Fair Housing Allocation

Beth Richardson
Tiffany Tillett
Porter Perry
Javier Palomares

Housing Allocation Overview

Assignment using Top Trading Cycles

Produces “core” assignment- no one wants to trade houses.

Consider 4 agents with the following preference ordering over each other’s houses:

$\succ_1$: $2 \succ 3 \succ 1 \succ 4$

$\succ_2$: $1 \succ 3 \succ 2 \succ 4$

$\succ_3$: $1 \succ 2 \succ 3 \succ 4$

$\succ_4$: $2 \succ 3 \succ 4 \succ 1$

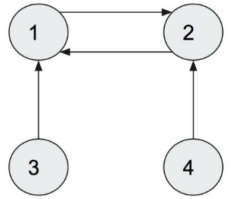
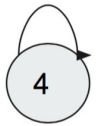


Figure 7.3: remove (1,2) and (2,1)

Figure 7.4: remove (3,3)



Bias in the Trading Cycles Algorithm

Algorithm does not try to find a fair assignment.

Uses the initial ownership of the houses and simply tries to find an assignment that does not lead to any agents wanting to swap houses.

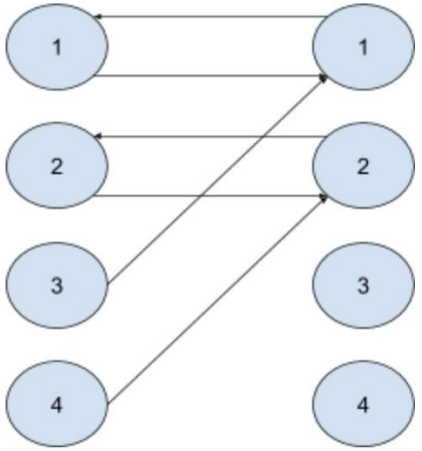
This can lead to some agents having an unfair advantage.

Bias Continued

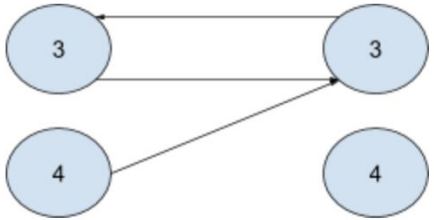
Top Trading Cycle leaves Agent 4 with House 4 (last choice).

An assignment where 2 agents get their 1st choice, 1 agent gets their 2nd choice and the final agent gets their last choice is hard to label as fair.

Initial Graph



After Step 1 (removed cycles on 1 and 2)



After Step 2 (removed cycle on 3)

Our Goal

We provide a mechanism to solve for and identify a housing allocation that is *fair* to the agents involved in the algorithm.

Fairness for us is defined as:

“The majority of agents are assigned to a house with a rank close to the same rank house that other agents have been assigned.”

In other words, there are no real “winners” or “losers” in our model.

Our Model

In order to experiment with multiple algorithm strategies, we modeled the housing assignment problem with a set of Initialization and Matching strategies that could easily be swapped and combined to solve the problem.

Initialization Strategy - provides an interface to set initial house ownership. For example: *No Owners*, *Middle*, *Owners*

Matching Strategy - provides technique to solve for the final house ownership (providing a method for breaking ties). For example, *Poorest Agen*t *Wins* or *Fair Matching*

Swapping strategies

```
Housing(String filename, InitializationStrategy init, MatchingStrategy match, List<Agent> agentPrior) {
    n = ParseFile(filename); // will init agents as well since it's not dependent on strategy
    initializationStrategy = init;
    matchingStrategy = match;
    this.agentPriority = agentPrior;
    initialize();
}

void initialize() {
    owners = initializationStrategy.initializeOwners(n, agents);
}

Map<House, Agent> solve() {
    return this.matchingStrategy.findMatching(agents, owners, houses, agentPriority);
}

void setAgentPriority(List<Agent> agentPriority) {
    this.agentPriority = agentPriority;
}
```

Selecting Strategies

```
class NoOwnersDefaultMatchingHousing extends Housing {

    NoOwnersDefaultMatchingHousing(String filename) {
        super(filename, new NoOwnersInitialization(), new DefaultMatching(), agentPrior: null);
    }

}
```

Fairness Definition

Define fairness score = $\sum_i$ | $rank_i(H)$ - average($rank_j(H)$) |

Lower score is more fair.

