CS 70 Discrete Mathematics and Probability Theory
Spring 2024 Seshia, Sinclair

HW 02

1 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) The pairing results in (C_i, J_i) for each $i \in \{1, 2, ..., n\}$. This result can be proved by induction:

Our base case is when $n = 1$, so the only pairing is (C_1, J_1), and thus the base case is trivially true.

Now assume this is true for some $n \in \mathbb{N}$. On the first day with $n+1$ jobs and $n+1$ candidates, all $n+1$ jobs will propose to C_1. C_1 prefers J_1 the most, and the rest of the jobs will be rejected. This leaves a set of n unpaired jobs and n unpaired candidates who all have the same preferences (after the pairing of (C_1, J_1)). By the process of induction, this means that every i^{th} preferred candidate will be paired with the i^{th} preferred job.

(b) The pairings will again result in (J_i, C_i) for each $i \in \{1, 2, ..., n\}$. This can be proved by induction in the same as above, but replacing "job" with "candidate" and vice-versa.

(c) We know that job-proposing produces a candidate-pessimal stable pairing. We also know that candidate-proposing produces a candidate-optimal stable pairing. We found that candidate-optimal and candidate-pessimal pairings are the same. This means that there is only one stable pairing, since both the best and worst pairings (for candidates) are the same pairings.

2 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:

To prove that there exists such a stable matching instance for any even $n \geq 2$, it suffices to construct such an instance. But first, we look at the $n = 2$ case to generate some intuition. We can recognize that for the following preferences:

J_1	$C_1 > C_2$
J_2	$C_2 > C_1$

C_1	$J_2 > J_1$
C_2	$J_1 > J_2$

both $S = \{(J_1, C_1), (J_2, C_2)\}$ and $T = \{(C_1, J_2), (C_2, J_1)\}$ are stable pairings.

The $n/2$ in the exponent motivates us to consider pairing the n jobs into $n/2$ groups of 2 and likewise for the candidates. We pair up job $2k-1$ and $2k$ into a pair and candidate $2k-1$ and $2k$ into a pair, for $1 \leq k \leq n/2$.

From here, we recognize that for each pair (J_{2k-1}, J_{2k}) and (C_{2k-1}, C_{2k}), mirroring the preferences above would yield 2 stable matchings from the perspective of just these pairs. If we can extend this perspective to all $n/2$ pairs, this would be a total of $2^{n/2}$ stable matchings.

Our construction thus results in preference lists like follows:

J_1	$C_1 > C_2 > \ldots$
J_2	$C_2 > C_1 > \ldots$
$\vdots$	$\vdots$
J_{2k-1}	$C_{2k-1} > C_{2k} > \ldots$
J_{2k}	$C_{2k} > C_{2k-1} > \ldots$
$\vdots$	$\vdots$
J_{n-1}	$C_{n-1} > C_n > \ldots$
J_n	$C_n > C_{n-1} > \ldots$

C_1	$J_2 > J_1 > \ldots$
C_2	$J_1 > J_2 > \ldots$
$\vdots$	$\vdots$
C_{2k-1}	$J_{2k} > J_{2k-1} > \ldots$
C_{2k}	$J_{2k-1} > J_{2k} > \ldots$
$\vdots$	$\vdots$
C_{n-1}	$J_n > J_{n-1} > \ldots$
C_n	$J_{n-1} > J_n > \ldots$

Each match will have jobs in the kth pair paired to candidates in the kth pair for $1 \leq k \leq n/2$.

A job j in pair k will never form a rogue couple with any candidate c in pair $m \neq k$ since it always prefers the candidates in this pair over all candidates across other pairs. Since each job in pair k can be stably matched to either candidate in pair k, and there are $n/2$ total pairs, the number of stable matchings is $2^{n/2}$.

3 Upper Bound

Note 4

(a) In the notes, we show that the stable matching algorithm terminates in at most n^2 days. Prove the following stronger result: the stable matching algorithm will always terminate in at most $(n-1)^2 + 1 = n^2 - 2n + 2$ days.

