New algorithms for one-sided matching

Mark C. Wilson

UMass Amherst
Department of Mathematics and Statistics
2020-03-11

Example

- We have to allocate students to summer research projects. Each student should get a single project. Each project has a single supervisor.

Example

- We have to allocate students to summer research projects. Each student should get a single project. Each project has a single supervisor.
- We can consider this as one-sided matching if supervisors' preferences are ignored, or two-sided matching if they are considered.

Example

- We have to allocate students to summer research projects. Each student should get a single project. Each project has a single supervisor.
- We can consider this as one-sided matching if supervisors' preferences are ignored, or two-sided matching if they are considered.
- If every student has a different first choice project, and supervisors have no preferences, it is easy, but otherwise?

Example

- We have to allocate students to summer research projects. Each student should get a single project. Each project has a single supervisor.
- We can consider this as one-sided matching if supervisors' preferences are ignored, or two-sided matching if they are considered.
- If every student has a different first choice project, and supervisors have no preferences, it is easy, but otherwise?
- The same basic problem (with some variations) occurs in matching interns to hospitals, military staff to bases, children to public schools, volunteer teachers to schools, ...

Resource allocation basics

- We have a set of n agents and a set of m items.

Resource allocation basics

- We have a set of n agents and a set of m items.
- We want to assign a set of items to each agent.

Resource allocation basics

- We have a set of n agents and a set of m items.
- We want to assign a set of items to each agent.
- Sometimes the items have preferences over the agents. They may actually be agents themselves (two-sided matching).

Resource allocation basics

- We have a set of n agents and a set of m items.
- We want to assign a set of items to each agent.
- Sometimes the items have preferences over the agents. They may actually be agents themselves (two-sided matching).
- When $m \neq n$ there are several complications, so we stick to the case $m = n$ today.

Resource allocation basics

- We have a set of n agents and a set of m items.
- We want to assign a set of items to each agent.
- Sometimes the items have preferences over the agents. They may actually be agents themselves (two-sided matching).
- When $m \neq n$ there are several complications, so we stick to the case $m = n$ today.
- We stick to the deterministic case today - random allocations are also very interesting but lead to many subtleties.

Two famous deterministic algorithms

- Serial dictatorship - order agents in some way; each in turn chooses their most preferred object from those remaining.

Two famous deterministic algorithms

- Serial dictatorship - order agents in some way; each in turn chooses their most preferred object from those remaining.
- Boston school choice - allocate as many agents to their first choice as possible, then as many of those remaining to their second, etc, breaking ties according to a fixed order of agents.

Example (Famous one-sided algorithms)

- Suppose agents $1, 2, 3$ have respective preferences $a \succ b \succ c, a \succ b \succ c, b \succ a \succ c$.

Example (Famous one-sided algorithms)

- Suppose agents $1, 2, 3$ have respective preferences $a \succ b \succ c, a \succ b \succ c, b \succ a \succ c$.
- Serial dictatorship with the given order of agents results in the allocation $1 : a, 2 : b, 3 : c$.

Example (Famous one-sided algorithms)

- Suppose agents $1, 2, 3$ have respective preferences $a \succ b \succ c, a \succ b \succ c, b \succ a \succ c$.
- Serial dictatorship with the given order of agents results in the allocation $1 : a, 2 : b, 3 : c$.
- Boston gives $1 : a, 2 : c, 3 : b$.

Properties of solutions

- Efficiency - if we improve some agent's allocation we must make another worse off.

Properties of solutions

- Efficiency - if we improve some agent's allocation we must make another worse off.
- Strategyproofness - no incentive for agents to lie about preferences.

Properties of solutions

- Efficiency - if we improve some agent's allocation we must make another worse off.
- Strategyproofness - no incentive for agents to lie about preferences.
- Fairness - no agent envies another's item.

Properties of solutions

- Efficiency - if we improve some agent's allocation we must make another worse off.
- Strategyproofness - no incentive for agents to lie about preferences.
- Fairness - no agent envies another's item.
- Welfare - a combination of efficiency and fairness, measuring overall happiness of agents.

Properties of algorithms

Algorithm	Fast	Efficient	Strategyproof	Envy-free
SD	✓	✓	✓	✗
Boston	✓	✓	✗	✗

- Boston is easy to manipulate, and SD is clearly “unfair”.

Properties of algorithms

Algorithm	Fast	Efficient	Strategyproof	Envy-free
SD	✓	✓	✓	✗
Boston	✓	✓	✗	✗

- Boston is easy to manipulate, and SD is clearly "unfair".
- No algorithm can be envy-free in every situation.

