol{j}$ is not worst valid
- → exists stable S' with $\boldsymbol{h}$ paired to $\boldsymbol{j}'$, where $\boldsymbol{h}$ prefers to $\boldsymbol{j}$ to $\boldsymbol{j}'$
- Let $\boldsymbol{h}'$ be partner of $\boldsymbol{j}$ in S'
- $\boldsymbol{j}$ prefers to $\boldsymbol{h}$ to $\boldsymbol{h}'$ (by jockey-optimality of S)
- → $\boldsymbol{j}$ and $\boldsymbol{h}$ form blocking pair in S' ><

$\boldsymbol{j}$ and $\boldsymbol{h}$ are matched in S
$\boldsymbol{j}'$ and $\boldsymbol{h}$ are matched in S'
$\boldsymbol{j}$ and $\boldsymbol{h}'$ are matched in S'



INCENTIVE ISSUES

Can either side benefit by misreporting?

- (Slight extension for rest of talk: participants can mark possible matches as unacceptable – a form of preference list truncation)

Any algorithm that yields a jockey-(horse-)optimal matching
→
truthful revelation by jockeys (horses) is dominant strategy [Roth 1982]

In GS with jockey proposing, horses can benefit by misreporting preferences

Truthful reporting

Alice	Donkey	Spirit
Bob	Spirit	Donkey

Donkey	Bob	Alice
Spirit	Alice	Bob

Alice	Donkey	Spirit
Bob	Spirit	Donkey

Donkey	Bob	Alice
Spirit	Alice	Bob

Strategic reporting

Alice	Donkey	Spirit
Bob	Spirit	Donkey

Donkey	Bob	⃠
Spirit	Alice	Bob

Alice	Donkey	Spirit
Bob	Spirit	Donkey

Donkey	Bob	⃠
Spirit	Alice	Bob

Claim

There is **no** matching mechanism that:

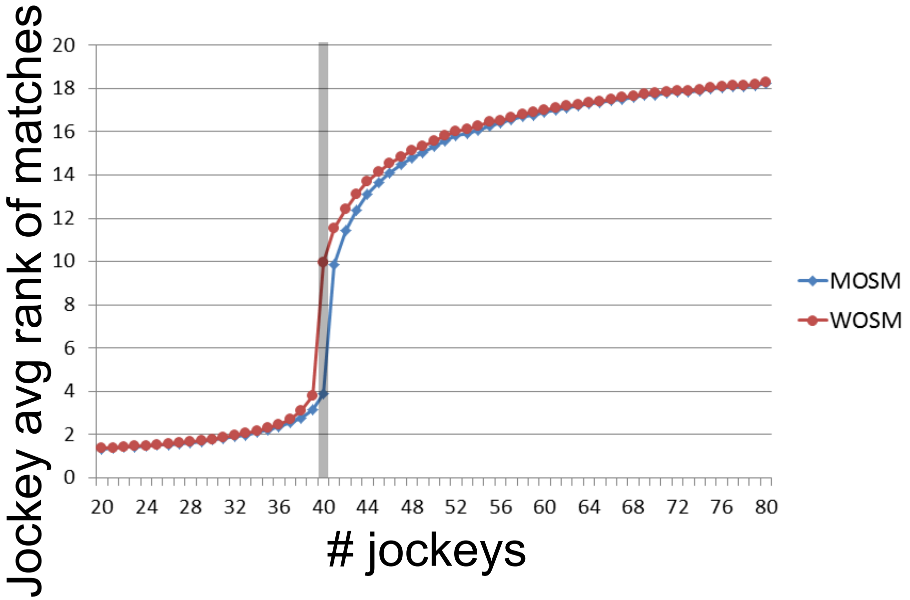
1. is strategy proof (for both sides); and
2. always results in a stable outcome (given revealed preferences)

EXTENSIONS TO STABLE MATCHING

IMBALANCE [ASHLAGI ET AL. 2013]

What if we have *n* jockeys and *n'* ≠ *n* horses?

How does this affect participants? Core size?

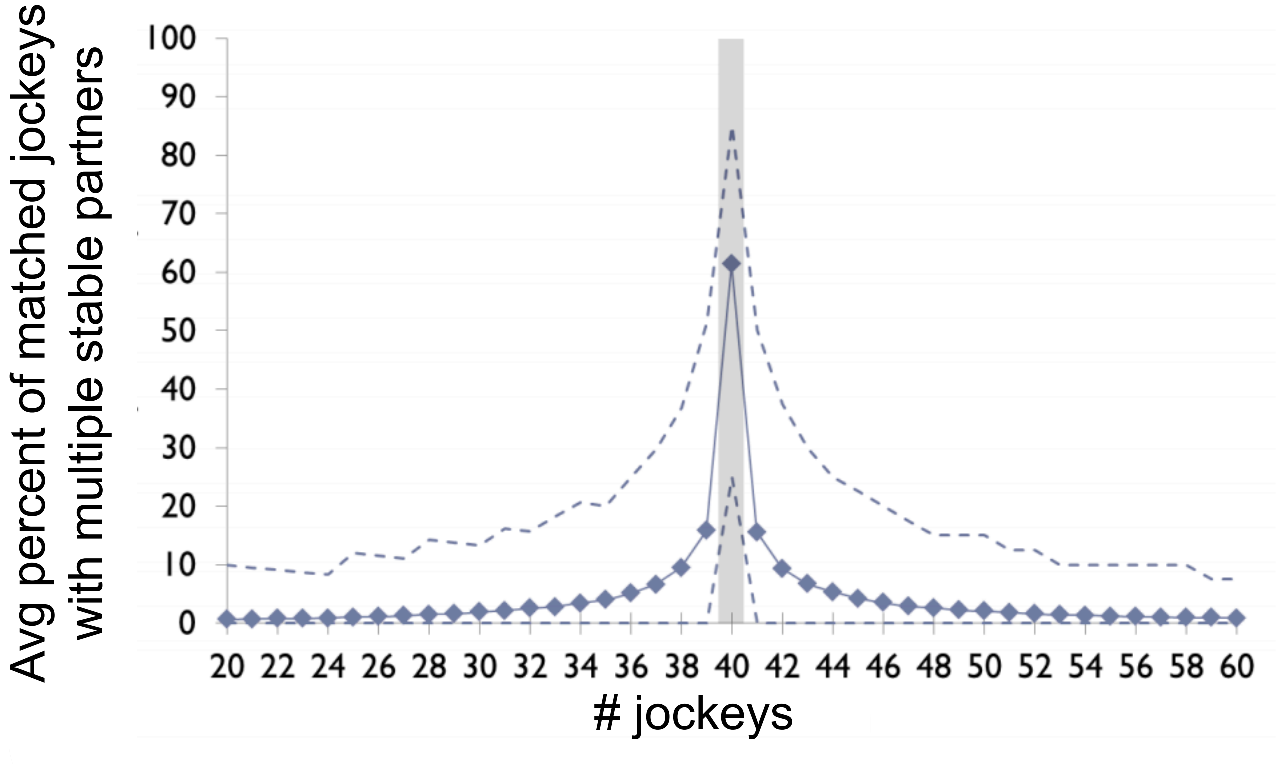


 # horses held constant at n' = 40

- Being on short side of market: good!
- W.h.p., short side get rank ~*log*(*n*)
- *…* long side gets rank ~random

IMBALANCE [ASHLAGI ET AL. 2013]

Not many stable matchings with even small imbalances in the market



IMBALANCE [ASHLAGI ET AL. 2013]

“Rural hospital theorem” [Roth 1986]:

- The set of jockeys and horses that are unmatched is **the same** for all stable matchings

Assume *n* jockeys, *n+1* horses

- One horse h unmatched in all stable matchings
- → Drop h, same stable matchings

Take stable matchings with *n* horses

- Stay stable when we add in h **if** no jockeys prefer h to their current match
- → average rank of jockey’s matches is low

ONLINE ARRIVAL [KHULLER ET AL. 1993]

Random preferences, jockeys arrive over time, once matched nobody can switch

Algorithm: match j to highest-ranked free h

- On average, $O(n\log(n))$ unstable pairs

No deterministic or randomized algorithm can do better than $\Omega(n^2)$ unstable pairs!

- Not better with randomization ☹

INCOMPLETE PREFS
[MANLOVE ET AL. 2002]

Before: complete + strict preferences

- Easy to compute, lots of nice properties

Incomplete preferences

- May exist: stable matchings of **different sizes**

Everything becomes hard!

- Finding max or min cardinality stable matching
- Determining if $< \boldsymbol{j}, \boldsymbol{h} >$ are stable
- Finding/approx. finding "egalitarian" matching

NON-BIPARTITE GRAPH …?

Matching is defined on general graphs:

- “Set of edges, each vertex included at most once”

The stable roommates problem is bipartite stable matching generalized to any graph

Each vertex ranks all n-1 other vertices

- (Variations with/without truncation)

Same notion of stability

IS THIS DIFFERENT THAN BIPARTITE STABLE MATCHING?

 >  > 

Alana	Brian	Cynthia	Dracula
Brian	Cynthia	Alana	Dracula
Cynthia	Alana	Brian	Dracula
Dracula	(Anyone)	(Anyone)	(Anyone)

No stable matching exists!

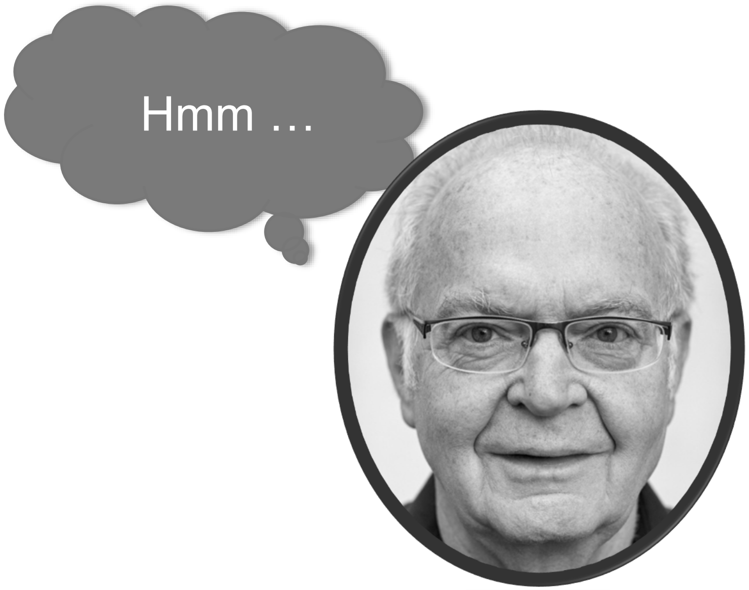
Anyone paired with Dracula (i) prefers some other *v* and (ii) is preferred by that *v*

HOPELESS?

