Approximation Algorithms - Homework 1

Ilay Menahem, and Aaron Ross

December 23, 2025

1 Question 1

Question: Design a 2-approximation algorithm for the problem of finding a minimum cardinaliy maximal matching in an undirected graph.

Solution: we will use the following algorithm:
Input: An undirected graph $G = (V, E)$
Output: A maximal matching M of G
Algorithm:

1. Start with $M = \emptyset$
2. While there are edges remaining in the graph:
   (a) Pick an arbitrary edge $e = (u, v)$
   (b) $M = M \cup \{e\}$
   (c) Remove all edges adjacent to u or v from the graph

First we will prove that the algorithm returns a maximal matching M. Assume by contradiction that M is not maximal. Then there exists an edge $e = (x, y)$ such that neither x nor y are matched in M. But if that is the case, then this edge e would not have been removed from the graph during the algorithm, and thus there would have been a step in which we could have picked e and added it to M. This contradicts the assumption that the algorithm has terminated, and thus M must be maximal.

Lastly, we will prove that the algorithm runs in polynomial time. The algorithm iterates over all edges of the graph, and in each iteration it removes at least one edge from the graph. Thus, the algorithm does at most $|E|$

iterations, and assuming that removing edges can be done in $O(|E|)$ time, the total running time of the algorithm is $O(|E|^2)$, which is polynomial.

Now we will prove that the algorithm is a 2-approximation. Let M^* be an optimal maximal matching of G. We will show that $|M| \leq 2|M^*|$ by showing that for every one or two edges in M, there is at least one edge in M^* that is unique to these edges. Consider an arbitrary edge $e^* = (u, v) \in M^*$ If $e^* \in M$ we will match it to its incident in M. if $e^* \notin M$ there can be at most two edges in M that are adjacent to e^* (one adjacent to u and one adjacent to v). if there are adjacent edges from M, we will match these one or two edges e^*, there is also uniqueness due to the fact that in a matching, no two edges share an endpoint. Every edge or two edges in M is matched to a unique edge in M^*, since if an edge in M was not matched to an edge in M^*, it means that this edge was not in or adjacent to any edge in M^*, and thus M^* is not maximal, which is a contradiction.

2 Question 2

We will construct a bipartite graph $G = (L \cup R, E)$ as following. L consists of k vertices, such that $|L| = k$. For R we split the vertices into disjoint sets, such that each set consists of $\lfloor \frac{k}{i} \rfloor$ vertices where $i = 2, 3, \ldots, k$. In each set S_i , every vertex in that set has exactly $k \div \lfloor \frac{k}{i} \rfloor$ neighbors in L , such that, each vertex in L has exactly one neighbor from S_i.(when $\lfloor \frac{k}{i} \rfloor \in \mathbb{N}$ each vertex in S_i will have i neighbors).

The Algorithm
After construction each vertex in L will have a degree of exactly $k - 1$ and in R each vertex in S_i will have degree i or higher.
Therefore, the algorithm will choose in the first step all vertices from S_k (The vertex in S_k has degree k). This will result in vertices in L having degree $k - 2$. On the next step vertices from S_{k-1} will be choosen because they have degree $k - 1$ and so on. In general for step j vertices from S_{k-j-1} will be chosen because vertices in L will have a degree of $k - j - 2$. This will continue until R is empty, therefore the algorithm will choose all the vertices from R as the vertex cover.

In total we get a lower bound for the number of vertices the algorithm chooses

$$\sum_{i=2}^{k} \left\lfloor \frac{k}{i} \right\rfloor \geq \sum_{i=1}^{k} \left(\frac{k}{i} - 1 \right) - k = k \sum_{i=1}^{k} \frac{1}{i} - 2k \geq klogk - 2k$$

The optimal vertex cover is to choose all vertices in L, Therefore $|OPT| = k$

In total we get that the approximation ratio is:

$$\frac{klogk - 2k}{k} \in \Omega(\log(k))$$

note that $n \in \Theta(k \log(k))$ (n is the number of vertices), Therefore $\log(n) \in \Omega(\log(k \log(k))) = \Omega(\log(k) + \log(\log(k))) = \Omega(\log(k))$

3 Question 3

3.1 Tightness of the approximation

Question: Give a tight example in which $w(I) = H_n * OPT - \varepsilon$ for all $\varepsilon > 0$. (I is the cover that the greedy algorithm we saw in class outputs).

