CS 70 Discrete Mathematics and Probability Theory
Spring 2021

HW 2

1 Stable Matching for Classes!

Let's consider the system for getting into classes. For simplicity, we will start with the problem assigning students to lab sections first, since it is clear that there are a finite number of seats. We are given n students and m lab sections. Each lab section ℓ has some number, q_ℓ of seats, and we assume that the total number of students is larger than the total number of seats (i.e. $\sum_{\ell=1}^{m} q_\ell < n$) and so some students are going to end up with no lab. Each student ranks the m lab sections in order of preference, and the instructor for each lab ranks the n students. Our goal is to find an assignment of students to seats (one student per seat) that is *stable* in the following sense:

- There is no student-lab pair (s, ℓ) such that s prefers ℓ to her allocated lab section and the instructor for ℓ prefers s to one of the students assigned to ℓ. (This is like the stability criterion you have seen for jobs: it says there is no student-lab pair that would induce that lab instructor to kick out an existing student and take this new student instead.)
- There is no lab section ℓ for which the instructor prefers some unassigned student s to one of the students assigned to ℓ. (This extends the stability criterion to take account of the fact that some students are not assigned to labs.)

Note that this problem is almost the same as the Stable Matching problem for jobs/internships presented in the lecture note, with two differences: (i) there are more students than seats; and (ii) each lab section can have more than one seat.

Perhaps you will agree that this extended model is more realistic, even for the jobs context!

(a) Explain how to modify the propose-and-reject algorithm so that it finds a stable assignment of students to seats. [*Hint*: What roles of students/instructors will be in the propose-and-reject algorithm? What does "candidates have a job offer in hand (on a string)" mean in this setting?]

(b) State a version of the Improvement Lemma (see the Stable Matchings Lecture Note) that applies to your algorithm, and prove that it holds.

(c) Use your Improvement Lemma to give a proof that your algorithm terminates, that every seat is filled, and that the assignment your algorithm returns is stable.

(d) Let us consider the potential of a pair of students wishing to swap lab sections, subject to global stability (i.e. the swap can't make the matching as a whole unstable). Either prove that your algorithm will not have any such swap requests or modify it to have no such swap requests and prove that the modified one will not have any pair of student wanting to stably-swap sections.

Solution:

(a) We will extend the propose-and-reject algorithm given in the lecture notes. Students will play the role of jobs and discussion section instructors will play the role of candidates. Instead of keeping a single person as in the original algorithm, each discussion section instructor will keep a *waitlist* of size equal to its quota.

Note that there are other valid ways to modify the propose-and-reject algorithm such that a stable assignment is produced. One way is to have the instructors playing the role of jobs and the students playing the role of candidates, with the difference that now we have jobs proposing to up to q_u candidates each day. It is also possible to "expand" each instructor by the size of his or her quota by making q_u copies of instructor u and adding empty discussions for the unassigned students.

The extended procedure for students proposing and instructors keeping a waitlist works as follows:

- All students apply to their first-choice discussion section.
- Each discussion section u with a quota of q_u, then places on its waitlist the q_u applicants who rank highest (or all the applicants if there are fewer than q_u of them) and rejects all the rest.
- Rejected applicants then apply to their second-choice discussion section, and again each discussion section instructor u selects the top q_u students from among the new applicants AND those on its waitlist; it puts the selected students on its new waitlist, and rejects the rest of its applicants (including those who were previously on its waitlist but now are not).
- The above procedure is repeated until every applicant is either on a waitlist or has been rejected from every discussion section. At this point, each discussion section admits everyone on its waitlist.

(b) Improvement Lemma: Assume that discussion sections maintain their waitlist in decreasing order of their preference for the students on the lists. Let $\oplus$ be the "null" element, which we will use as a placeholder in waitlist positions not yet assigned to a student. For any discussion section u, let q_u be its quota and let $s_i^k \in \{\text{Students}\} \cup \{\oplus\}$ be the student in the i'th position on the waitlist after the k'th round of the algorithm. (If there are fewer than q_u students on the list, we fill the bottom of the list with $\oplus$'s.) Then for all i and k, the discussion section instructor likes s_i^{k+1} at least as much as s_i^k. (Here we assume that the discussion section instructor prefers any student to no student.)