Can we build an algorithm that:

- Finds a stable matching; or
- Reports nonexistence

… In polynomial time?



Yes! [Irving 1985]

- Builds on Gale-Shapley ideas and work by McVitie and Wilson [1971]

IRVING'S ALGORITHM: PHASE 1

Idea: Run an algorithm very similar to Gale-Shapley

- Everyone proposes to everyone
- Individuals hold 2 types of temporary matches: matches where they propose and matches where they are proposed to (the former *will* be weakly better)

After this step: one person is unmatched → nonexistence

Else: create a reduced set of preferences

- *a* holds proposal from *b* → *a* truncates all *x* after *b*
- For each removed *x*, also remove *a* from *x*'s preferences
- Note: *b* at end of *a*'s list → *a* at start of *b*'s list

If any reduced set is empty: nonexistence

Else: this is a "stable table" – continue to Phase 2

RUNNING THE ALGORITHM: PHASE 1

Example from: https://www.youtube.com/watch?v=9Lo7TFAkohE&ab_channel=OscarRobertson

		>	>	>	>
Alice	Bob	Dave	Frank	Eve	Carol
Bob	Eve	Carol	Frank	Dave	Alice
Carol	Bob	Frank	Eve	Alice	Dave
Dave	Carol	Alice	Frank	Bob	Eve
Eve	Bob	Alice	Frank	Carol	Dave
Frank	Alice	Dave	Eve	Carol	Bob

Green = Locked proposal from self
Blue = Locked proposal to self
Yellow = Currently proposing
Orange = Currently being proposed to

RUNNING THE ALGORITHM: PHASE 1

	>	>	>	>	
Alice	Bob	Dave	Frank	Eve	Carol
Bob	Eve	Carol	Frank	Dave	Alice
Carol	Bob	Frank	Eve	Alice	Dave
Dave	Carol	Alice	Frank	Bob	Eve
Eve	Bob	Alice	Frank	Carol	Dave
Frank	Alice	Dave	Eve	Carol	Bob

Green = Locked proposal from self
Blue = Locked proposal to self
Yellow = Currently proposing
Orange = Currently being proposed to

RUNNING THE ALGORITHM: PHASE 1

	>	>	>	>	
Alice	Bob	Dave	Frank	Eve	Carol
Bob	Eve	Carol	Frank	Dave	Alice
Carol	Bob	Frank	Eve	Alice	Dave
Dave	Carol	Alice	Frank	Bob	Eve
Eve	Bob	Alice	Frank	Carol	Dave
Frank	Alice	Dave	Eve	Carol	Bob

Green = Locked proposal from self
Blue = Locked proposal to self
Yellow = Currently proposing
Orange = Currently being proposed to

RUNNING THE ALGORITHM: PHASE 1

> > > >

Alice	Bob	Dave	Frank	Eve	Carol
Bob	Eve	Carol	Frank	Dave	Alice
Carol	Bob	Frank	Eve	Alice	Dave
Dave	Carol	Alice	Frank	Bob	Eve
Eve	Bob	Alice	Frank	Carol	Dave
Frank	Alice	Dave	Eve	Carol	Bob

Green = Locked proposal from self
Blue = Locked proposal to self
Yellow = Currently proposing
Orange = Currently being proposed to

RUNNING THE ALGORITHM: PHASE 1

	>	>	>	>	
Alice	Bob	Dave	Frank	Eve	Carol
Bob	Eve	Carol	Frank	Dave	Alice
Carol	Bob	Frank	Eve	Alice	Dave
Dave	Carol	Alice	Frank	Bob	Eve
Eve	Bob	Alice	Frank	Carol	Dave
Frank	Alice	Dave	Eve	Carol	Bob

Green = Locked proposal from self
Blue = Locked proposal to self
Yellow = Currently proposing
Orange = Currently being proposed to

RUNNING THE ALGORITHM: PHASE 1

		>	>	>	>
Alice	Bob	Dave	Frank	Eve	Carol
Bob	Eve	Carol	Frank	Dave	Alice
Carol	Bob	Frank	Eve	Alice	Dave
Dave	Carol	Alice	Frank	Bob	Eve
Eve	Bob	Alice	Frank	Carol	Dave
Frank	Alice	Dave	Eve	Carol	Bob

Green = Locked proposal from self
Blue = Locked proposal to self
Yellow = Currently proposing
Orange = Currently being proposed to

RUNNING THE ALGORITHM: PHASE 1

		>	>	>	>
Alice		Dave	Frank	Eve	Carol
Bob	Eve	Carol	Frank	Dave	
Carol	Bob	Frank	Eve	Alice	Dave
Dave	Carol	Alice	Frank	Bob	Eve
Eve	Bob	Alice	Frank	Carol	Dave
Frank	Alice	Dave	Eve	Carol	Bob

Green = Locked proposal from self
Blue = Locked proposal to self
Yellow = Currently proposing
Orange = Currently being proposed to

RUNNING THE ALGORITHM: PHASE 1

		>	>	>	>
Alice		Dave	Frank	Eve	Carol
Bob	Eve	Carol	Frank	Dave	
Carol	Bob	Frank	Eve	Alice	Dave
Dave	Carol	Alice	Frank	Bob	Eve
Eve	Bob	Alice	Frank	Carol	Dave
Frank	Alice	Dave	Eve	Carol	Bob

Green = Locked proposal from self
Blue = Locked proposal to self
Yellow = Currently proposing
Orange = Currently being proposed to

RUNNING THE ALGORITHM: PHASE 1

		>	>	>	>
Alice		Dave	Frank	Eve	Carol
Bob	Eve	Carol	Frank	Dave	
Carol	Bob	Frank	Eve	Alice	Dave
Dave	Carol	Alice	Frank	Bob	Eve
Eve	Bob	Alice	Frank	Carol	Dave
Frank	Alice	Dave	Eve	Carol	Bob

Green = Locked proposal from self
Blue = Locked proposal to self
Yellow = Currently proposing
Orange = Currently being proposed to

RUNNING THE ALGORITHM: PHASE 1

		>	>	>	>
Alice		Dave	Frank	Eve	Carol
Bob	Eve	Carol	Frank	Dave	
Carol	Bob	Frank	Eve	Alice	Dave
Dave	Carol	Alice	Frank	Bob	Eve
Eve	Bob	Alice	Frank	Carol	Dave
Frank	Alice	Dave	Eve	Carol	Bob

Green = Locked proposal from self
Blue = Locked proposal to self
Yellow = Currently proposing
Orange = Currently being proposed to

RUNNING THE ALGORITHM: PHASE 1

		>	>	>	>
Alice		Dave	Frank	Eve	Carol
Bob	Eve	Carol	Frank	Dave	
Carol	Bob	Frank	Eve	Alice	Dave
Dave	Carol	Alice	Frank	Bob	Eve
Eve	Bob	Alice	Frank	Carol	Dave
Frank	Alice	Dave	Eve	Carol	Bob

Green = Locked proposal from self
Blue = Locked proposal to self
Yellow = Currently proposing
Orange = Currently being proposed to

RUNNING THE ALGORITHM: PHASE 1

		>	>	>	>
Alice		Dave	Frank	Eve	Carol
Bob	Eve	Carol	Frank	Dave	
Carol	Bob	Frank	Eve	Alice	Dave
Dave	Carol	Alice	Frank	Bob	Eve
Eve	Bob	Alice	Frank	Carol	Dave
Frank	Alice	Dave	Eve	Carol	Bob

Green = Locked proposal from self
Blue = Locked proposal to self
Yellow = Currently proposing
Orange = Currently being proposed to

RUNNING THE ALGORITHM: PHASE 1

		>	>	>	>
Alice		Dave	Frank	Eve	Carol
Bob	Eve		Frank	Dave	
Carol		Frank	Eve	Alice	Dave
Dave	Carol	Alice	Frank	Bob	Eve
Eve	Bob	Alice	Frank	Carol	Dave
Frank	Alice	Dave	Eve	Carol	Bob

Green = Locked proposal from self
Blue = Locked proposal to self
Yellow = Currently proposing
Orange = Currently being proposed to

RUNNING THE ALGORITHM: PHASE 1

		>	>	>	>
Alice		Dave	Frank	Eve	Carol
Bob	Eve		Frank	Dave	
Carol		Frank	Eve	Alice	Dave
Dave	Carol	Alice	Frank	Bob	Eve
Eve	Bob	Alice	Frank	Carol	Dave
Frank	Alice	Dave	Eve	Carol	Bob

Green = Locked proposal from self
Blue = Locked proposal to self
Yellow = Currently proposing
Orange = Currently being proposed to

RUNNING THE ALGORITHM: PHASE 1

		>	>	>	>
Alice		Dave	Frank	Eve	Carol
Bob	Eve		Frank	Dave	
Carol		Frank	Eve	Alice	Dave
Dave	Carol	Alice	Frank	Bob	Eve
Eve	Bob	Alice	Frank	Carol	Dave
Frank	Alice	Dave	Eve	Carol	Bob

Green = Locked proposal from self
Blue = Locked proposal to self
Yellow = Currently proposing
Orange = Currently being proposed to