Properties of algorithms

Algorithm	Fast	Efficient	Strategyproof	Envy-free
SD	✓	✓	✓	✗
Boston	✓	✓	✗	✗

- Boston is easy to manipulate, and SD is clearly "unfair".
- No algorithm can be envy-free in every situation.
- Thus we need to consider tradeoffs, and maybe there are new algorithms that make a good compromise.

Main idea

- There is an inevitable tradeoff between the axiomatic properties.

Main idea

- There is an inevitable tradeoff between the axiomatic properties.
- Each axiom selects an "extreme point of algorithm space".

Main idea

- There is an inevitable tradeoff between the axiomatic properties.
- Each axiom selects an "extreme point of algorithm space".
- My main idea: good algorithms probably exist in the "interior of algorithm space", and we should look for them.

Main idea

- There is an inevitable tradeoff between the axiomatic properties.
- Each axiom selects an "extreme point of algorithm space".
- My main idea: good algorithms probably exist in the "interior of algorithm space", and we should look for them.
- We should search for algorithms that behave "well" on "most" inputs.

New algorithm: Yankee swap

- Inspired by a departmental Christmas party!

New algorithm: Yankee swap

- Inspired by a departmental Christmas party!
- All items are initially unmarked. Once marked, an item remains marked. Agents are given a fixed order.

New algorithm: Yankee swap

- Inspired by a departmental Christmas party!
- All items are initially unmarked. Once marked, an item remains marked. Agents are given a fixed order.
- Play proceeds in rounds. At the beginning of each round, the next unmatched agent takes its top choice item. The previous holder of the item becomes the current agent, and the round continues. No agent may hold a given item more than once per round.

New algorithm: Yankee swap

- Inspired by a departmental Christmas party!
- All items are initially unmarked. Once marked, an item remains marked. Agents are given a fixed order.
- Play proceeds in rounds. At the beginning of each round, the next unmatched agent takes its top choice item. The previous holder of the item becomes the current agent, and the round continues. No agent may hold a given item more than once per round.
- Each round ends when the current agent takes a previously unmarked item.

New algorithm: Yankee swap

- Inspired by a departmental Christmas party!
- All items are initially unmarked. Once marked, an item remains marked. Agents are given a fixed order.
- Play proceeds in rounds. At the beginning of each round, the next unmatched agent takes its top choice item. The previous holder of the item becomes the current agent, and the round continues. No agent may hold a given item more than once per round.
- Each round ends when the current agent takes a previously unmarked item.
- Termination occurs when there are no unmarked items left.

Example (Yankee Swap)

- Suppose agents $1, 2, 3$ have respective preferences $a \succ b \succ c, a \succ b \succ c, b \succ a \succ c$.

Example (Yankee Swap)

- Suppose agents $1, 2, 3$ have respective preferences $a \succ b \succ c, a \succ b \succ c, b \succ a \succ c$.
- In round 1, 1 takes a.

Example (Yankee Swap)

- Suppose agents $1, 2, 3$ have respective preferences $a \succ b \succ c, a \succ b \succ c, b \succ a \succ c$.
- In round 1, 1 takes a.
- In round 2, 2 takes a, then 1 takes b.

Example (Yankee Swap)

- Suppose agents $1, 2, 3$ have respective preferences $a \succ b \succ c, a \succ b \succ c, b \succ a \succ c$.
- In round 1, 1 takes a.
- In round 2, 2 takes a, then 1 takes b.
- In round 3, 3 takes b, 1 takes a, 2 takes b, 3 takes a, 1 takes c.

Example (Yankee Swap)

- Suppose agents $1, 2, 3$ have respective preferences $a \succ b \succ c, a \succ b \succ c, b \succ a \succ c$.
- In round 1, 1 takes a.
- In round 2, 2 takes a, then 1 takes b.
- In round 3, 3 takes b, 1 takes a, 2 takes b, 3 takes a, 1 takes c.
- Final allocation: $1 : c, 2 : b, 3 : a$.

Properties of algorithms

Algorithm	Fast	Efficient	Strategyproof	Envy-free
SD	✓	✓	✓	✗
Boston	✓	✓	✗	✗
YS	✓	✗	✗	✗

- At first sight, YS doesn't seem very competitive. However . . .

Top Trading Cycle (Shapley-Scarf 1974; attr. D. Gale)

- Assume each agent already is matched to an item (the initial endowment).

Top Trading Cycle (Shapley-Scarf 1974; attr. D. Gale)

- Assume each agent already is matched to an item (the initial endowment).
- Each agent points to the agent holding its top choice.