(b) Provide a set of preference lists for 4 jobs and 4 candidates that will result in the upper bound from part (a) when running the Propose-and-Reject algorithm. Verify this by running the Propose-and-Reject algorithm on your preference lists.

Solution:

(a) Recall that there is always a candidate who receives only one proposal (on the last day). Other than that candidate, every other candidate can reject up to $n-1$ jobs. Thus, there's a total of $(n-1)^2 = n^2 - 2n + 1$ rejections. Conceptually, in the worst case scenario, there would be exactly one rejection per day; if we were to hand out none, then the algorithm would terminate. On the final day, the candidate who is proposed to only once receives their offer. Thus, the process takes at most $(n-1)^2 + 1 = n^2 - 2n + 2$ days.

(b) As per discussion in the solution to part (a), the main motivation for the construction will be the line "in the worst case scenario, there would be exactly one rejection per day". One such preference list that would result in this as follows, and the execution of the algorithm is presented as well.

Jobs	Preferences
J_1	$C_1 > C_2 > C_3 > C_4$
J_2	$C_2 > C_3 > C_1 > C_4$
J_3	$C_3 > C_1 > C_2 > C_4$
J_4	$C_1 > C_2 > C_3 > C_4$

Candidates	Preferences
C_1	$J_2 > J_1 >$ anything
C_2	$J_3 > J_2 >$ anything
C_3	$J_4 > J_3 >$ anything
C_4	anything

Candidate	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7	Day 8	Day 9	Day 10
C_1	J_1, J_4	J_1	J_1	J_1, J_3	J_1	J_1	J_1, J_2	J_2	J_2	J_2
C_2	J_2	J_2, J_4	J_2	J_2	J_2, J_3	J_3	J_3	J_1, J_3	J_3	J_3
C_3	J_3	J_3	J_3, J_4	J_4	J_4	J_2, J_4	J_4	J_4	J_1, J_4	J_4
C_4										J_1

4 Build-Up Error?

Note 5

What is wrong with the following "proof"? In addition to finding a counterexample, you should explain what is fundamentally wrong with this approach, and why it demonstrates the danger of build-up error.

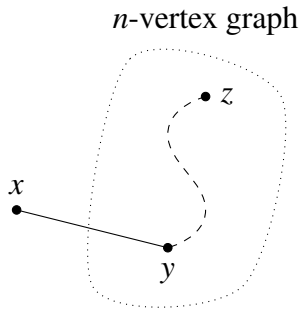
False Claim: If every vertex in an undirected graph has degree at least 1, then the graph is connected.

Proof? We use induction on the number of vertices $n \geq 1$.

Base case: There is only one graph with a single vertex and it has degree 0. Therefore, the base case is vacuously true, since the if-part is false.

Inductive hypothesis: Assume the claim is true for some $n \geq 1$.

Inductive step: We prove the claim is also true for $n+1$. Consider an undirected graph on n vertices in which every vertex has degree at least 1. By the inductive hypothesis, this graph is connected. Now add one more vertex x to obtain a graph on $(n+1)$ vertices, as shown below.



All that remains is to check that there is a path from x to every other vertex z. Since x has degree at least 1, there is an edge from x to some other vertex; call it y. Thus, we can obtain a path from x to z by adjoining the edge $\{x,y\}$ to the path from y to z. This proves the claim for $n+1$. □

Solution:

The mistake is in the argument that *"every $(n+1)$-vertex graph with minimum degree 1 can be obtained from an n-vertex graph with minimum degree 1 by adding 1 more vertex"*. Instead of starting by considering an arbitrary $(n+1)$-vertex graph, this proof only considers an $(n+1)$-vertex graph that you can make by starting with an n-vertex graph with minimum degree 1. As a counterexample, consider a graph on four vertices $V = \{1,2,3,4\}$ with two edges $E = \{\{1,2\},\{3,4\}\}$. Every vertex in this graph has degree 1, but there is no way to build this 4-vertex graph from a 3-vertex graph with minimum degree 1.