RUNNING THE ALGORITHM: PHASE 1

		>	>	>	>
Alice		Dave	Frank	Eve	Carol
Bob	Eve		Frank	Dave	
Carol		Frank	Eve	Alice	Dave
Dave	Carol	Alice	Frank	Bob	Eve
Eve	Bob	Alice	Frank	Carol	Dave
Frank	Alice	Dave	Eve	Carol	Bob

Green = Locked proposal from self
Blue = Locked proposal to self
Yellow = Currently proposing
Orange = Currently being proposed to

RUNNING THE ALGORITHM: PHASE 1

		>	>	>	>
Alice		Dave	Frank	Eve	Carol
Bob	Eve		Frank	Dave	
Carol		Frank	Eve	Alice	Dave
Dave	Carol	Alice	Frank	Bob	Eve
Eve	Bob	Alice	Frank	Carol	Dave
Frank	Alice	Dave	Eve	Carol	Bob

Green = Locked proposal from self
Blue = Locked proposal to self
Yellow = Currently proposing
Orange = Currently being proposed to

RUNNING THE ALGORITHM: PHASE 1

Remove anyone below the proposal offered to you

		>	>	>	>
Alice		Dave	Frank		
Bob	Eve				
Carol		Frank	Eve	Alice	Dave
Dave	Carol	Alice			
Eve	Bob				
Frank	Alice	Dave	Eve	Carol	

Green = Locked proposal from self
Blue = Locked proposal to self

RUNNING THE ALGORITHM: PHASE 1

If you are not on someone else's list, remove them from your list

		>	>	>	>
Alice		Dave	Frank		
Bob	Eve				
Carol		Frank			Dave
Dave	Carol	Alice			
Eve	Bob				
Frank	Alice			Carol	

Green = Locked proposal from self
Blue = Locked proposal to self

RUNNING THE ALGORITHM: PHASE 1

If you are not on someone else's list, remove them from your list

> > > >

Alice	Dave	Frank
Bob	Eve	
Carol	Frank	Dave
Dave	Carol	Alice
Eve	Bob	
Frank	Alice	Carol

Green = Locked proposal from self
Blue = Locked proposal to self

STABLE TABLES

1. ***a* is first on *b*'s list iff *b* is last on *a*'s**
2. ***a* is not on *b*'s list iff**
   - *b* is not on *a*'s list
   - *a* prefers last element on list to *b*
3. **No reduced list is empty**

Note 1: stable table with all lists length 1 is a stable matching

Note 2: any stable subtable of a stable table can be obtained via *rotation eliminations*

Phase 2!

IRVING'S ALGORITHM: PHASE 2

Stable table has length 1 lists: return matching

Identify a rotation:

$(a_0,b_0),(a_1,b_1),\ldots,(a_{k-1},b_{k-1})$ such that:

- b_i is a_i's second preference
- a_{i+1} is b_i's last preference
- a_0 is b_{k-1}'s last preference (i.e., we have cycled)

Eliminate it:

- b_i rejects a_{i+1}, and repeat rotation finding as necessary

If any list becomes empty: nonexistence

If the subtable hits length 1 lists: return matching

RUNNING THE ALGORITHM: PHASE 2

Alice a_0	Dave	Frank b_0
Bob	Eve	
Carol a_1	Frank	Dave b_1
Dave b_1	Carol	Alice a_2
Eve	Bob	
Frank b_0	Alice	Carol a_1

$(a_0,b_0),(a_1,b_1),\ldots,(a_{k-1},b_{k-1})$ such that:

- b_i is a_i's second preference
- a_{i+1} is b_i's last preference
- a_0 is b_{k-1}'s last preference

RUNNING THE ALGORITHM: PHASE 2

Alice a_0	Dave	Frank b_0
Bob	Eve	
Carol a_1	Frank	Dave b_1
Dave b_1	Carol	Alice a_2
Eve	Bob	
Frank b_0	Alice	Carol a_1

$(a_0,b_0),(a_1,b_1),\ldots,(a_{k-1},b_{k-1})$ such that:

- b_i is a_i's second preference
- a_{i+1} is b_i's last preference
- a_0 is b_{k-1}'s last preference

Next: b_i rejects a_{i+1}

RUNNING THE ALGORITHM: PHASE 2

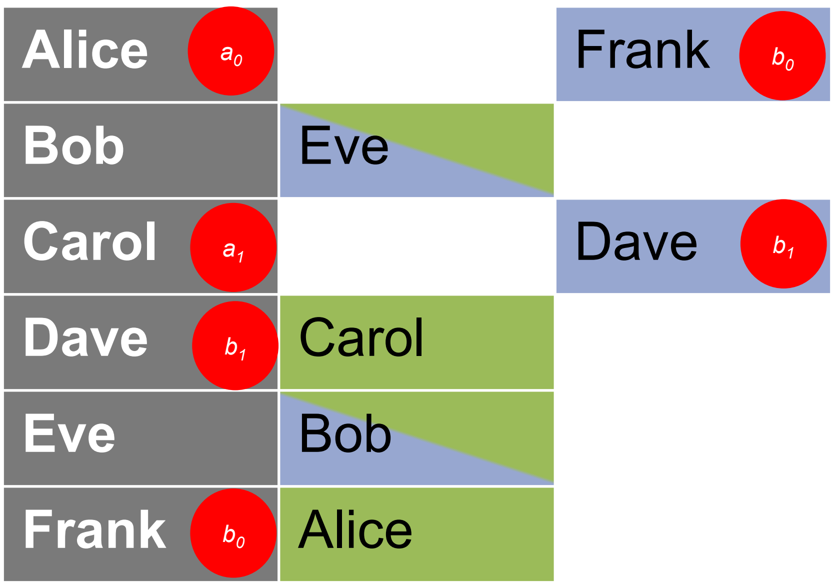
Alice a_0	Dave	Frank b_0
Bob	Eve	
Carol a_1		Dave b_1
Dave b_1	Carol	Alice a_2
Eve	Bob	
Frank b_0	Alice	

$(a_0,b_0),(a_1,b_1),\ldots,(a_{k-1},b_{k-1})$ such that:

- b_i is a_i's second preference
- a_{i+1} is b_i's last preference
- a_0 is b_{k-1}'s last preference

Next: b_i rejects a_{i+1}

RUNNING THE ALGORITHM: PHASE 2



$(a_0,b_0),(a_1,b_1),\ldots,(a_{k-1},b_{k-1})$ such that:

- b_i is a_i's second preference
- a_{i+1} is b_i's last preference
- a_0 is b_{k-1}'s last preference

Next: b_i rejects a_{i+1}

Claim

Irving's algorithm for the stable roommates problem terminates in polynomial time – specifically O(n^2).

This requires some data structure considerations

- Naïve implementation of rotations is ~O(n^3)

ONE-TO-MANY MATCHING

The hospitals/residents problem (aka college/students problem aka admissions problem):

- Strict preference rankings from each side
- One side (hospitals) can accept $q > 1$ residents

Also introduced in [Gale and Shapley 1962]

Has seen lots of traction in the real world

- E.g., the **National Resident Matching Program** (NRMP)
- Other American, British, and Canadian medical labor markets
- Canadian lawyer labor markets
- Sororities

HISTORY OF THE NRMP [Roth 2002]