Top Trading Cycle (Shapley-Scarf 1974; attr. D. Gale)

- Assume each agent already is matched to an item (the initial endowment).
- Each agent points to the agent holding its top choice.
- The resulting directed graph must have a cycle (why?). We can reallocate along a cycle to make those agents happy.

Top Trading Cycle (Shapley-Scarf 1974; attr. D. Gale)

- Assume each agent already is matched to an item (the initial endowment).
- Each agent points to the agent holding its top choice.
- The resulting directed graph must have a cycle (why?). We can reallocate along a cycle to make those agents happy.
- Remove the reallocated items and agents, and continue (starting with 2nd preferences) until no more cycles exist.

Top Trading Cycle (Shapley-Scarf 1974; attr. D. Gale)

- Assume each agent already is matched to an item (the initial endowment).
- Each agent points to the agent holding its top choice.
- The resulting directed graph must have a cycle (why?). We can reallocate along a cycle to make those agents happy.
- Remove the reallocated items and agents, and continue (starting with 2nd preferences) until no more cycles exist.
- This runs in polynomial time, and yields an efficient allocation.

Properties of algorithms

Algorithm	Fast	Efficient	Strategyproof	Envy-free
SD	✓	✓	✓	✗
Boston	✓	✓	✗	✗
YS	✓	✗	✗	✗
YS+TTC	✓	✓	✗	✗

- YS followed by TTC is now efficient. Experiments show it outperforms SD (and usually Boston) in welfare and envy. It seems hard to manipulate strategically and feels “fairer” because if an item is taken from us, we can steal it back later.

Two-sided matching: Gale-Shapley

- Each agent proposes to its items in descending order of preference.

Two-sided matching: Gale-Shapley

- Each agent proposes to its items in descending order of preference.
- If an item is unmatched, it accepts the proposal tentatively.

Two-sided matching: Gale-Shapley

- Each agent proposes to its items in descending order of preference.
- If an item is unmatched, it accepts the proposal tentatively.
- If a better agent (more preferred by the item) comes along, the item rejects the current one and accepts the new one.

Two-sided matching: Gale-Shapley

- Each agent proposes to its items in descending order of preference.
- If an item is unmatched, it accepts the proposal tentatively.
- If a better agent (more preferred by the item) comes along, the item rejects the current one and accepts the new one.
- Each agent may propose at most once to each item (so rejections are permanent), which guarantees termination.

Connecting one- and two-sided matching

- We can give fictitious preferences to the items, and run Gale-Shapley with these to get an allocation.

Connecting one- and two-sided matching

- We can give fictitious preferences to the items, and run Gale-Shapley with these to get an allocation.
- To get more interesting algorithms, we can modify the items' preferences dynamically based on the proposals received.

Connecting one- and two-sided matching

- We can give fictitious preferences to the items, and run Gale-Shapley with these to get an allocation.
- To get more interesting algorithms, we can modify the items' preferences dynamically based on the proposals received.
- Main ideas: accept first (prefer agents in order they have proposed) and accept last (prefer them in reverse order). There may be other good ideas.

Connecting one- and two-sided matching

- We can give fictitious preferences to the items, and run Gale-Shapley with these to get an allocation.
- To get more interesting algorithms, we can modify the items' preferences dynamically based on the proposals received.
- Main ideas: accept first (prefer agents in order they have proposed) and accept last (prefer them in reverse order). There may be other good ideas.
- Yankee Swap uses accept-last, since the current agent trying to steal an item is always accepted. SD and Boston use accept-first.

Memory

- We can assume the history of proposals is forgotten at the end of each round (each time a new item is matched).

Memory

- We can assume the history of proposals is forgotten at the end of each round (each time a new item is matched).
- This allows another chance for agents to propose to items they have already been rejected by, and intuitively may lead to higher welfare by avoiding local maxima.

Memory

- We can assume the history of proposals is forgotten at the end of each round (each time a new item is matched).
- This allows another chance for agents to propose to items they have already been rejected by, and intuitively may lead to higher welfare by avoiding local maxima.
- Serial Dictatorship and Boston have permanent memory, Yankee Swap has temporary memory.

Order of proposals

- For standard Gale-Shapley the order of proposals does not matter.

Order of proposals

- For standard Gale-Shapley the order of proposals does not matter.
- In our dynamic preference setup it does. There are two obvious choices:

Order of proposals

- For standard Gale-Shapley the order of proposals does not matter.
- In our dynamic preference setup it does. There are two obvious choices:
  - Queue-based: first in, first out;

Order of proposals

- For standard Gale-Shapley the order of proposals does not matter.
- In our dynamic preference setup it does. There are two obvious choices:
  - Queue-based: first in, first out;
  - Stack-based: last in, first out.