More generally, this is an example of *build-up error* in proof by induction. Usually this arises from a faulty assumption that every size $n+1$ graph with some property can be "built up" from a size n graph with the same property. (This assumption is correct for some properties, but incorrect for others, such as the one in the argument above.)

One way to avoid an accidental build-up error is to use a *"shrink down, grow back"* process in the inductive step: start with a size $n+1$ graph, remove a vertex (or edge), apply the inductive hypothesis $P(n)$ to the smaller graph, and then add back the vertex (or edge) and argue that $P(n+1)$ holds.

Let's see what would have happened if we'd tried to prove the claim above by this method. In the inductive step, we must show that $P(n)$ implies $P(n+1)$ for all $n \geq 1$. Consider an $(n+1)$- vertex graph G in which every vertex has degree at least 1. Remove an arbitrary vertex v, leaving an n-vertex graph G' in which every vertex has degree... uh-oh! The reduced graph G' might contain a vertex of degree 0, making the inductive hypothesis $P(n)$ inapplicable! We are stuck—and properly so, since the claim is false!

5 Proofs in Graphs

Note 5

(a) On the axis from San Francisco traffic habits to Los Angeles traffic habits, Old California is more towards San Francisco: that is, civilized. In Old California, all roads were one way streets. Suppose Old California had n cities ($n \geq 2$) such that for every pair of cities X and Y,

either X had a road to Y or Y had a road to X.

Prove that there existed a city which was reachable from every other city by traveling through at most 2 roads.

[*Hint:* Induction]

(b) Consider a connected graph G with n vertices which has exactly $2m$ vertices of odd degree, where $m > 0$. Prove that there are m walks that *together* cover all the edges of G (i.e., each edge of G occurs in exactly one of the m walks, and each of the walks should not contain any particular edge more than once).

[*Hint:* In lecture, we have shown that a connected undirected graph has an Eulerian tour if and only if every vertex has even degree. This fact may be useful in the proof.]

(c) Prove that any graph G is bipartite if and only if it has no tours of odd length.

[*Hint:* In one of the directions, consider the lengths of paths starting from a given vertex.]

Solution:

(a) We prove this by induction on the number of cities n.

Base case: For $n = 2$, there's always a road from one city to the other.

Inductive Hypothesis: When there are k cities, there exists a city c that is reachable from every other city by traveling through at most 2 roads.

Inductive Step: Consider the case where there are $k+1$ cities. Remove one of the cities d and all of the roads to and from d. Now there are k cities, and by our inductive hypothesis, there exists some city c which is reachable from every other city by traveling through at most 2 roads. Let A be the set of cities with a road to c, and B be the set of cities two roads away from c. The inductive hypothesis states that the set S of the k cities consists of $S = \{c\} \cup A \cup B$.

Now add back d and all roads to and from d.

Between d and every city in S, there must be a road from one to the other. If there is at least one road from d to $\{c\} \cup A$, c would still be reachable from d with at most 2 road traversals. Otherwise, if all roads from $\{c\} \cup A$ point to d, d will be reachable from every city in B with at most 2 road traversals, because every city in B can take one road to go to a city in A, then take one more road to go to d. In either case there exists a city in the new set of $k+1$ cities that is reachable from every other city by traveling at most 2 roads.

Alternate Solution :Alternatively, we can prove this using properties of directed graphs. Let c be the city with the largest in-degree. Note that this graph is essentially a complete graph, where each edge is a directed edge instead of an undirected edge. Therefore, the total in degree sums to $n(n-1)/2$, and so does the total out degree. In addition, the in degree + out degree of any vertex must add up to $n-1$.

Because the total in-degree of all vertices is $n(n-1)/2$, The largest in-degree is $d \geq (n-1)/2$. Let S be the these d cities that can reach c by one edge.