1900-ish first medical internships

1940's Fierce competition, eventual market failure

1951 First centralized clearing market

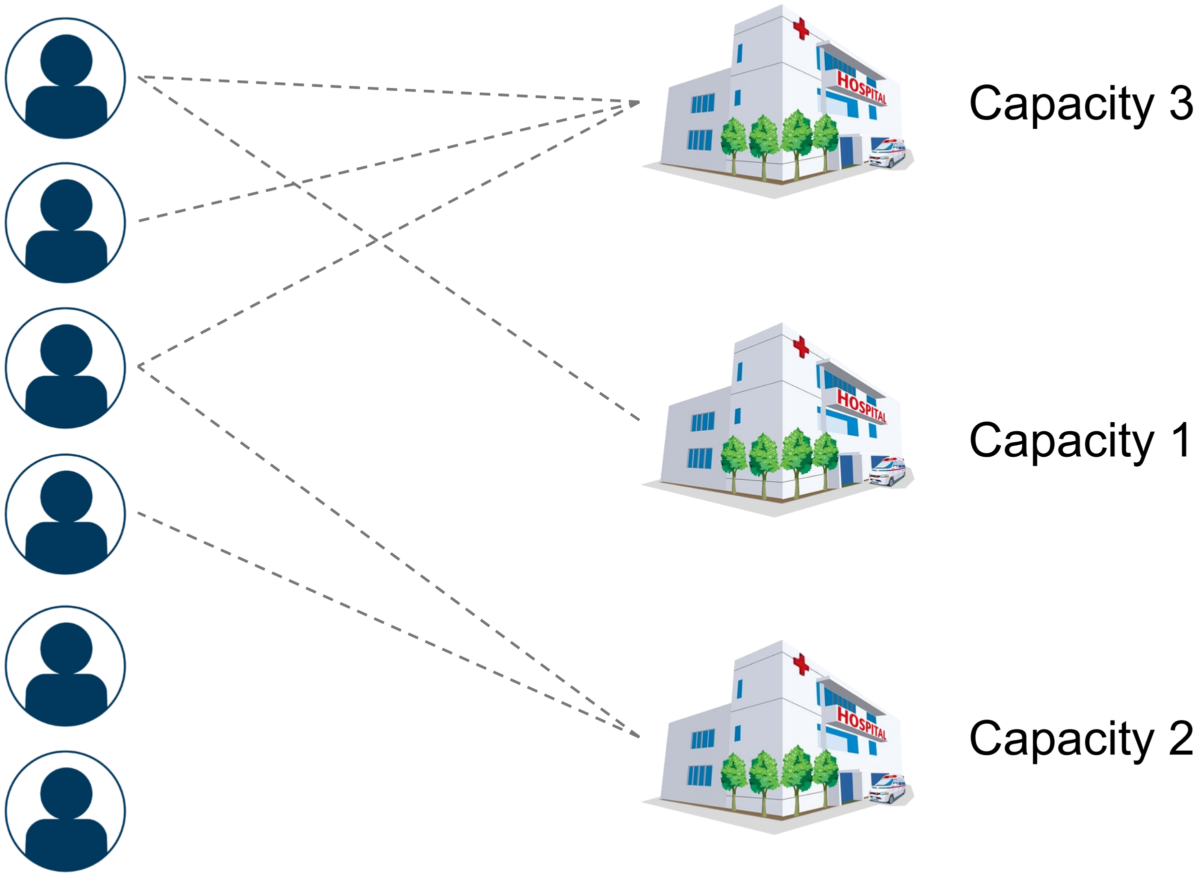
1970s More couples on the market

1995 Crisis of confidence

1996-7 Redesign completed and implemented

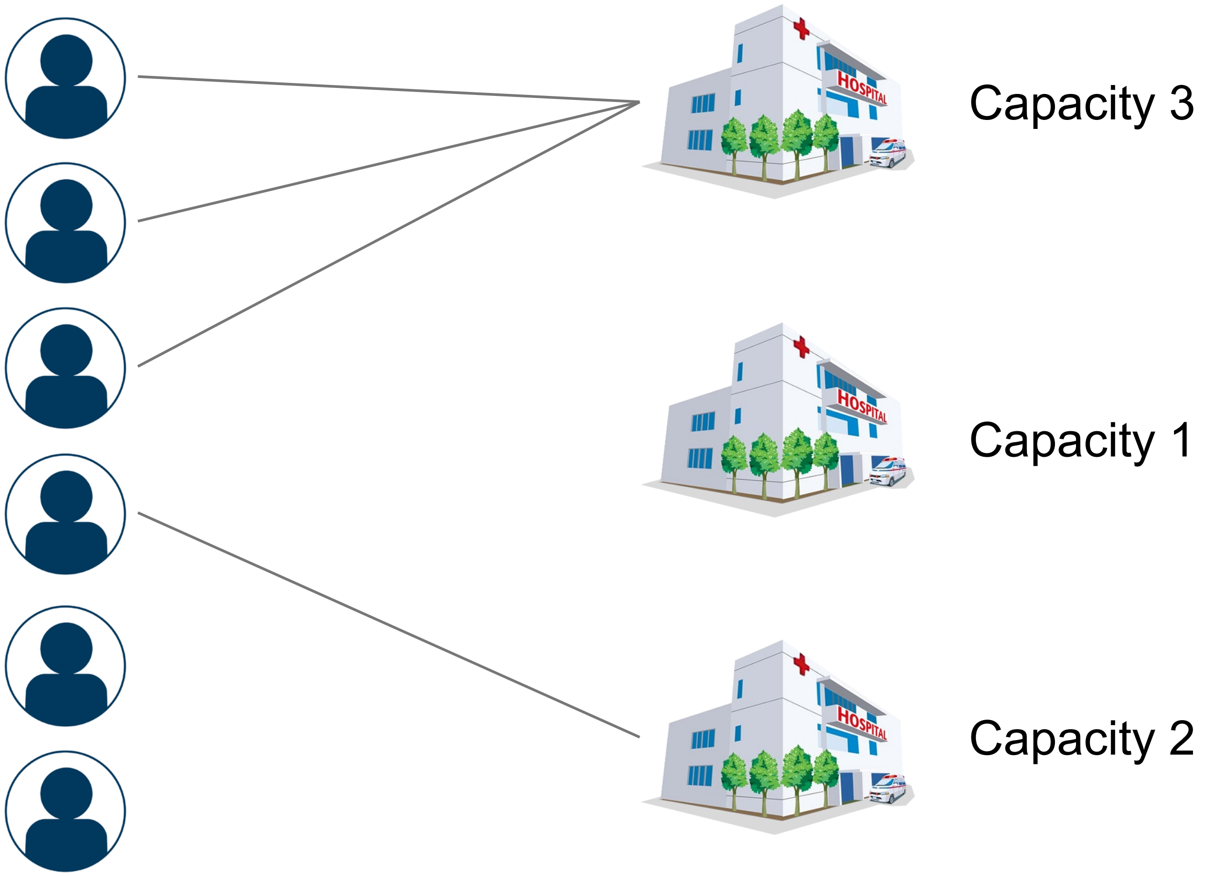
NRMP SIMPLIFIED:
A GALE-SHAPLEY APPROACH

Idea: Hospitals propose to as many doctors as they have open spots. Doctors accept/reject as normal. Repeat until done.



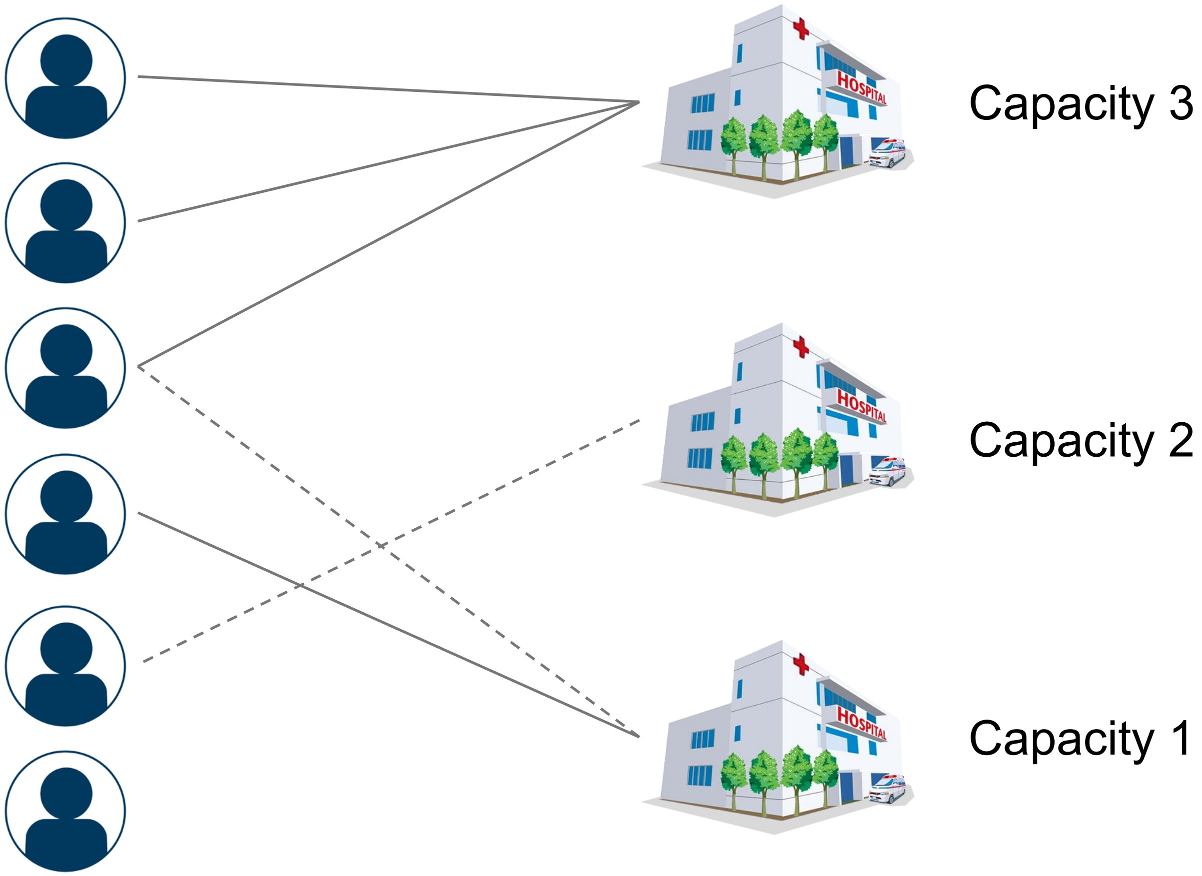
NRMP SIMPLIFIED:
A GALE-SHAPLEY APPROACH

Idea: Hospitals propose to as many doctors as they have open spots. Doctors accept/reject as normal. Repeat until done.



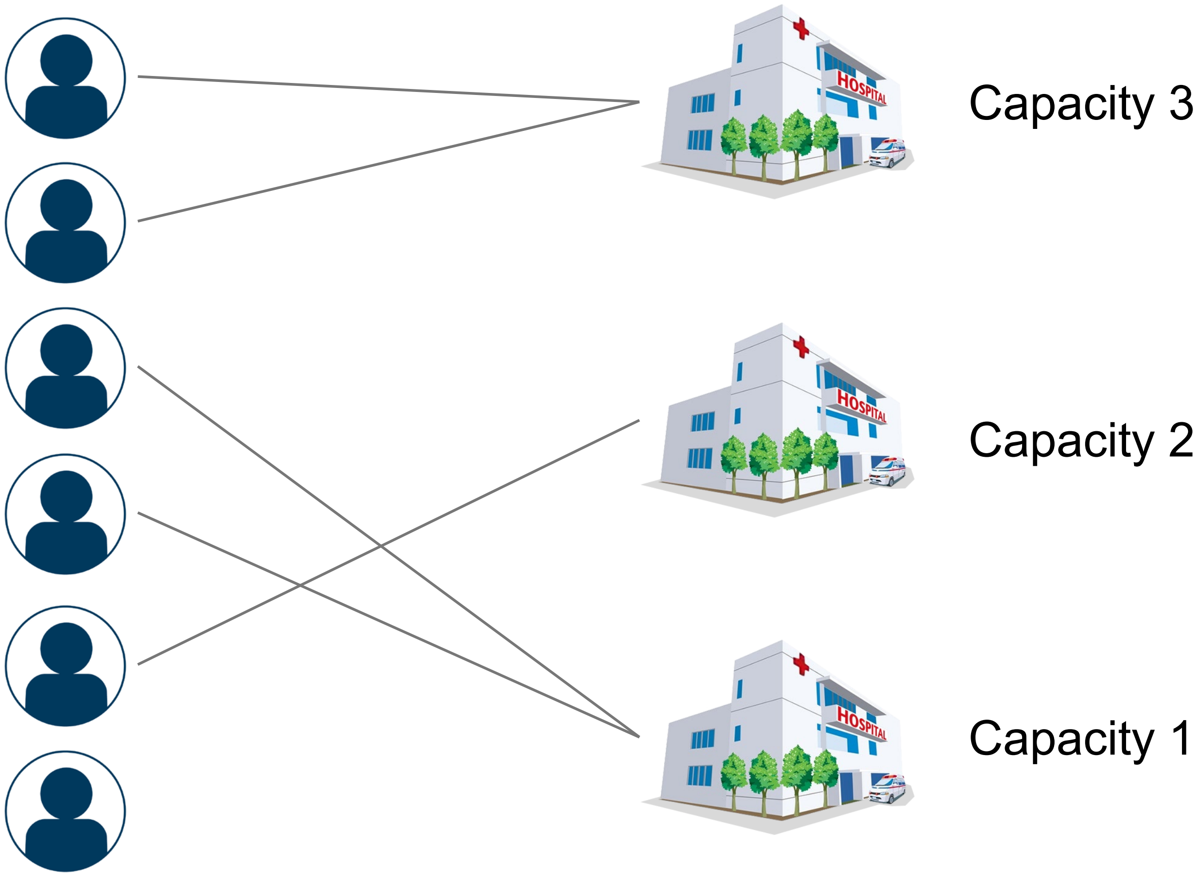
NRMP SIMPLIFIED: A GALE-SHAPLEY APPROACH

Idea: Hospitals propose to as many doctors as they have open spots. Doctors accept/reject as normal. Repeat until done.