Order of proposals

- For standard Gale-Shapley the order of proposals does not matter.
- In our dynamic preference setup it does. There are two obvious choices:
  - Queue-based: first in, first out;
  - Stack-based: last in, first out.
- Serial Dictatorship uses stack, Boston uses queue, Yankee Swap uses stack.

Old algorithms in this framework

- Serial Dictatorship: permanent memory, stack, accept-first;

This gives a motivation for the consideration of Yankee Swap, which seemed somewhat arbitrary before.

Old algorithms in this framework

- Serial Dictatorship: permanent memory, stack, accept-first;
- Boston: permanent memory, queue, accept-last;

This gives a motivation for the consideration of Yankee Swap, which seemed somewhat arbitrary before.

Old algorithms in this framework

- Serial Dictatorship: permanent memory, stack, accept-first;
- Boston: permanent memory, queue, accept-last;
- Yankee Swap: temporary memory, stack, accept-last.

This gives a motivation for the consideration of Yankee Swap, which seemed somewhat arbitrary before.

What about the other algorithms?

- There are 2 choices for memory/no memory, 2 for queue/stack, 2 for accept first/accept last.

What about the other algorithms?

- There are 2 choices for memory/no memory, 2 for queue/stack, 2 for accept first/accept last.
- We can also append TTC to any of the algorithms.

What about the other algorithms?

- There are 2 choices for memory/no memory, 2 for queue/stack, 2 for accept first/accept last.
- We can also append TTC to any of the algorithms.
- We can show that all the Accept-First ones are efficient, and none of the Accept-Last ones are.

What about the other algorithms?

- There are 2 choices for memory/no memory, 2 for queue/stack, 2 for accept first/accept last.
- We can also append TTC to any of the algorithms.
- We can show that all the Accept-First ones are efficient, and none of the Accept-Last ones are.
- This yields $2^3 + 4 = 12$ algorithms, so there are 9 more to discuss.

Behavior of algorithms on standard profile: 3 agents $a \succ b \succ c \succ d$, 1 agent $b \succ a \succ c \succ d$

Algorithm	Output matching	Number of proposals
PFS	1:a, 2:b, 3:c, 4:d	10
PFQ	1:a, 2:c, 3:d, 4:b	9
PLS	1:d, 2:c, 3:a, 4:b	9
PLQ	1:d, 2:c, 3:b, 4:a	10
TFS	1:d, 2:a, 3:c, 4:b	18
TFQ	1:a, 2:b, 3:d, 4:c	33
TLS	1:b, 2:a, 3:d, 4:c	18
TLQ	1:a, 2:b, 3:d, 4:c	21

A new fairness criterion

- All our algorithms require a predetermined order of agents. We can choose this randomly, and therefore achieve fairness ex ante, but sometimes ex post fairness is what we need.

A new fairness criterion

- All our algorithms require a predetermined order of agents. We can choose this randomly, and therefore achieve fairness ex ante, but sometimes ex post fairness is what we need.
- Also, randomness may not be acceptable in some situations.

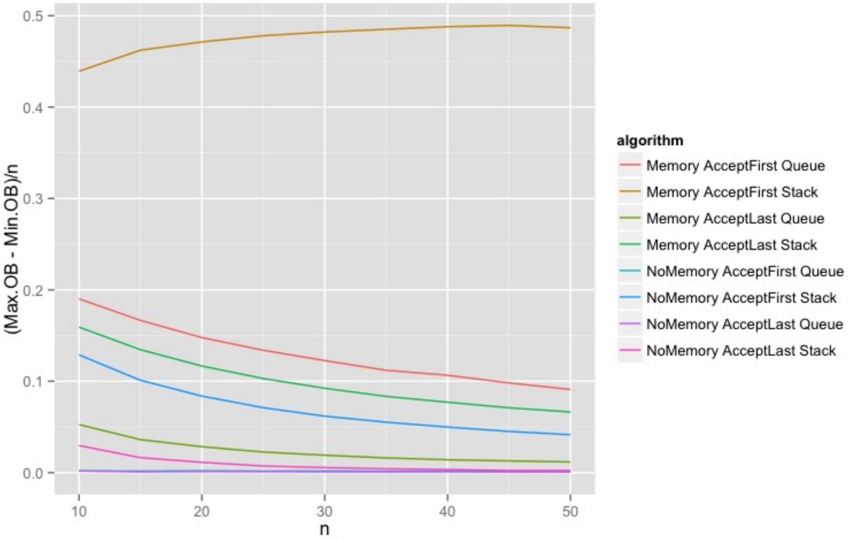
A new fairness criterion

- All our algorithms require a predetermined order of agents. We can choose this randomly, and therefore achieve fairness ex ante, but sometimes ex post fairness is what we need.
- Also, randomness may not be acceptable in some situations.
- We call an algorithm order-fair if for all i, j the expected rank of the item allocated to the ith agent is the same as for the jth agent, as we average over all preference profiles.