Solution: we will use the following example:

- Let $U = [n]$
- Let $S_i = \{i\}$ for $i = 1, 2, \ldots, n$, $S_{OPT} = [n]$
- Let $w(S_i) = \frac{1}{i} - \frac{\varepsilon}{n}$ for $i = 1, 2, \ldots, n$, and $w(S_{OPT}) = 1$

The optimal solution for this example is to take the cover $C^* = \{S_{OPT}\}$, this is the optimal solution since the only two covers without redundent sets are $C^* = \{S_{OPT}\}$ and $C = \{S_1, S_2, \ldots, S_n\}$, and the cost of C^* is $w(C^*) = 1$ while the cost of C is $w(C) = \sum_{i=1}^{n}(\frac{1}{i} - \frac{\varepsilon}{n}) = H_n - \varepsilon$. Thus the optimal solution is C^* with cost $OPT = 1$.

Now we will run the our greedy algorithm on this example. Let's show in induction that in step i of the greedy algorithm, the algorithm picks the set S_i which is the set with the minimum cost per newly covered elements. for the first step, the algorithm picks S_n since $\min_{S_i} \frac{w(S_i)}{\hat{S}_i} = \min_i \frac{\frac{1}{i} - \frac{\varepsilon}{n}}{1} = \frac{1}{n} - \frac{\varepsilon}{n}$ which is achieved for $i = n$ and it has a lower cost per newly covered elements than S_{OPT} which has cost per newly covered elements of $\frac{w(S_{OPT})}{n} = \frac{1}{n}$. Assume that in step k the algorithm picks S_{n-k}. In step $k+1$, the sets $S_{n-k+1}, S_{n-k+2}, \ldots, S_n$ have already been picked, and thus the only sets that can cover new elements are $S_{n-k-1}, S_{n-k-2}, \ldots, S_1$ and S_{OPT}. The cost per newly covered elements for these sets are:

- For S_{k+1}: $\frac{w(S_{k+1})}{1} = \frac{1}{k+1} + \frac{\varepsilon}{n}$
- For S_{OPT}: $\frac{w(S_{OPT})}{n-k} = \frac{1}{n-k}$

Since $\frac{1}{k+1} + \frac{\varepsilon}{n} < \frac{1}{n-k}$ for all $k < n$, the algorithm picks S_{k+1} in step $k+1$. Thus by induction, we have shown that in step i the algorithm picks S_i for all $i = 1, 2, \ldots, n$.

we got that the greedy algorithm picks the sets in the order $S_1, S_2, \ldots, S_n$. Thus the cost of the cover C returned by the greedy algorithm is:

$$w(C) = \sum_{i=1}^{n} w(S_i) = \sum_{i=1}^{n}(\frac{1}{i} - \frac{\varepsilon}{n}) = H_n - \varepsilon = H_n * OPT - \varepsilon$$

3.2 $SC \in NP - hard$

Question: Show that this variant of the Set Cover problem is NP-hard.

Solution: we will show that $VC_k \leq_P SC_k$. we will take the following reduction:

Input: An instance of VC_k: a graph $G = (V, E)$ and an integer k.

Output: weather G has a vertex cover of size at most k.

Reduction:

1. Let $U = E$
2. For each $v_i \in V$ let $S_i = \{e \in E | v_i \in e\}$
3. Let k be the same integer as in the input
4. Output the instance of SC_k defined by U, $S_1, S_2, \ldots, S_m$ and k.
5. Solve the instance of SC_k using an oracle, and mark the solution as C
6. Output true iff the $\cup_{S_i \in C} S_i = U$

First we will prove that the reduction is polynomial. This reduction requires us to iterate over all vertices and edges of the graph once, and thus it runs in $O(|V| + |E|)$ time, which is polynomial.

Now we will prove the correctness of the reduction by showing that G has a vertex cover of size at most k iff the instance of SC_k has a set cover of size at most k.