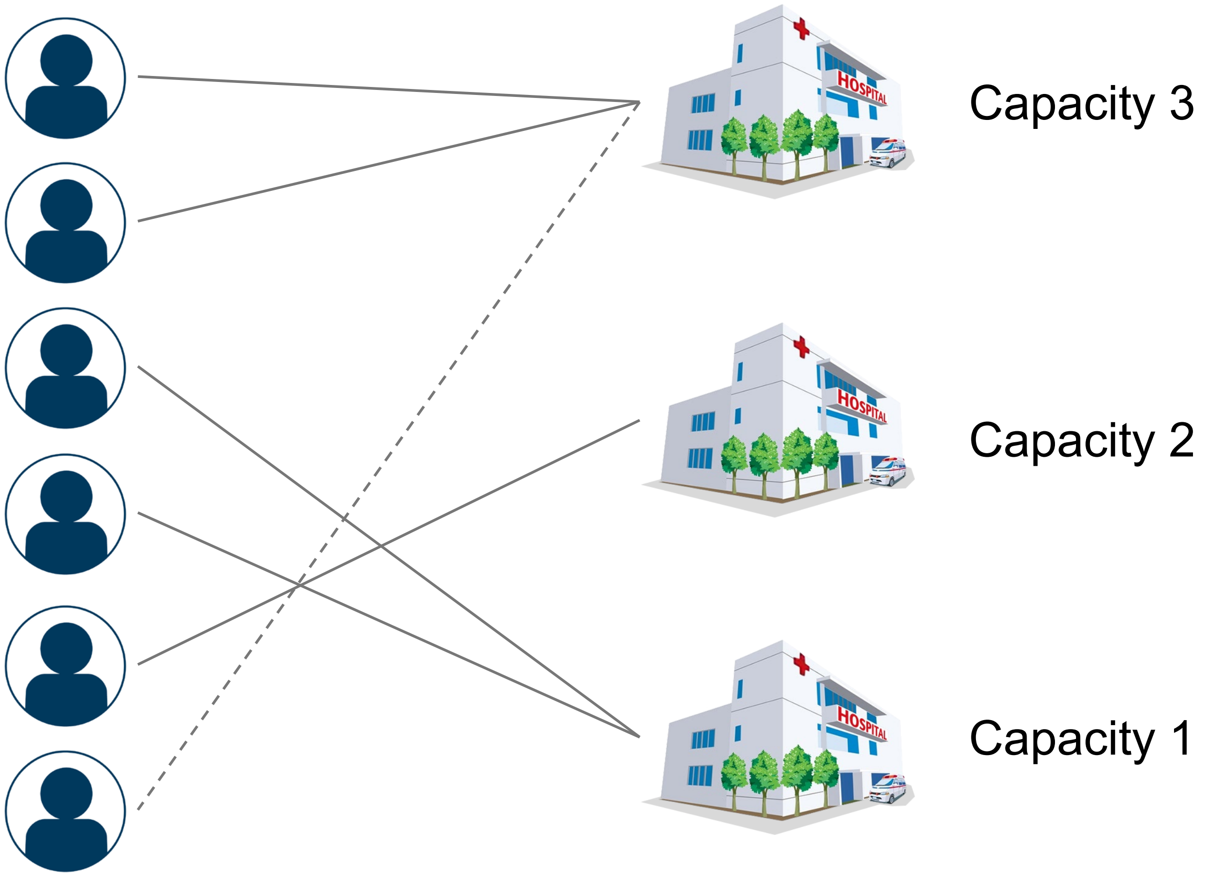
NRMP SIMPLIFIED: A GALE-SHAPLEY APPROACH

Idea: Hospitals propose to as many doctors as they have open spots. Doctors accept/reject as normal. Repeat until done.



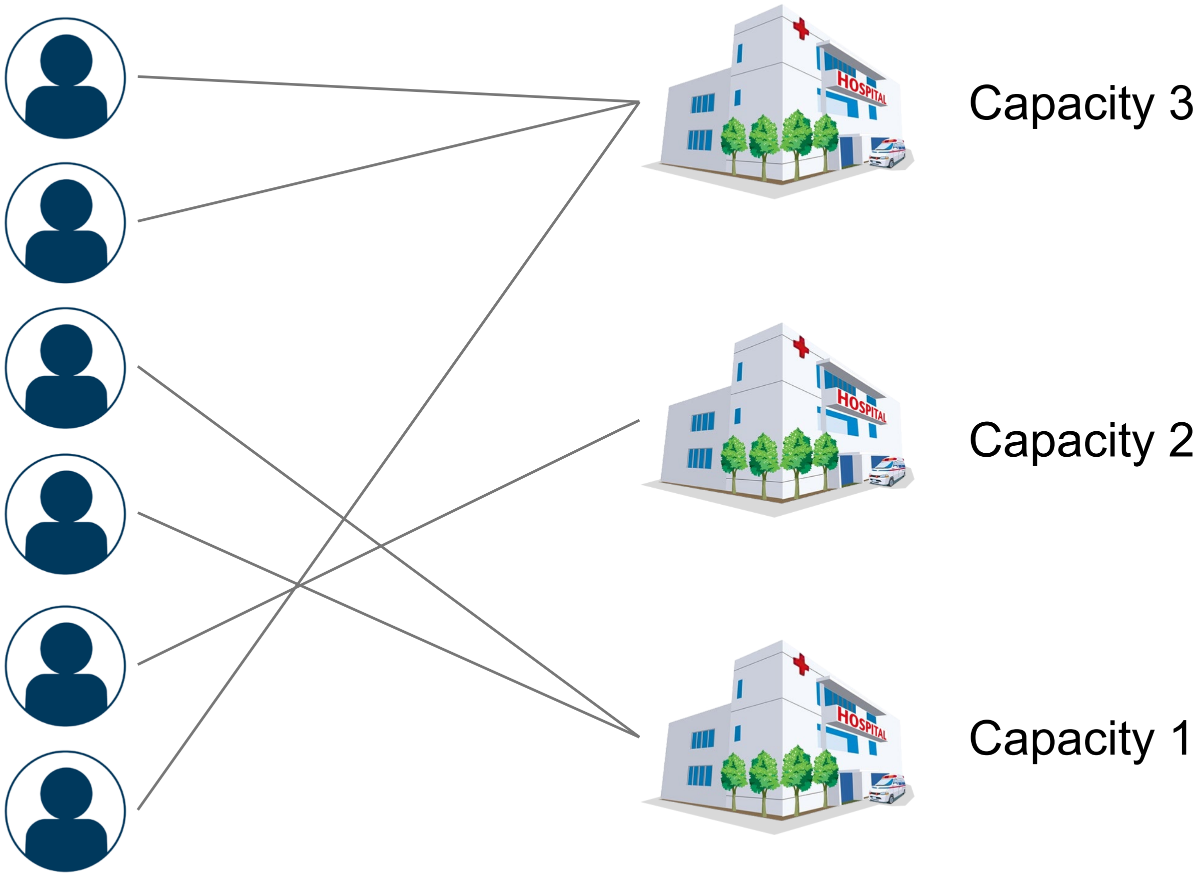
NRMP SIMPLIFIED: A GALE-SHAPLEY APPROACH

Idea: Hospitals propose to as many doctors as they have open spots. Doctors accept/reject as normal. Repeat until done.



NRMP SIMPLIFIED: A GALE-SHAPLEY APPROACH

Idea: Hospitals propose to as many doctors as they have open spots. Doctors accept/reject as normal. Repeat until done.



NRMP SIMPLIFIED: A GALE-SHAPLEY APPROACH

Idea: Hospitals propose to as many doctors as they have open spots. Doctors accept/reject as normal. Repeat until done.

This was the 1951 version of the market

Hospitals proposing: hospital-optimal, doctor-pessimal ✖

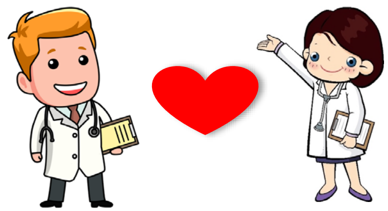
Other idea: Doctors propose to their top hospital. Hospitals accept/reject as normal *up to their capacity*. Repeat until done.

Doctor proposing: doctor-optimal, hospital pessimal ✓

In NRMP: We ended up switching to this version.