A new fairness criterion

- All our algorithms require a predetermined order of agents. We can choose this randomly, and therefore achieve fairness ex ante, but sometimes ex post fairness is what we need.
- Also, randomness may not be acceptable in some situations.
- We call an algorithm order-fair if for all i, j the expected rank of the item allocated to the ith agent is the same as for the jth agent, as we average over all preference profiles.
- SD is very far from order-fair, since the last agent is much worse off than the first one. Boston is better, but still heavily biased toward early agents.

Normalized order bias of our 8 basic algorithms



Some observations about the new algorithms

- None of the new algorithms clearly dominate the old in efficiency, fairness, or welfare, but in some situations they do much better.

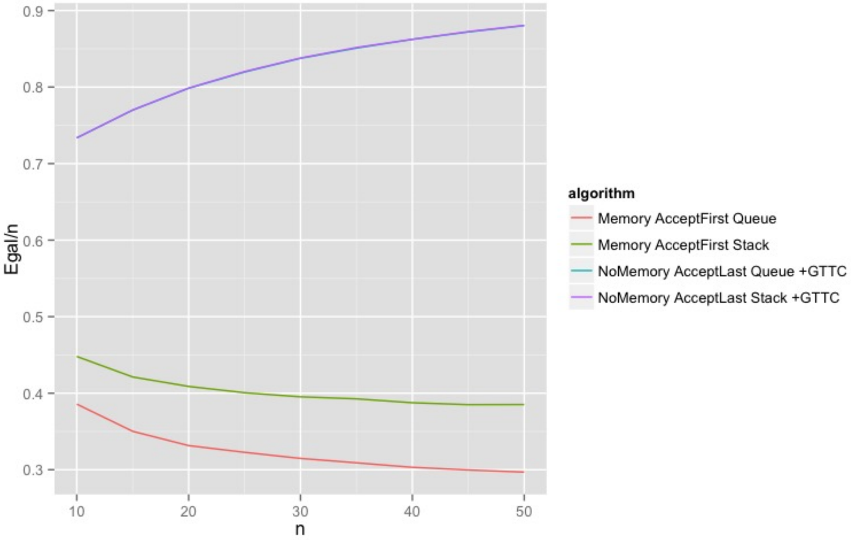
Some observations about the new algorithms

- None of the new algorithms clearly dominate the old in efficiency, fairness, or welfare, but in some situations they do much better.
- The temporary-memory equivalent of SD proceeds like SD in each round. It favours the last agent over all the others, and may be appropriate for situations in which a single agent should be privileged.

Some observations about the new algorithms

- None of the new algorithms clearly dominate the old in efficiency, fairness, or welfare, but in some situations they do much better.
- The temporary-memory equivalent of SD proceeds like SD in each round. It favours the last agent over all the others, and may be appropriate for situations in which a single agent should be privileged.
- Interestingly, the queue-based algorithms are much fairer than the stack-based ones, and the queue analogue of Yankee Swap has amazingly small bias.

Egalitarian welfare of selected algorithms



Where to from here?

- For random preferences we should be able to study performance of some of these algorithms analytically (e.g. Boston), and the rest we can do by simulation.

Where to from here?

- For random preferences we should be able to study performance of some of these algorithms analytically (e.g. Boston), and the rest we can do by simulation.
- Choosing realistic artificial preferences for simulation is tricky. We can use models based on agents imitating others.

Where to from here?

- For random preferences we should be able to study performance of some of these algorithms analytically (e.g. Boston), and the rest we can do by simulation.
- Choosing realistic artificial preferences for simulation is tricky. We can use models based on agents imitating others.
- We can create random allocation algorithms by simply randomizing the agent order, making it easier to avoid envy.

Where to from here?

- For random preferences we should be able to study performance of some of these algorithms analytically (e.g. Boston), and the rest we can do by simulation.
- Choosing realistic artificial preferences for simulation is tricky. We can use models based on agents imitating others.
- We can create random allocation algorithms by simply randomizing the agent order, making it easier to avoid envy.
- There is still much that is unexplored. Who knows what other algorithms are out there?