Greedy alg proofs

Where S is the ordered set of students, C_s is the preference list of s, X is a set of unmatched students in the first round, C is the set of classes and "available" means it has open seats:

Algorithm 1 Greedy Algorithm (GA)

```
1:  for each student s in S do                          ▷ start of round 1
2:      for each class c in C_s do
3:          if c is available then
4:              μ(s) ← c
5:              break                                   ▷ move to next student
6:          end if
7:      end for
8:      append s to X
9:  end for
10: for each student s in X do                          ▷ start of round 2
11:     choose random available c from C
12:     μ(s) ← c
13: end for
14: return matching μ
```

Theorem 0.1. *GA outputs a stable matching in a vanilla environment.*

Proof. Let μ denote the outputted matching over an ordered set of students, S. Let $a, b \in S$, where a comes before b in the order. Assume for the sake of a contradiction that a and b form a blocking pair, such that $\mu(b) >_a \mu(a)$ and $\mu(a) >_b \mu(b)$.

By lines 3 and 11 of GA, we know $\mu(a)$ was either a's top available choice or a random assignment. So, if $\mu(b) >_a \mu(a)$, then $\mu(b)$ must not be available when a is matched, which means b was matched first. However, this is a contradiction of our assumption that a comes before b in the input to the algorithm. Thus, $\mu(b)$ is available when a is matched, so it must be the case that $\mu(a) >_a \mu(b)$. Therefore, a and b do not form a blocking pair, and we can conclude that μ is stable. □

Theorem 0.2. *GA does not always output a stable matching with quotas (caps).*

Proof. Let μ denote GA's outputted matching over a set of students, S. We will prove by counter example that μ is not stable.

Let $C = \{c_1, c_2, c_3\}$, where there are two types of student α and β such that each $c \in C$ can have at most one student of each type. Let $S = \{A1, B1, A2, B2\}$ where students $A1, A2$ are of type α and $B1, B2$ are of type β. Let the preference lists of

the students be as follows:

$$\begin{aligned} A1: &\quad c_1 \succ c_3 \\ B1: &\quad c_2 \succ c_3 \\ A2: &\quad c_1 \succ c_2 \\ B2: &\quad c_2 \succ c_1 \end{aligned}$$

Observe that in μ, $\mu(A1) = c_1$ and $\mu(B1) = c_2$, because they are the first two students to be matched and their first choices are different, so they receive their top choices. Next, $\mu(A2) = c_2$ because c_1 already has a student of type α so c_1 is no longer available when matching c_α, and similarly $\mu(B2) = c_1$ because c_2 already has a student of type β. Notice that $\mu(B2) >_A 2\mu(A2)$ and $\mu(A2) >_B 2\mu(B2)$, forming a blocking pair in which $A2$ and $B2$ are incentivized to switch classes with each other. So, it must be the case that μ is not a stable matching and therefore GA cannot guarantee a stable matching under quota/cap constraints. □

Theorem 0.3. *There is not always a stable matching with quotas (caps) and incomplete preferences.*

Proof. Take the above instance, and note that if we alter the environment slightly there is no stable matching. Remove c_3 from C so that $C = \{c_1, c_2\}$ and let the preference lists be as follows:

$$\begin{aligned} A1: &\quad c_1 \\ B1: &\quad c_2 \\ A2: &\quad c_1 \succ c_2 \\ B2: &\quad c_2 \succ c_1 \end{aligned}$$

In this case, regardless of the order of matching or algorithm, there is no stable matching. The A students will both wish to be matched to c_1, which only has one α spot, and both B students wish to be matched to c_2 which only has one β spot. So, when the first round of matching is done, that means that there is still an open β spot for c_1 and an open α spot for c_2, which the unhappy A and B students can then switch into on their own, respectively.

Let's look at a more basic constraint environment, where we have 0/1 preferences and therefore no notion of stability (for now). Let C denote the set of seats over all classes, and S the set of students. We will put lower bounds on the seats, such that the number of seats in each class will be reserved evenly among each type of student, with an extra group of seats that is free to any student. We will let C_r be the set of reserved seats. In this case, we have a weighted graph matching problem, where $G = (V, E)$ such that $V = S \cup C$ and each edge goes from some s to some c if and only if $c \in P_s$, where P_s is the preference list of s. Each edge has a weight $w(e)$. Let the weight be as follows:

$$w(e) = \mathbf{1}\{\text{if seat } c \text{ is reserved for } s\text{'s type}\} + \mathbf{1}\{\text{if student } s \text{ prefers class } c\}.$$

□

Theorem 0.4. *If a matching that respects student preferences and seat reservations exists, it will be the max-weight graph where* $w(\mu) = |S| + |C_r|$.

Proof. Let μ be a matching that assigns every student to a seat they prefer while also filling reserved seats first, such that all reserved seats are filled before the overflow seats. By construction of the weights, we can determine a lower and upper bound for $w(\mu)$.