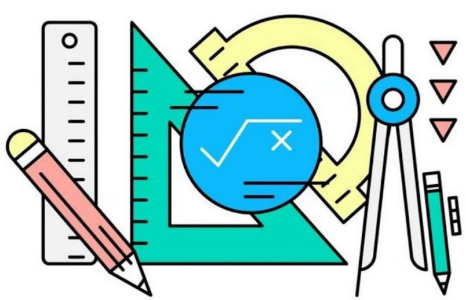
TRUE NRMP: NOT SO SIMPLE

We have four main matching variations:



- **Couples**: pairs who seek “nearby” positions
- **Program types**: specific “1st year programs” are prereqs for specific “2nd year programs”.
    - Applicants match with 2nd year programs AND one of the prereq 1st year programs
    - Capacities between 1st and 2nd year programs are linked
- **Even slots**: programs requiring an even number of residents

Like before, *this breaks the theory*.



HANDLING COUPLES

Couples list ranks as pairs.

 >  > 

Andrew **Bart**	Hospital A Hospital B	Hospital C Hospital C	
Chloe	Hospital A	Hospital B	

Hospital A-1	Andrew	Chloe	
Hospital B-1	Chloe	Bart	
Hospital C-2	Bart	Andrew	

HANDLING COUPLES

Couples list ranks as pairs.

 >  > 

Andrew	Hospital A	Hospital C	
Bart	Hospital B	Hospital C	
Chloe	Hospital A	Hospital B	

Hospital A-1	Andrew	Chloe	
Hospital B-1	Chloe	Bart	
Hospital C-2	Bart	Andrew	

HANDLING COUPLES

Couples list ranks as pairs.

>

>

Andrew **Bart**	Hospital A Hospital B	Hospital C Hospital C	
Chloe	Hospital A	Hospital B	

Hospital A-1	Andrew	Chloe	
Hospital B-1	Chloe	Bart	
Hospital C-2	Bart	Andrew	

HANDLING COUPLES

Couples list ranks as pairs.

 >  > 

Andrew	Hospital A	Hospital C	
Bart	Hospital B	Hospital C	
Chloe	Hospital A	Hospital B	

Hospital A-1	Andrew	Chloe	
Hospital B-1	Chloe	Bart	
Hospital C-2	Bart	Andrew	

HANDLING COUPLES

Couples list ranks as pairs.

 >  > 

Andrew	Hospital A	Hospital C	
Bart	Hospital B	Hospital C	
Chloe	Hospital A	Hospital B	

Hospital A-1	Andrew	Chloe	
Hospital B-1	Chloe	Bart	
Hospital C-2	Bart	Andrew	

HANDLING COUPLES

Couples list ranks as pairs.

 >  > 

Andrew	Hospital A	Hospital C	
Bart	Hospital B	Hospital C	
Chloe	Hospital A	Hospital B	

Hospital A-1	Andrew	Chloe	
Hospital B-1	Chloe	Bart	
Hospital C-2	Bart	Andrew	

HANDLING COUPLES

Couples list ranks as pairs.

 >  > 

Andrew	Hospital A	Hospital C	
Bart	Hospital B	Hospital C	
Chloe	Hospital A	Hospital B	

Hospital A-1	Andrew	Chloe	
Hospital B-1	Chloe	Bart	
Hospital C-2	Bart	Andrew	

HANDLING COUPLES

Couples list ranks as pairs.

 >  > 

Andrew	Hospital A	Hospital C	
Bart	Hospital B	Hospital C	
Chloe	Hospital A	Hospital B	

Hospital A-1	Andrew	Chloe	
Hospital B-1	Chloe	Bart	
Hospital C-2	Bart	Andrew	

HANDLING COUPLES

Couples list ranks as pairs.

 >  > 

Andrew	Hospital A	Hospital C	
Bart	Hospital B	Hospital C	
Chloe	Hospital A	Hospital B	

Hospital A-1	Andrew	Chloe	
Hospital B-1	Chloe	Bart	
Hospital C-2	Bart	Andrew	

HANDLING COUPLES

Couples list ranks as pairs.

 >  > 

Andrew	Hospital A	Hospital C	
Bart	Hospital B	Hospital C	
Chloe	Hospital A	Hospital B	

Hospital A-1	Andrew	Chloe	
Hospital B-1	Chloe	Bart	
Hospital C-2	Bart	Andrew	

HANDLING COUPLES

Couples list ranks as pairs.

 >  > 

Andrew Bart	Hospital A Hospital B	Hospital C Hospital C	
Chloe	Hospital A	Hospital B	

Hospital A-1	Andrew	Chloe	
Hospital B-1	Chloe	Bart	
Hospital C-2	Bart	Andrew	

Is this a **stable matching?**

HANDLING COUPLES

Couples list ranks as pairs.

 >  > 

Andrew	Hospital A	Hospital C	
Bart	Hospital B	Hospital C	
Chloe	Hospital A	Hospital B	

Hospital A-1	Andrew	Chloe	
Hospital B-1	Chloe	Bart	
Hospital C-2	Bart	Andrew	

No! Hospital A and Chloe are ☹

HANDLING COUPLES

Couples list ranks as pairs.

>

>

Andrew	Hospital A	Hospital C	
Bart	Hospital B	Hospital C	
Chloe	Hospital A	Hospital B	

Hospital A-1	Andrew	Chloe	
Hospital B-1	Chloe	Bart	
Hospital C-2	Bart	Andrew	

Sometimes, there are no stable matches ☹ ☹

COMPLEMENTARIES EFFECTS

Simple Markets	Markets with Complementaries
Optimal stable matchings exist	No stable matching may exist. Even if they do, we may not be able to achieve an optimal-resident or hospital solution
Same applicants matched, same positions filled (Rural Hosp'ls Thm)	Different stable matchings may have different applicants and positions filled
When applicant proposing is used a dominant strategy for applicants is to submit true preferences	No algorithm where a dominant strategy for all agents to state true preferences

EXPLORING COMPLEMENTARIES

Are there a lot of variations?

- 4% couples
- 8-12% submit supplemental rank order lists (ROLs)
- 7% of programs have positions that revert to other positions if unfilled
- Thoracic Surgery match is a simple match

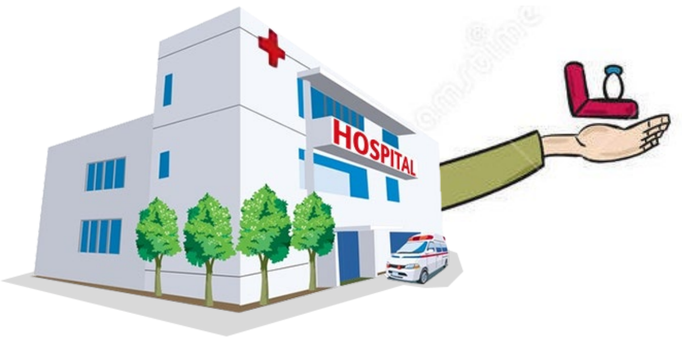
Two (of many) questions to ask:

- Does a program optimal solution make the physicians happy?
- Can applicants act strategically?

THE PREEXISTING ALGORITHM

Phase 1

- Program proposing
- Ignores most variations
- Couples hold onto offers



Phase 2

- Identifies instabilities

Phase 3

- Fixes instabilities one by one
- Sometimes couples propose to programs

When no match variations are present this produces program-optimal stable matching (Thoracic Surgery)

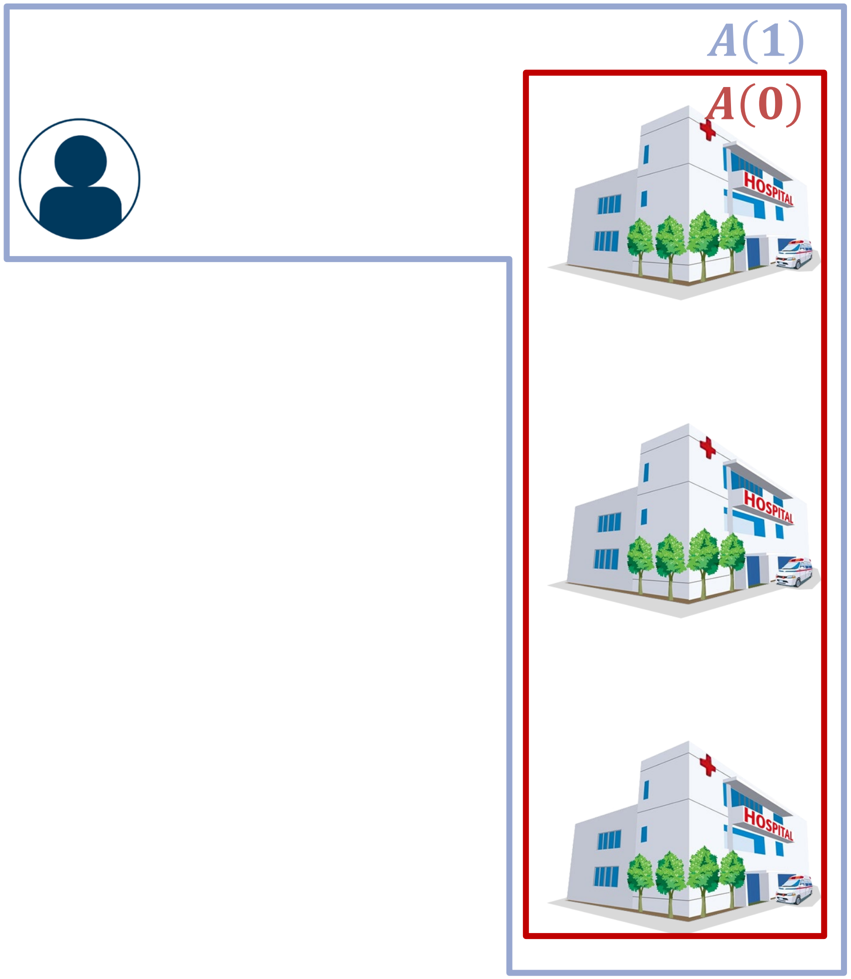
THE NRMP ALGORITHM

$A(0)$ is the set of all hospitals.

THE NRMP ALGORITHM

$A(0)$ is the set of all hospitals.