For any other city x, it has to have at least $(n-1)-d$ out-degree (because in-degree $<= d$). Notice that there are n total vertices, two of which are x or c, and d vertices that connect to c through one edge. Thus, there are $n-2-d$ other vertices. Since x has out degree at least $n-1-d > n-2-d$, it must therefore connect to at least one vertex in S by the pigeonhole principle.

Thus, all vertices are either connected to c through 1 or 2 edges.

(b) We split the $2m$ odd-degree vertices into m pairs, and join each pair with an edge, adding m more edges in total. (Here, we allow for the possibility of multi-edges, that is, pairs of vertices with more than one edge between them.) Notice that now all vertices in this graph are of even degree. Now by Euler's theorem the resulting graph has an Eulerian tour. Removing the m added edges breaks the tour into m walks covering all the edges in the original graph, with each edge belonging to exactly one walk.

(c) To prove the claim, we need to prove two directions: if G is bipartite, it contains no tours of odd length, and if G contains no tours of odd length, it must be bipartite.

Suppose G is bipartite, and let L and R be the two disjoint sets of vertices such that there does not exist any edge between two vertices in L or two vertices in R. Further, suppose there is some tour in G, and we start traversing this tour at $v_0 \in L$.

Since each edge in G connects a vertex in L to a vertex in R, the first edge in the tour connects the start vertex v_0 to a vertex $v_1 \in R$. Similarly, the second edge connects $v_1 \in R$ to $v_2 \in L$. In general, it must be the case that the $2k$th edge connects vertex $v_{2k-1} \in R$ to $v_{2k} \in L$, and the $2k+1$th edge connects vertex $v_{2k} \in L$ to $v_{2k+1} \in R$.

Since only even numbered edges connect to vertices in L, and we started our tour in L, the tour must end with an even number of edges.

For the opposite direction, suppose G contains no tours of odd length. Without loss of generality, let us consider one connected component of G; the following reasoning can be applied to all of the connected components of G.

Let v be an arbitrary vertex in G; we can divide all of the vertices in G into two disjoint sets:

$$R = \{u \mid \text{the shortest path from } u \text{ to } v \text{ is even}\}$$
$$L = \{u \mid \text{the shortest path from } u \text{ to } v \text{ is odd}\}$$

We claim that no two vertices in L are adjacent. For contradiction, suppose there do exist adjacent vertices $u_1, u_2 \in L$. Consider the tour consisting of:

- the shortest path from v to u_1 (odd length)
- the edge (u_1, u_2) (length 1)
- the shortest path from u_2 to v (odd length)

This tour has odd length, and contradicts our assumption that G has no tours of odd length. This means that no two vertices in L are adjacent.

Similarly, we claim that no two vertices in R are adjacent. For contradiction, suppose there do exist adjacent vertices $u_1, u_2 \in R$. Consider the tour consisting of:

- the shortest path from v to u_1 (even length)
- the edge (u_1, u_2) (length 1)
- the shortest path from u_2 to v (even length)

This tour has odd length, and contradicts our assumption that G has no tours of odd length. This means that no two vertices in R are adjacent.

We've just shown that there are no edges between two vertices in L, and no edges between two vertices in R. If there are multiple connected components in G, the same partition can be applied to all of the components. Together, this means that G is bipartite.

6 (Optional) Nothing Can Be Better Than Something

Note 4

In the stable matching problem, suppose that some jobs and candidates have hard requirements and might not be able to just settle for anything. In other words, each job/candidate prefers being unmatched rather than be matched with those below a certain point in their preference list. Let the term "entity" refer to a candidate/job. A matching could ultimately have to be partial, i.e., some entities would and should remain unmatched.

Consequently, the notion of stability here should be adjusted a little bit to capture the autonomy of both jobs to unilaterally fire employees and/or employees to just walk away. A matching is stable if

- there is no matched entity who prefers being unmatched over being with their current partner;
- there is no matched/filled job and unmatched candidate that would both prefer to be matched with each other over their current status;
- there is no matched job and matched candidate that would both prefer to be matched with each other over their current partners; and
- similarly, there is no unmatched job and matched candidate that would both prefer to be matched with each other over their current status;
- there is no unmatched job and unmatched candidate that would both prefer to be with each other over being unmatched.