1. **Lower bound of** μ Each edge exists only if the student prefers that seat, so every edge in μ contributes at least 1. By our assumption of characteristics of μ, every student in μ got a seat they prefer, so summing over all students contributes at least $|S|$ to $w(\mu)$. We also assumed each reserved seat was assigned to a student in μ, which adds an extra +1 to the weight for the edge from each of those reserved seats. Summing over all reserved seats gives an additional contribution of $|C_r|$, because the rest of the seats are still 1-weight edges. Therefore, we can bound μ:

$$w(\mu) \geq |S| + |C_r|.$$

2. **Upper bound of** μ From our definition, we know each edge weight is at most 2. Similarly to the lower bound, we know μ has $|S|$ edges because each student was matched to one of their preferences. However, because only the reserved seats contribute an additional weight to an edge, we know that the number of 2-weight edges $\leq |C_r|$. So, the maximum summed weight of all edges must be $2|C_r| + |S| - |C_r|$, representing the maximum number of 2-weight edges plus all students assigned to an overflow seat. Note that is equivalent to $|S| + |C_r|$. Therefore, we can bound μ:

$$w(\mu) \leq |S| + |C_r|.$$

So, μ is a maximum-weight graph with a lower bound of $|S| + |C_r|$ and an upper bound of $|S| + |C_r|$, so it must be that a matching that fully respects student preferences and seat reservations has a weight $w(\mu) = |S| + |C_r|$. □

We will define a new algorithm called the **Weighted-Trading** algorithm, where we first construct a graph G based on 0/1 preferences as represented by the environment from theorem []. We will then run the max-weight algorithm. If this algorithm outputs a matching μ such that $w(\mu) = |S| + |C_r|$, we will then run a trading algorithm on it to reintroduce some notion of stability. If $w(\mu) < |S| + |C_r|$, we will relax the constraints by reducing the number of reserved seats per class and re-run the max-weight algorithm until we get a matching where $w(\mu) = |S| + |C_r|$. The trading involves going through the students in each group sequentially and determining whether they form a blocking pair with another student in their group, and if so, swapping them until there are no more blocking pairs.

We are introducing a new notion of stability here, where we will consider a matching stable for our purposes if it exhibits *intra-group* stability. This means that for any type of student, that student does not form a blocking pair with another student of the same type. For now, inter-group instability, or blocking pairs involving students of different types, does not interfere with our notion of stability.

Formally, a matching μ is stable if, for any two students a_1 and a_2 of type $A \subset S$, $\mu(a_1) >_{a_1} \mu(a_2)$ or $\mu(a_2) >_{a_2} \mu(a_1)$ such that there exist no intra-group blocking pairs.

Algorithm 2 Weighted-Trading Algorithm *WT*

1: $w(\mu) \leftarrow 0$
2: **while** $w(\mu) \neq |S| + |C_r|$ **do**
3: $\quad \mu \leftarrow$ Max-Weight(S, C) ▷ with reduced $|C_r|$ (after the first time)
4: **end while**
5: **for** each $A \subset S$ **do**
6: $\quad$ **while** $\exists$ blocking pair a, a_i **do**
7: $\quad\quad \mu(a) \leftarrow \mu(a_i)$, $\mu(a_i) \leftarrow \mu(a)$
8: $\quad$ **end while**
9: **end for**
10: **return** matching μ

Theorem 0.5. *For a matching μ output by the WT algorithm, $w(\mu) = |S| + |C_r|$.*

Proof. At line 5 of Algorithm 2, $w(\mu) = |S| + |C_r|$ because it was just output by the max-weight algorithm. So, we have to evaluate the for loop that compares each student of some type to each other student of that same type. In the for loop, the students swap seats if they both prefer each other's match in μ. This means that each student still prefers their new match, so the weight of the edge retains weight at least 1 from that student. Then, if either student was in a seat reserved for that type, we replaced them with a student of the same type, so the edge retains the additional weight of the reserved seat. Therefore, swapping any two students of the same type does not change the weight of μ. □

Theorem 0.6. *The WT algorithm will output a stable matching.*

Proof. Let μ be the matching output by the Max-Weight algorithm at line 4 of Algorithm 2. Observe lines 5 and 6 of Algorithm 2, which outline that within each student group, we must swap the assignments of blocking pairs until no more remain. We know this loop will terminate because of our assumption that preference lists are finite, so students will eventually reach their highest possible preference and no longer be able to form blocking pairs. The maximum length of a preference list is equivalent to the number of classes, $|C|$. So, the maximum number of times the while loop can execute per for loop is as follows:

$$|C| \times |S|$$

because at most we can traverse each student's entire preference list once. If we terminate this loop for each type of student $A \subset S$, then we must have intra-group stability because no more blocking pairs remain in each group. Therefore, the WT algorithm outputs a stable matching. □