Best possible score when all agents are assigned the same housing rank.

How to assign

Can’t assign the same housing rank to every agent- may lead to conflicts where one house is assigned to multiple agents.

Instead, find core allocation that minimizes fairness score.

Algorithm - Inputs

N = number of Agents and Houses

Preferences = Data structure containing the strict house preferences of each Agent

Algorithm - Core Allocation Initialization

AgentPriority = Input list of Agents ordered by tie-breaking priority

Unmatched = Set of agents who are not yet matched (initialized to all Agents)

Wish = Data structure containing Agent's most preferred unassigned house

Conflicts = data structure containing Agents whose current wishes are the same house as another agent

Owners = data structure containing matched agents to their matched houses

Algorithm - Core Allocation Psuedocode

1. Initialize *Wish* for all agents to their top house choice
2. Check if there are conflicts
3. While (conflicts exist)
   3.1. For each conflict:
      3.1.1. Break tie by assigning *winner* as *agent* with highest *AgentPriority*
      3.1.2. Increment *Wish for all agents who are not the winner*
   3.2. Update conflicts
4. End While
5. Set Owners based on current value of Wish
6. Return Owners

Algorithm - Fair House Selection

1. Generate set of *AgentPrioity* list permutations
2. For each *AgentPrioirty* list
   2.1. Generate Owner Core Allocation given *AgentPriority*
   2.2. Add resultant Owners to set of all Allocations
3. For each Owner Allocation
   3.1. Calculate fairness score
4. Report Owner allocation with lowest (best) fairness score.

Algorithm Execution

Agent	Pref 1	Pref 2	Pref 3	Pref 4
1	3	2	1	4
2	1	2	3	4
3	4	2	1	3
4	1	3	2	4

Conflicts:

House	Agents
1	2, 4

Agent priority: 1, 4, 2, 3

The current proposal is highlighted in green. We use the agent priority to break ties.

Agent 4 has higher priority than Agent 2, so Agent 2 moves on to their next preferred house.

Algorithm Execution

Agent	Pref 1	Pref 2	Pref 3	Pref 4
1	3	2	1	4
2	1	2	3	4
3	4	2	1	3
4	1	3	2	4

Conflicts:

House	Agents

Agent priority: 1, 4, 2, 3

We have no more conflicts, so this is a valid allocation.

Algorithm Execution

If I run this algorithm with all possible priority orderings, I get three possible allocations:

A	1	2	3	4
1	3	2	1	4
2	1	2	3	4
3	4	2	1	3
4	1	3	2	4

Avg rank = (1 + 2 + 1 + 1) / 4 = 1.25

Fairness score = .25 + .75 + .25 + .25 = 1.5

A	1	2	3	4
1	3	2	1	4
2	1	2	3	4
3	4	2	1	3
4	1	3	2	4

Avg rank = (1 + 1 + 1 + 3) / 4 = 1.5

Fairness score = .5 + .5 + .5 + 1.5 = 3

A	1	2	3	4
1	3	2	1	4
2	1	2	3	4
3	4	2	1	3
4	1	3	2	4

Avg rank = (2 + 1 + 1 + 2) / 4 = 1.5

Fairness score = .5 + .5 + .5 + .5 = 2

We can see that option 1 has the lowest fairness score, so this will be the final allocation.

Comparison with TTC

TTC

A	1	2	3	4
1	1	2	3	4
2	2	4	3	1
3	1	3	4	2
4	2	3	1	4

Avg rank = (1 + 1 + 2 + 4) / 4 = 2

Fairness score = 1 + 1 + 0 + 2 = 4

Fair Housing Algorithm

A	1	2	3	4
1	1	2	3	4
2	2	4	3	1
3	1	3	4	2
4	2	3	1	4

Avg rank = (2 + 2 + 1 + 2) / 4 = 1.75

Fairness score = .25 + .25 + .75 + .25 = 1.5

The Fair Housing Algorithm is both more socially optimal and more fair than the TTC Algorithm.

Future Work

With additional time:

1. We could make our algorithm more efficient which would enable us to try larger examples.
2. We could explore the tradeoffs between fairness and optimality.
3. We could also experiment more with our calculation of fairness score.
4. We could attempt to solve this problem as a linear program