(a) Prove that a stable pairing still exists in the case where we allow unmatched entities.

(HINT: You can approach this by introducing imaginary/virtual entities that jobs/candidates "match" if they are unmatched. How should you adjust the preference lists of jobs/candidates, including those of the newly introduced imaginary ones for this to work?)

(b) As you saw in the lecture, we may have different stable matchings. But interestingly, if an entity remains unmatched in one stable matching, they must remain unmatched in any other stable matching as well. Prove this fact by contradiction.

Solution:

(a) We form an instance of the standard stable matching problem as follows. Following, the hint, we introduce an imaginary mate, r_e, (let's call it a robot) for each entity. Note that we introduce one robot for each entity, i.e. there are as many robots as there are candidates+jobs. For simplicity let us say each robot, r_e, is owned by the entity, e, we introduce it for.

Each robot, r_e, prefers its owner e, i.e. it puts its owner at the top of its preference list. The rest of its preference list is arbitrary and includes all other owners and the robots of other types of entities. An entity e of a robot, r_e puts it in their preference list exactly after the last entity they are willing to match with. i.e. owners like their robots more than entities they are not willing to match, but less than entities they like to match. The other robots can be placed in arbitrary order in e's list.

For this instance of the stable matching problem which we refer to as the robot instance, I, the propose and reject algorithm will give us a stable matching, M.

To extract the desired partial matching, we simply remove all pairs in M with at least one robot (two robots can match each other). We refer to each entity which is not matched as single. We observe, an entity, e, will never be matched with another entity's robot, $r_{e'}$, because then e and its robot, r_e, would form a rogue couple (r_e prefers e to other owners, and e prefers r_e more than other robots). It remains to show this is a stable matching as specified in the problem as follows:

- there is no matched entity, e, who prefers being unmatched over being with their current partner since the e and r_e would form a rogue couple in M;
- there is no matched/filled job, j, and unmatched candidate, c, that would both prefer to be matched with each other since otherwise j and c would form a rogue couple in M;
- there is no matched job j and matched candidate c that would both prefer to be matched with each other over their current partners since then j and c would form a rogue couple in M;
- similarly, there is no unmatched job, j, and matched candidate, c, that would both prefer to be matched with each other over their current status since j and c would be a rogue couple in M;
- there is no unmatched job, j, and unmatched candidate, c, that would both prefer to be with each other over being unmatched since j and c would be a rogue couple in M.

Thus, the resulting partial matching is stable.

(b) We will perform proof by contradiction. Assume that there exists some job j_1 who is paired with a candidate c_1 in stable pairing S and unpaired in stable pairing T. The stable pairing S where j_1 and c_1 are paired means j_1 and c_1 both prefer to be with each other over being single. Since T is a stable pairing and j_1 is unpaired, c_1 must be paired in T with a job j_2 whom they prefer over j_1. (If c_1 were unpaired or paired with a job they do not prefer over j_1, then (j_1, c_1) would be a rogue couple in T, which is a contradiction.)

Since j_2 is paired with c_1 in T, it must be paired in S with some candidate c_2 whom j_2 prefers over c_1. This process continues (c_2 must be paired with some j_3 in T, j_3 must be paired with some c_3 in S, etc.) with the pattern that (c_i, j_i) is in S and (j_i, c_{i-1}) is in T. At some time i, one must encounter j_1 in this process as there are a finite number of jobs. A this point $(j_1 = j_i, c_{i-1})$ is in T, which implies that j_1 is matched in T.

A similar argument can be used for candidates.

This contradicts the assumption that j_1 is unmatched in T. Since no job or candidate can be paired in one stable pairing and unpaired in another, every job or candidate must be either paired in all stable pairings or unpaired in all stable pairings.