In short, the lemma says that no position in the list ever gets worse for the discussion section instructor as the algorithm proceeds. As in the Lecture Notes, we use the Well-Ordering Principle and prove the lemma by contradiction:

Suppose that the jth day, where $j > k$, is the first counterexample where, for index $i \leq q_u$, discussion section u, has either nobody or some student $\hat{s}$ inferior to s_i^k. Then on day $j-1$, the instructor has some student $\tilde{s}$ on a string that they like at least as much as s_i^k. Following the algorithm, $\tilde{s}$ still "proposes" to the instructor of section u on day j since they said "maybe" the

previous day. Therefore, the instructor has the choice of at least one student on the jth day, and his or her best option is at least as good as $\tilde{s}$, so the instructor would have chosen $\tilde{s}$ over $\hat{s}$. Therefore on day j, the instructor *does* have a student that they like at least as much as s_i^k in waitlist position $i \leq q_u$. This contradicts our initial assumption.

(c) First, the algorithm terminates. This follows by similar reasoning to the original propose-and-reject algorithm: in each round (except the last), at least one discussion section is crossed off the list of some rejected students.

Second, every seat is filled. Suppose some discussion section u has an unfilled seat at the end. Then the total number of students who applied to u must have been fewer than its quota q_u (since the Improvement Lemma in Part (b) ensures that a waitlist slot, once filled, will never later be unfilled). But the only students who do *not* apply to u are those who find a slot in some other discussion section. And since we are told that there are more students than seats, the number of students applying to u must be at least q_u.

Finally, the assignment is stable:

- Suppose there is a student-section pair (s,u) such that s prefers u to her discussion section u' in the final allocation. Then s must have proposed to u prior to proposing to u', and was rejected by u. Thus immediately after rejecting s, u must have had a full waitlist in which every student was preferred to s. By the Improvement Lemma, the same holds at all future times and hence at the end. Thus u does not prefer s to any of its assigned students, as required.
- Suppose s is a student left unassigned at the end. Consider any discussion section u. Since s must have applied to and have been rejected from all discussion sections, this holds in particular for u. Reasoning similar as in the previous case, using the Improvement Lemma, we see that in the final allocation, u prefers all its students to s. Therefore u does not prefer s to any of its assigned students, as required.

This concludes the proof that our algorithm finds a stable assignment when terminates.

(d) The intuition here is that the group proposing in the Propose-and-Reject algorithm is the group for which the resulting stable matchings are optimal. We proceed with a proof.

Assume that there is a pair (s,u) and a pair (s',u') such that s' prefers u to u' and s prefers u' to u. If this is the case, then s must have proposed to u' before u, and gotten rejected, and s' must have proposed to u before u', and gotten rejected.

Without loss of generality, assume s was rejected first.

If u' rejected any student, s'' that it prefers to s throughout the algorithm, then if s and s' swap, s" and u' would become a rogue pair which means swapping makes for an unstable pairing and we are done.

Thus, we can assume u' never rejects any student it prefers to s. This and the fact that everyone on u''s waitlist was preferred to s when s was rejected by u', implies that the waitlist for u' on

that day must be the exact students that are in u' wailtist at the end of the algorithm. Thus, s' must be on u''s waitlist at that time and therefore must have been rejected by u previously.

This contradicts the assumption that s was rejected first. Thus, switching the pairs creates an unstable matching.

2 Pairing Up

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 $n \geq 2$, we just need to show how to construct such an instance.

The idea here is that we can create pairs of jobs and pairs of candidates: pair up job $2k-1$ and $2k$ into a pair and candidate $2k-1$ and $2k$ into a pair, for $1 \leq k \leq n/2$ (you might come to this idea since we are asked to prove this for *even* n).

For n, we have $n/2$ pairs. Choose the preference lists such that the kth pair of jobs rank the kth pair of candidates just higher than the $(k+1)$th pair of candidates (the pairs wrap around from the last pair to the first pair), and the kth pair of candidates rank the kth pair of jobs just higher than the $(k+1)$th pair of jobs. Within each pair of pairs (j, j') and (c, c'), let j prefer c, let j' prefer c', let c prefer j', and let c' prefer j.

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 $m \neq k$. If $m > k$, then c prefers her current partner in the kth pair to j. If $m < k$, then j prefers its current partner in the kth pair to c. Then a rogue couple could only exist in the same pair - but this is impossible since exactly one of either j or c must be matched to their preferred choice in the pair.

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 A Better Stable Pairing

In this problem we examine a simple way to *merge* two different solutions to a stable matching problem. Let R, R' be two distinct stable pairings. Define the new pairing $R \wedge R'$ as follows:

> For every job j, j's partner in $R \wedge R'$ is whichever is better (according to j's preference list) of their partners in R and R'.