$A(1)$ is $A(0)$ plus one applicant

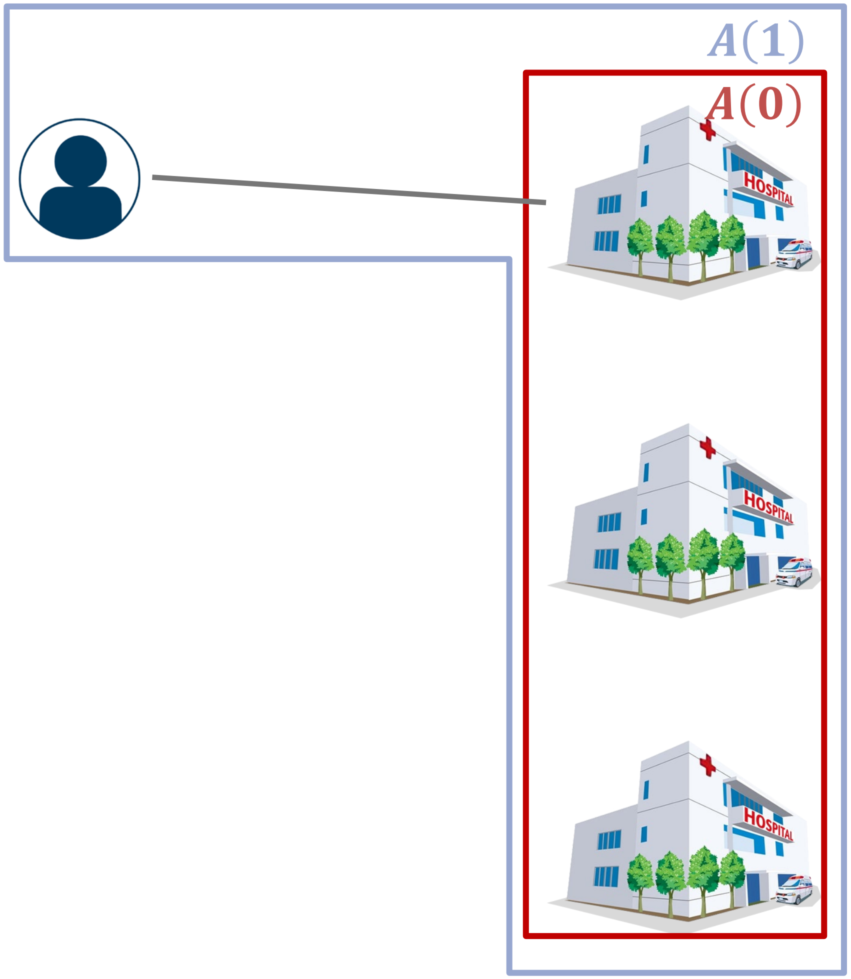


THE NRMP ALGORITHM

$A(0)$ is the set of all hospitals.

$A(1)$ is $A(0)$ plus one applicant

$M(1)$ is a matching found on $A(1)$.



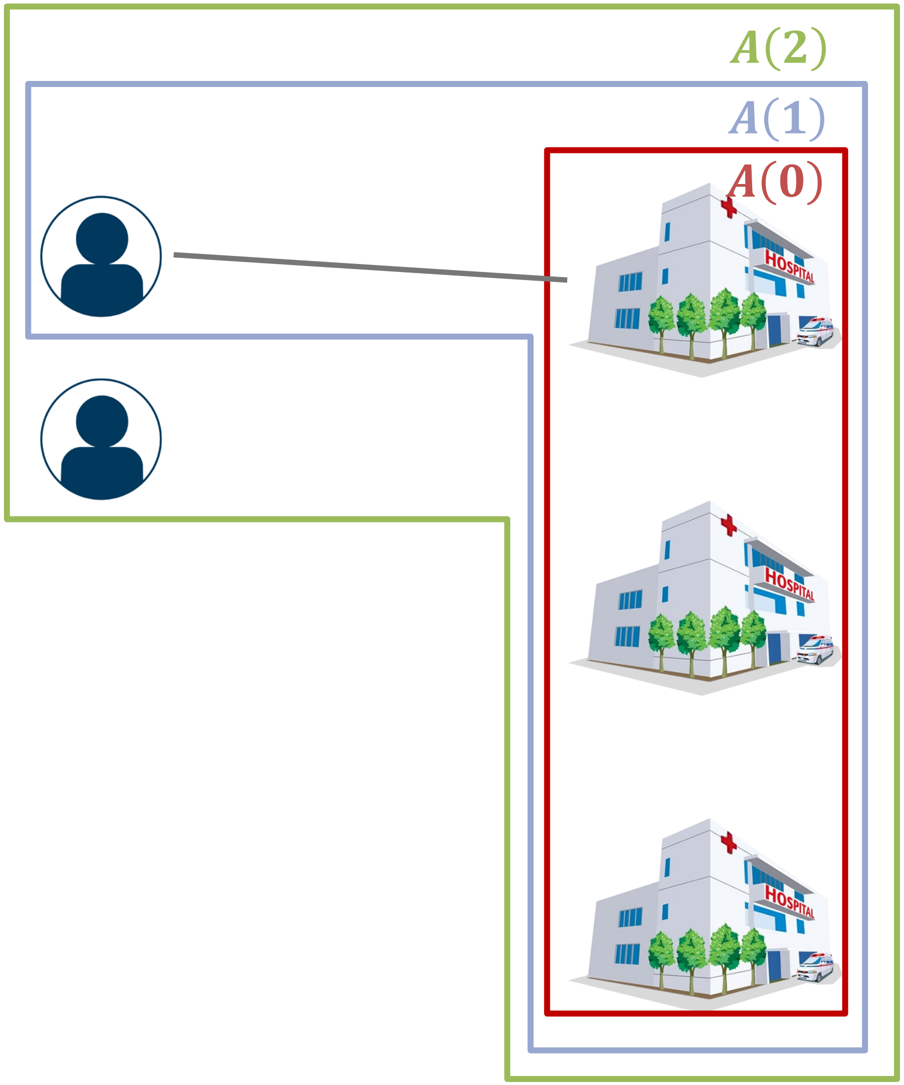
THE NRMP ALGORITHM

$A(0)$ is the set of all hospitals.

$A(1)$ is $A(0)$ plus one applicant

$M(1)$ is a matching found on $A(1)$.

$A(2)$ is $A(1)$ plus one applicant $S(1)$



THE NRMP ALGORITHM

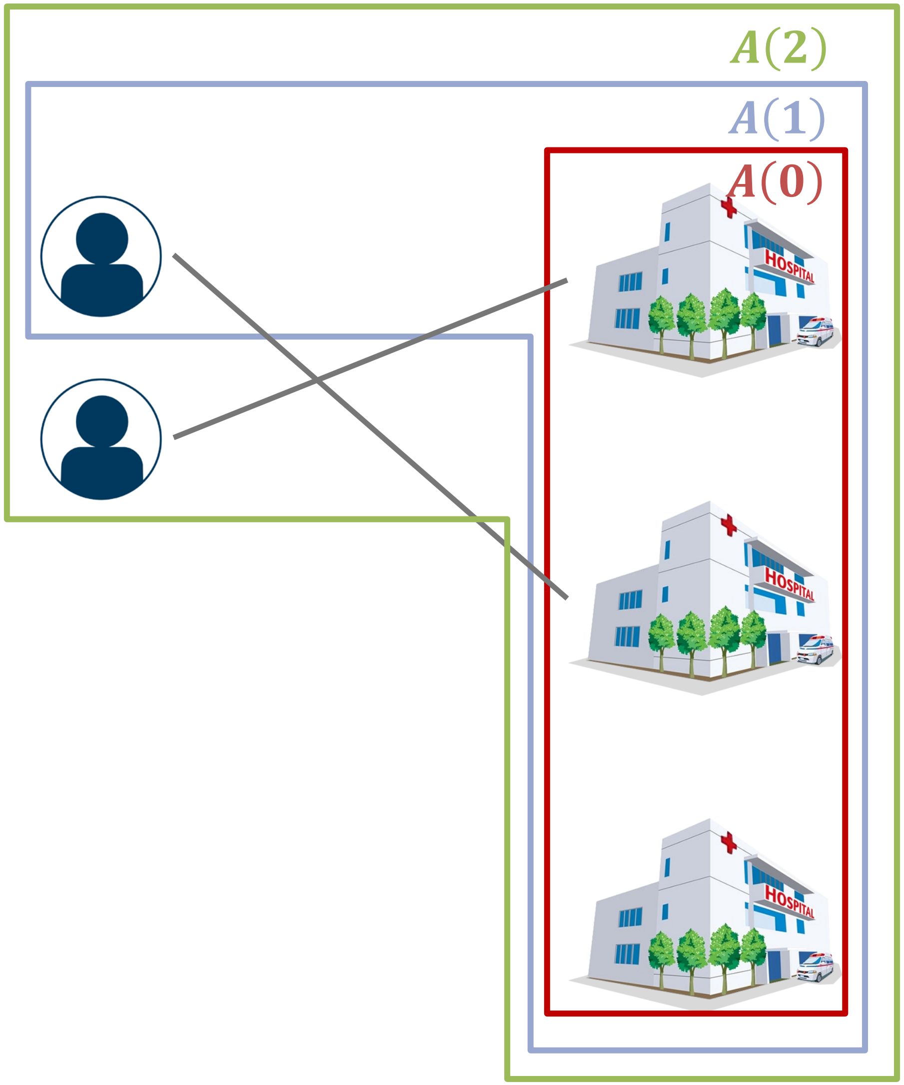
$A(0)$ is the set of all hospitals.

$A(1)$ is $A(0)$ plus one applicant

$M(1)$ is a matching found on $A(1)$.

$A(2)$ is $A(1)$ plus one applicant $S(1)$

$M(2)$ starts as $M(1)$, where $S(2)$ proposes until accepted, and $S(1)$ may have to re-propose.



THE NRMP ALGORITHM

$A(0)$ is the set of all hospitals.