($\Rightarrow$) Assume that G has a vertex cover C of size at most k. then the sets $C' = \{S_i | v_i \in C\}$ form a set cover of size at most k for the instance of SC_k, since every edge $e \in E$ is covered by at least one vertex in $v_i \in C$, and thus $e \in S_i$ for at least one $S_i \in C'$.

($\Leftarrow$) Assume that the instance of SC_k has a set cover C' of size at most k. then the vertices $C = \{v_i | S_i \in C'\}$ form a vertex cover of size at most k for G, since every edge $e \in E$ is covered by at least one set $S_i \in C'$, and thus e is adjacent to at least one vertex $v_i \in C$.

3.3 $1 - 1/e$ approximation for Max-k-Coverage

Question: Give a $1 - 1/e$-approximation algorithm for this problem. Hint: recall that $(1 - \frac{1}{n})^n \leq \frac{1}{e}$. Note: for maximization problems, we say that an

algorithm is α-approximation for $\alpha < 1$ if the value of the solution returned by the algorithm is at least α times the value of the optimal solution.

Solution: we will use the following algorithm: **Input:**

- A universe $U = [n]$
- A collection of sets $S_1, S_2, \ldots, S_m$ where $S_i \subseteq U$, and $\cup_{i=1}^{m} S_i = U$
- An integer k

Output: A collection of sets C such that $|C| = k$ and the number of elements covered by C is maximized. **Algorithm:**

1. Start with $C = \emptyset$
2. Let $covered = \emptyset$
3. While $|C| < k$:
   (a) $C = C \cup \arg\max_{S_i} |S_i \setminus covered|$
   (b) $covered = covered \cup S_i$
   (c) Remove S_i from the collection of sets
4. Output C

This algorithm returns a cover of size k, now we will show that it runs in polynomial time. The algorithm iterates k times, and in each iteration it needs to

- Find the set S_i that covers the maximum number of uncovered elements, this can be done in $O(m * n)$ time.
- Update the covered elements, this can be done in $O(n)$ time.
- Remove S_i from the collection of sets, this can be done in $O(m)$ time.

Thus the total running time (not tight) of the algorithm is $O(k * m * n) = O(m * n^2)$ which is polynomial.

Now we will prove that the algorithm is a $1 - 1/e$ approximation. Let y_i be the number of elements covered by the greedy algorithm after i iterations, and we will note the number of new elements covered in iteration i as $x_i = y_i - y_{i-1}$.

after iteration i, The number of elements covered by the optimal solution but not yet covered by the greedy solution is at most $OPT - y_i$. By the pigeonhole principle, there must exist at least one set in the optimal solution that covers at least $\frac{OPT-y_i}{k}$ of these uncovered elements. Since the greedy algorithm chooses the set that maximizes the number of newly covered elements, the set chosen in iteration $i+1$ must cover at least as many new elements as that set from the optimal solution. Therefore, we have:

$$y_{i+1} - y_i \geq x_{i+1} \geq \frac{OPT - y_i}{k}$$

$$y_{i+1} \geq y_i + \frac{OPT}{k} - \frac{y_i}{k} = y_i(1 - \frac{1}{k}) + \frac{OPT}{k}$$

using that relation iteratively, and substituting $y_0 = 0$, we get:

$$y_k \geq y_{k-1}(1 - \frac{1}{k}) + \frac{OPT}{k} \geq y_{k-2}(1 - \frac{1}{k})^2 + \frac{OPT}{k}(1 - \frac{1}{k}) + \frac{OPT}{k} \geq \ldots \geq$$

$$y_{k-i}(1 - \frac{1}{k})^i + \frac{OPT}{k}\sum_{j=0}^{i-1}(1 - \frac{1}{k})^j \geq \ldots \geq y_0(1 - \frac{1}{k})^k + \frac{OPT}{k}\sum_{j=0}^{k-1}(1 - \frac{1}{k})^j$$

$$= \frac{OPT}{k}\sum_{j=0}^{k-1}(1 - \frac{1}{k})^j = \frac{OPT}{k}\frac{1 - (1 - \frac{1}{k})^k}{1 - (1 - \frac{1}{k})} = OPT(1 - (1 - \frac{1}{k})^k)$$

thus we have shown that the greedy algorithm covers at least $OPT(1 - \frac{1}{e})$ elements, and thus it is a $1 - \frac{1}{e}$ approximation.