Also, we will say that a job/candidate *prefers* a pairing R to a pairing R' if they prefers their partner in R to their partner in R'. We will use the following example:

jobs	preferences	candidates	preferences
A	1>2>3>4	1	D>C>B>A
B	2>1>4>3	2	C>D>A>B
C	3>4>1>2	3	B>A>D>C
D	4>3>2>1	4	A>B>D>C

(a) $R = \{(A,4),(B,3),(C,1),(D,2)\}$ and $R' = \{(A,3),(B,4),(C,2),(D,1)\}$ are stable pairings for the example given above. Calculate $R \wedge R'$ and show that it is also stable.

(b) Prove that, for any pairings R, R', no job prefers R or R' to $R \wedge R'$.

(c) Prove that, for any stable pairings R, R' where j and c are partners in R but not in R', one of the following holds:

- j prefers R to R' and c prefers R' to R; or
- j prefers R' to R and c prefers R to R'.

[*Hint*: Let J and C denote the sets of jobs and candidates respectively that prefer R to R', and J' and C' the sets of jobs and candidates that prefer R' to R. Note that $|J| + |J'| = |C| + |C'|$. (Why is this?) Show that $|J| \leq |C'|$ and that $|J'| \leq |C|$. Deduce that $|J'| = |C|$ and $|J| = |C'|$. The claim should now follow quite easily.]

(You may assume this result in the next part even if you don't prove it here.)

(d) Prove an interesting result: for any stable pairings R, R', (i) $R \wedge R'$ is a pairing [*Hint*: use the results from (c)], and (ii) it is also stable.

Solution:

(a) $R \wedge R' = \{(A,3),(B,4),(C,1),(D,2)\}$. This pairing can be seen to be stable by considering the different combinations of jobs and candidates. For instance, A prefers 2 to their current partner 3. However, 2 prefers her current partner D to A. Similarly, A prefers 1 the most, but 1 prefers her current partner C to A. We can prove the stability of this pairing by considering the remaining pairs like this.

(b) Let j be a job, and let their partners in R and R' be c and c' respectively, and without loss of generality, let $c > c'$ in j's list. Then their partner in $R \wedge R'$ is c, whom they prefer over c'. However, for j to prefer R or R' over $R \wedge R'$, j must prefer c or c' over c, which is not possible (since $c > c'$ in their list).

(c) Let J and C denote the sets of jobs and candidates respectively that prefer R to R', and J' and C' the sets of jobs and candidates that prefer R' to R. Note that $|J| + |J'| = |C| + |C'|$, since the left-hand side is the number of jobs who have different partners in the two pairings, and the right-hand side is the number of candidates who have different partners.

Now, in R there cannot be a pair (j,c) such that $j \in J$ and $c \in C$, since this will be a rogue couple in R'. Hence the partner in R of every jobs in J must lie in C', and hence $|J| \leq |C'|$. A

similar argument shows that every jobs in J' must have a partner in R' who lies in C, and hence $|J'| \leq |C|$.

Since $|J| + |J'| = |C| + |C'|$, both these inequalities must actually be tight, and hence we have $|J'| = |C|$ and $|J| = |C'|$. The result is now immediate: if the jobs j is partners with the candidate c in one but not both pairings, then

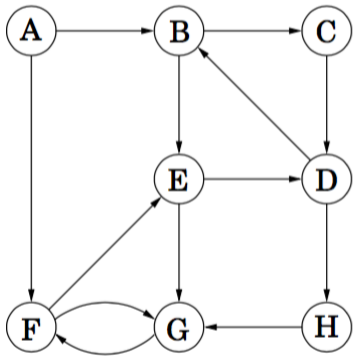
- either $j \in J$ and $c \in C'$, i.e., j prefers R to R' and c prefers R' to R,
- or $j \in J'$ and $c \in C$, i.e., j prefers R' to R and c prefers R to R'.

(d) (i) If $R \wedge R'$ is not a pairing, then it is because two jobs get the same candidate, or two candidates get the same job. Without loss of generality, assume it is the former case, with $(j,c) \in R$ and $(j',c) \in R'$ causing the problem. Hence j prefers R to R', and j' prefers R' to R. Using the results of the previous part would imply that c would prefer R' over R, and R over R' respectively, which is a contradiction.

(ii) Now suppose $R \wedge R'$ has a rogue couple (j,c). Then j strictly prefers c to their partners in both R and R'. Further, c prefers j to her partner in $R \wedge R'$. Let c's partners in R and R' be j_1 and j_2. If she is finally matched to j_1, then (j,c) is a rogue couple in R; on the other hand, if she is matched to j_2, then (j,c) is a rogue couple in R'. Since these are the only two choices for c's partner, we have a contradiction in either case.

4 Graph Basics

In the first few parts, you will be answering questions on the following graph G.



(a) What are the vertex and edge sets V and E for graph G?

(b) Which vertex has the highest in-degree? Which vertex has the lowest in-degree? Which vertices have the same in-degree and out-degree?