$A(1)$ is $A(0)$ plus one applicant

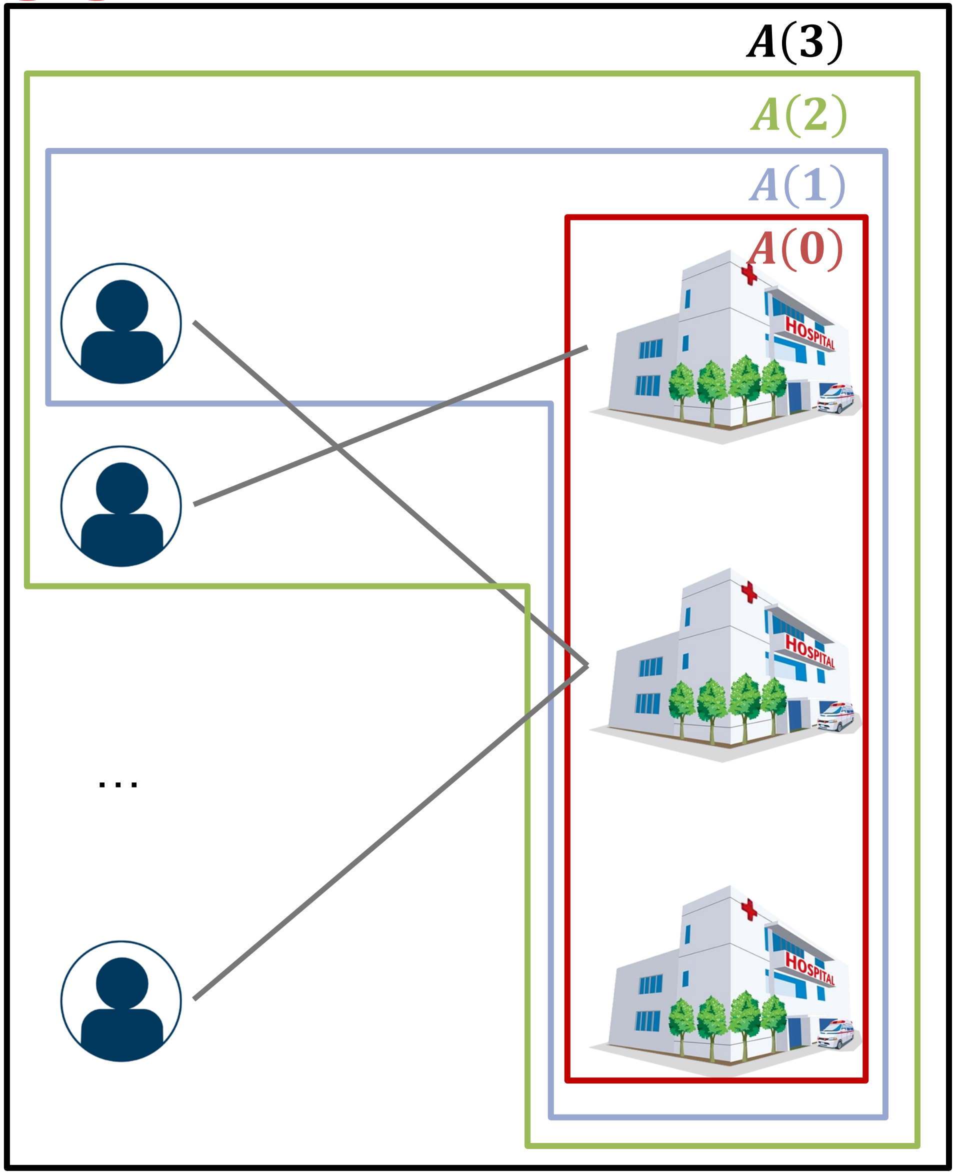
$M(1)$ is a matching found on $A(1)$.

$A(2)$ is $A(1)$ plus one applicant $S(1)$

$M(2)$ starts as $M(1)$, where $S(2)$ proposes until accepted, and $S(1)$ may have to re-propose.

…

$M(k)$ starts as $M(k-1)$, where $S(k)$ proposes until accepted, any displaced people may repropose, and so on.



THE NRMP ALGORITHM: COUPLES AND PREREQS

Happens at the end of each iteration

Couples:

- When a person is displaced, so is their partner. As a couple, they propose down their list.
- A new empty slot gets opened by partner. Add this to “program stack”.

Prerequisite programs:

- If a displaced person loses 2 slots, add slot to program stack.
- If proposer is accepted by such a program, then continue applying to prerequisite programs.
  - This may displace 2 people. Just process them one after the other.

Once applicants are done, remove programs from program stack one by one. Allow applicants with potential instabilities to propose again.

THE NRMP ALGORITHM: EVEN/ODD AND REVERSIONS

Happens at the end of the algorithm

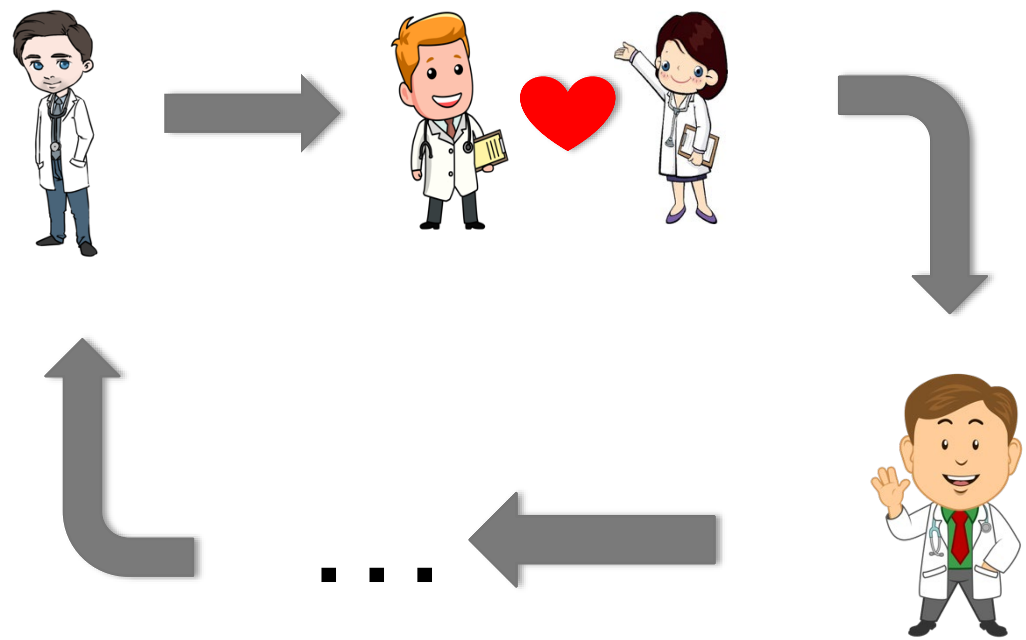
Even/Odd:

- Remove a single applicant as necessary.
- Displaced people can then continue to propose.

Reversions:

- All hospitals decide how many slots to revert for each program.
- Empty slots are added to the program stack.
- We again process the program stack as before.

LOOPS IN THE APPLICANT PROPOSING ALGORITHM



Loops can be detected. Then either:

- We can resolve by rerandomizing processing orders on stacks
- They imply no stability exists. This is rare.

SEQUENCE CHANGES

Ran computational experiments

Differences in matches was extremely small and did not appear to be systematic

Did effect number of loops

- Fewest when couples where introduced last

RESULTS OF THE NEW ALGORITHM

0.1% of applicants affected, 0.5% of programs affected

Most affected applicants preferred new, most affected programs did not

Would be zero by Rural Hosp'ls, still small though

TABLE 2—COMPARISON OF RESULTS BETWEEN ORIGINAL NRMP ALGORITHM AND APPLICANT-PROPOSING ALGORITHM

Result	1987	1993	1994	1995	1996
Applicants:					
Number of applicants affected	20	16	20	14	21
Applicant-proposing result preferred	12	16	11	14	12
Current NRMP result preferred	8	0	9	0	9
U.S. applicants affected	17	9	17	12	18
Independent applicants affected	3	7	3	2	3
Difference in result by rank number					
1 rank	12	11	13	8	8
2 ranks	3	1	4	2	6
3 ranks	2	3	2	2	3
More than 3 ranks	2	1	1	2	3
	(max 9)	(max 4)	(max 5)	(max 6)	(max 6)
New matched	0	0	0	0	1
New unmatched	1	0	0	0	0
Programs:					
Number of programs affected	20	15	23	15	19
Applicant-proposing result preferred	8	0	12	1	10
Current NRMP result preferred	12	15	11	14	9
Difference in result by rank number					
5 or fewer ranks	5	3	9	6	3
6–10 ranks	5	3	3	5	3
11–15 ranks	0	5	1	3	1
More than 15 ranks	9	4	6	0	11
	(max 178)	(max 36)	(max 31)		(max 191)
Programs with new position(s) filled	0	0	2	1	1
Programs with new unfilled position(s)	1	0	2	0	0

IS THE CHANGE WORTH IT?

0.1% of applicants affected

Most of those affected prefer the new algorithm

0.5% of programs affected

Most of those affected prefer the old algorithm

This does not imply the associated change in welfare is small

- Large increase for affected applicants
- Small decrease for the affected programs

STRATEGIC BEHAVIOR OF PARTICIPANTS

TABLE 4—UPPER LIMIT OF THE NUMBER OF APPLICANTS WHO COULD BENEFIT BY TRUNCATING THEIR LISTS AT ONE ABOVE THEIR ORIGINAL MATCH POINT

	Upper limit	
Year	Preexisting NRMP algorithm	Applicant-proposing algorithm
1987	12	0
1993	22	0
1994	13	2
1995	16	2
1996	11	9

STRATEGIC BEHAVIOR OF PROGRAMS

TABLE 5—UPPER LIMIT OF THE NUMBER OF PROGRAMS THAT COULD BENEFIT BY TRUNCATING THEIR LISTS AT ONE ABOVE THE ORIGINAL MATCH POINT

Year	Preexisting NRMP algorithm	Applicant-proposing algorithm
1987	15	27
1993	12	28
1994	15	27
1995	23	36
1996	14	18