4 Question 4

(a)

Given a graph $G = (V, E)$ with terminals R, and T an MST of $G[R]$ we will show that:

$$cost(T) \leq 2|OPT|$$

We look at an optimal solution T^* which is constructed of $e \in E(G[R]) \cap T^*$, $e \in C = \{(u, v) \mid u \in R,\ v \in V \backslash R\} \cap T^*$and $e \in E(G[V \backslash R]) \cap T^*$. We will construct a new Graph $G' \subseteq G[R]$ from T^*, such that $e \in E(G') \rightarrow e \in E(G[R])$ by replacing edges in C with edges in $E(G[R])$ And removing edges in $G[V \backslash R]$ (there cost is > 0). For edges $(u_1, v_1), (u_2, v_2), \ldots, (u_k, v_k) \in C$, Starting with u_1 we will find the first u_i such that the path between u_1 to u_i goes through $(u_1, v_1), (u_i, v_i)$ and remove (u_1, v_1) and replace it with $(u_1, u_i) \in G[R]$. We will continue now starting with (u_i, v_i) and replacing it with (u_i, u_k) For the first k that upholds the above, if no such k exists we will remove the edge and move on to (u_{i+1}, v_{i+1}) (because the edge doesn't contribute to any path), and continue until C is empty. Note that if $i+1 > k$ we will loop back to $i = 1$.

We will First prove that G' is connected.

Every swap of edges the algorithm does, makes no change to the reach of every vertex. For instance if $(u_i, v_i) \in C$ was swapped with $(u_i, u_j) \in E(G[R])$ There still exists a path $P \subseteq E(G')$ between u_i and every $u \in R$ because any path that went through (u_i, v_i) we can replace with a path that goes through (u_i, u_j) and then the path from u_j to v_i (because at that point in the construction of G' there is a path between v_i and u_j). Therefore, every time we replace an edge G' stays connected.

Now Since G' has edges only in R and is connected, it holds that:

$$cost(G') \geq (T)$$

Also since every $e_{T^*} \in C$ is replaced with exactly one edge $e_{T'} \in G[R]$ During the replacement step, we note that every edge $e_i = (u_i, u_k)$ that replaces an edge $e_i^* = (u_i, v_i) \in C$ holds the following:

$$w(e_i) \leq w(e_i^*) + w(e_k^*)$$

where e_k^* is the next edge(if exists) that will be replaced. Also e_i^* can contribute at most to two of these bounds, once when it is swapped and once more the iteration before it is swapped when another edge chooses it(because that means it will definitely be chosen and removed in the next iteration). Therefore, it holds that:

$$\sum_{e \in E(G') \setminus T^*} w(e) \leq \sum_{e \in C} 2w(e)$$

And in total we get:

$$cost(T) \leq \sum_{e \in E(G')} w(e) = \sum_{e \in E(T^*) \cap E(G')} w(e) + \sum_{e \in E(G') \setminus T^*} w(e)$$

$$\leq \sum_{e \in T^* \cap E(G')} w(e) + \sum_{e \in C} 2w(e)$$

$$\leq \sum_{e \in T^* \cap E(G')} 2w(e) + \sum_{e \in C} 2w(e) \leq \sum_{e \in T*} 2w(e) \leq 2|OPT|$$

(b)

First we note that from the previous question $cost(\widetilde{T}) \leq 2|OPT(\widetilde{G})|$. This is because $\widetilde{G}$ is complete and the weight function of the shortest path satisfies the triangle inequality.
Also $|OPT(G)| \geq |OPT(\widetilde{G})|$ because every edge in $\widetilde{G}$ is at most the same cost of its corresponding edge in G, Therefore the cost of any tree in $\widetilde{G}$ is smaller than the same tree in G.
$cost(T) \leq cost(\widetilde{T})$ because they have the exact same weight before removal of edges in T, Therefore we get the following.

$$cost(T) \leq cost(\widetilde{T}) \leq 2|OPT(\widetilde{G})| \leq 2|OPT(G)|$$