(c) What are the paths from vertex B to F, assuming no vertex is visited twice? Which one is the shortest path?

(d) Which of the following are cycles in G?

i. $(B,C),(C,D),(D,B)$

ii. $(F,G),(G,F)$

iii. $(A,B),(B,C),(C,D),(D,B)$

iv. $(B,C),(C,D),(D,H),(H,G),(G,F),(F,E),(E,D),(D,B)$

(e) Which of the following are walks in G?

i. (E,G)

ii. $(E,G),(G,F)$

iii. $(F,G),(G,F)$

iv. $(A,B),(B,C),(C,D),(H,G)$

v. $(E,G),(G,F),(F,G),(G,C)$

vi. $(E,D),(D,B),(B,E),(E,D),(D,H),(H,G),(G,F)$

(f) Which of the following are tours in G?

i. (E,G)

ii. $(E,G),(G,F)$

iii. $(F,G),(G,F)$

iv. $(E,D),(D,B),(B,E),(E,D),(D,H),(H,G),(G,F)$

v. $(B,C),(C,D),(D,H),(H,G),(G,F),(F,E),(E,D),(D,B)$

In the following three parts, let's consider a general undirected graph G with n vertices ($n \geq 3$). If true, provide a short proof. If false, show a counterexample.

(g) True/False: If each vertex of G has degree at most 1, then G does not have a cycle.

(h) True/False: If each vertex of G has degree at least 2, then G has a cycle.

(i) True/False: If each vertex of G has degree at most 2, then G is not connected.

Solution:

(a) A graph is specified as an ordered pair $G = (V,E)$, where V is the vertex set and E is the edge set.

$V = \{A,B,C,D,E,F,G,H\}$,
$E = \{(A,B),(A,F),(B,C),(B,E),(C,D),(D,B),(D,H),(E,D),(E,G),(F,E),(F,G),(G,F),$
$(H,G)\}$.

(b) G has the highest in-degree (3). A has the lowest in-degree (0).

$\{B,C,D,E,F,H\}$ all have the same in-degree and out-degree. H and C has in-degree (out-degree) equal to 1 and the other four have in-degree (out-degree) equal to 2.

(c) There are three paths:
$(B,C),(C,D),(D,H),(H,G),(G,F)$
$(B,E),(E,D),(D,H),(H,G),(G,F)$
$(B,E),(E,G),(G,F)$
The first two have length 5, while the last one has length 3, so the last one is the shortest path.

(d) A cycle is a path that starts and ends at the same point. This means that (iii) is not a cycle, since it starts at A but ends at B. In addition, all the vertices $\{v_1,\ldots,v_n\}$ in the cycle $(v_1,v_2),(v_2,v_3),\ldots,(v_{n-1},v_n)$ should be distinct, so (iv) is not a cycle. The correct answers are (i) and (ii).

(e) A walk consists of any sequence of edges such that the endpoint of each edge is the same as the starting vertex of the next edge in the sequence. Example (iv) does not fit this definition–even though it uses only valid edges, the endpoint of the second to last edge in D, while the start point of the next edge is H. Example (v) also is not a walk, since it tries to walk from G to C as its last step, but there is no such edge. All the rest are walks.

(f) A tour is a walk that has the same start and end vertex; examples (iii) and (v) satisfy this definition. Note in part (d), we already said that (iii) was a cycle–and indeed, all cycles are also tours.

(g) True. In order for there to be a cycle in G starting and ending at some vertex v, we would need at least two edges incident to v: one to leave v at the start of the cycle, and one to return to v at the end. If every vertex has degree at most 1, no vertex has two or more edges incident on it, so no vertex is capable of acting as the start and end point of a cycle.

(h) True. Consider starting a walk at some vertex v_0, and at each step, walking along a previously untraversed edge, stopping when we first visit some vertex w for the second time. If this process terminates, the part of our walk from the first time we visited w until the second time is a cycle. Thus, it remains only to argue this process always terminates.

Each time we take a step from some vertex v, since we are not stopping, we must have visited that vertex exactly once and not yet left. It follows that we have used at most one edge incident with v (either we started at v, or we took an edge into v). Since v has degree at least 2, there must be another edge leaving v for us to take.

(i) False. For example, a 3-cycle (triangle) is connected and every vertex has degree 2.

5 Proofs in Graphs

Please prove or disprove the following claims.

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

[*Hint:* Induction]

(b) In lecture, we have shown that a connected undirected graph has an Eulerian tour if and only if every vertex has even degree.

Consider a connected graph G with n vertices which has exactly $2m$ vertices of odd degree, where $m > 0$. Prove or disprove 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).

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.

(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.