CS 70 Discrete Mathematics and Probability Theory
Fall 2024 Rao, Hug

HW 02

Due: Saturday, 9/13, 4:00 PM
Grace period until Saturday, 9/13, 6:00 PM

Sundry

Before you start writing your final homework submission, state briefly how you worked on it. Who else did you work with? List names and email addresses. (In case of homework party, you can just describe the group.)

Solution: I worked with Lawrence Rhee (lawrencejrhee@berkeley.edu). We discussed our approaches to each problem and asked each other questions.

1 Airport

Note 3

Suppose that there are $2n+1$ airports, where n is a positive integer. The distances between any two airports are all different. For each airport, exactly one airplane departs from it and is destined for the closest airport. Prove by induction that there is an airport which has no airplanes destined for it.

Solution:

Proof. (Base) Let $n = 1$. We have 3 airports. For any configuration of distances, there will be a smallest distance d among the finite set of distinct airport distances. For the 2 airports being connected by this smallest distance, each will send airplanes to each other because it is the closest airport for each of them. The single airport left that has not sent an airplane yet will end up with no planes destined to it because it can't send its airplane to itself.

(Hypothesis) Suppose for $2k+1$ airports there always exists an airport with no planes destined to it.

(Inductive step) Suppose we have $2k+3$ airports. Similar to the logic we employed in the base case, the 2 airports connected by the smallest distance in the set of distances will send their planes to each other. From here on out, there are 2 cases:

Case 1: as the airports send their planes to the airports, one of the airplanes end up in one of the 2 initially matched airports. Then, an airport has 2 airplanes going into it. Suppose every airport has at least 1 plane going into it at the end of the process. Then we will end up with at least $2k+4$ airplanes, which is more than the number of airports we have. This is a contradiction; therefore, in this case, there will be some airport with no planes destined to it.

Case 2: as the airports send their planes to the airports, none of the airplanes end up in one of the 2 initially matched airports. Then we can completely ignore the 2 matched airports. Their distances to the other airports don't matter anymore and we can simply delete the 2 airports and their distances from the problem. Then we end up with some $2k+1$ instance. Because in every $2k+1$ instance an airport has no airplanes destined to it, so will $2k+3$. □

2 Grid Induction

Note 3

Pacman is walking on an infinite 2D grid. He starts at some location $(i, j) \in \mathbb{N}^2$ in the first quadrant, and is constrained to stay in the first quadrant (say, by walls along the x and y axes).

Every second he does one of the following (if possible):

(i) Walk one step down, to $(i, j-1)$.

(ii) Walk one step left, to $(i-1, j)$.

For example, if he is at $(5,0)$, his only option is to walk left to $(4,0)$; if Pacman is instead at $(3,2)$, he could walk either to $(2,2)$ or $(3,1)$.

Prove by induction that no matter how he walks, he will always reach $(0,0)$ in finite time.

(*Hint*: Try starting Pacman at a few small points like $(2,1)$ and looking all the different paths he could take to reach $(0,0)$. Do you notice a pattern in the number of steps he takes? Try to use this to strengthen the inductive hypothesis.)

Solution: If Pacman starts on location $(i, j) \in \mathbb{N}^2$, he will always land on $(0,0)$ on the $(i+j)^{th}$ second. Therefore, he will always reach $(0,0)$ in finite time.

Proof. (Base case) If Pacman starts on $(0,0)$ he will always land on $(0,0)$ on the $(0+0) = 0^{th}$ second.

(Inductive hypothesis) If Pacman starts on any location $x, y \in \mathbb{N}$ where $x \leq i \wedge y \leq j \wedge (x,y) \neq (i,j)$, he will always land on $(0,0)$ on the $(x+y)^{th}$ second.

(Inductive step) If Pacman starts on (i, j), on the next second he will always land on either $(i, j-1)$ or $(i-1, j)$. Because of the hypothesis, it will always take Pacman $i+j-1$ seconds to reach $(0,0)$ after the first second. Therefore, it will always take Pacman a total of $i+j-1+1 = i+j$ seconds to reach $(0,0)$. □

3 Universal Preference

Note 4

Suppose that preferences in a stable matching instance are universal: all n jobs share the preferences $C_1 > C_2 > \cdots > C_n$ and all candidates share the preferences $J_1 > J_2 > \cdots > J_n$.

(a) What pairing do we get from running the algorithm with jobs proposing? Can you prove this happens for all n?

(b) What pairing do we get from running the algorithm with candidates proposing?

(c) What does this tell us about the number of stable pairings?

Solution:

(a) For simplicity, we will reorder the indexes in the problem statement so that the shared preferences are:

$$C_n > C_{n-1} > \cdots > C_1$$

and

$$J_n > J_{n-1} > \cdots > J_1.$$

For the given restriction, the pairing we get from running the algorithm with jobs proposing will be

$$(C_1, J_1), (C_2, J_2), \ldots, (C_n, J_n)$$

over all n.

Proof. We prove our claim via induction.

(Base case) Let $n = 1$. Then, there is only 1 matching: (C_1, J_1) that can happen (and it must happen). Therefore, the claim holds.

(Hypothesis) Let the resulting pairing for $n = k$ be

$$(C_1, J_1), (C_2, J_2), \ldots, (C_k, J_k).$$

(Inductive step) In the case of $n = k+1$, we have a new candidate C_{k+1} and new job J_{k+1} so that for all jobs C_{k+1} is more preferred than C_k and for all candidates J_{k+1} is more preferred than J_k. Then, on the first day, all the jobs will give an offer to C_{k+1} since that is their most preferred candidate. C_{k+1} will reject all the offers except for the one from J_{k+1} since that is their most preferred job. On any sequential day, no job will be ever offered to C_{k+1} except for the one from J_{k+1} because C_{k+1} will be crossed from their lists after the first day. So, after the first day, C_{k+1} is permanently paired with J_{k+1} and we can and will remove them from our instance. We are left with the problem of $n = k$ exactly. The algorithm will therefore output

$$(C_1, J_1), (C_2, J_2), \ldots, (C_{k+1}, J_{k+1}).$$

□

(b) By symmetry, the pairing we get from running the algorithm with candidates proposing will be the exact same result.

(c) This tells us there is only 1 stable pairing in this instance. This is because the job optimal (and also candidate pessimal) pairing (a) is equal to the candidate optimal pairing (and also job pessimal) pairing (b). There are no "in between" matchings that are neither optimal or pessimal for any party. Think of it as: $x \leq y$ and $x \geq y$ are both true, so x must equal y and there is thus only 1 possible stable matching.

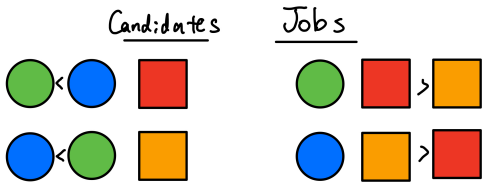
4 Pairing Up

Note 4

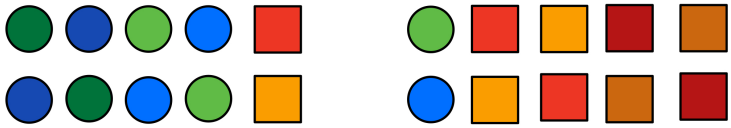
Prove that for every even $n \geq 2$, there exists an instance of the stable matching problem with n jobs and n candidates such that the instance has at least $2^{n/2}$ distinct stable matchings.

Solution:

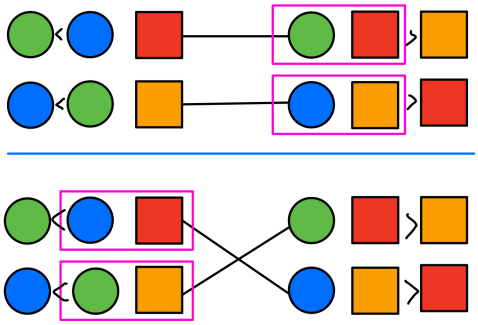
Proof. Let's define a "2 pair" over a group of 2 candidates and 2 jobs as the following entity consisting of the candidates, their preferences, the jobs, and their preferences:



Included in the definition of a "2 pair" is the rule that if we add a "2 pair" to an existing instance where there are already other candidates and jobs, we will make them all less preferrable then those in the group.



We can see there are 2 distinct matchings within a "2 pair" regardless of how many other candidates and jobs also exist and their configurations.



In the top matching, the 2 jobs in the "2 pair" got their most preferred candidate so there are no rogue couples involving any job or candidate in the "2 pair". Symmetrically, in the bottom matching, all candidates got their most preferred job, so there are no rogue couples involving any job or candidate in the "2 pair".

For positive even integer n, if we had an instance made up of only $n/2$ "2 pairs", we would have at least $2^{n/2}$ distinct stable matchings.

□

5 Optimal Candidates

Note 4

In the notes, we proved that the propose-and-reject algorithm always outputs the job-optimal pairing. However, we never explicitly showed why it is guaranteed that putting every job with its optimal candidate results in a pairing at all. Prove by contradiction that no two jobs can have the same optimal candidate. (Note: your proof should not rely on the fact that the propose-and-reject algorithm outputs the job-optimal pairing.)

Solution: Recall that a job's *optimal candidate* is the most preferrable candidate the job can get matched with out of all possible stable matchings.

Proof. Suppose that 2 jobs J and J^* have the same optimal candidate C. WLOG (without loss of generality) C prefers J over J^* (can't have equal preference for 2 different jobs). Then, there exists some stable matching that contains the pair (J,C) (when J gets its optimal candidate) and there also exists some stable matching that contains the pair (J^*,C) (when J^* gets its optimal candidate). Then, in the stable matching containing (J^*,C), J is paired with a candidate it prefers less than C, (C is the best it can get, so in any stable matching J is not with C it is with someone less preferred than C). C also prefers J over J^* as established earlier. Then, we have a rogue couple. Thus, the matching is not stable and we have a contradiction because we've already established this to be a stable matching. □