

Approximation Algorithms - Homework 1

Ilay Menahem, and Aaron Ross

November 29, 2025

1 Question 1

Question: Design a 2-approximation algorithm for the problem of finding a minimum cardinality 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

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 $i = 1, 2, \dots, n$, $S_{OPT} = [n]$
- Let $w(S_i) = \frac{1}{i} - \frac{\varepsilon}{n}$ for $i = 1, 2, \dots, n$, 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 redundant sets are $C^* = \{S_{OPT}\}$ and $C = \{S_1, S_2, \dots, S_n\}$, 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 $\min_{S_i} \frac{w(S_i)}{|S_i|} = \min_i \frac{\frac{1}{i} - \frac{\varepsilon}{n}}{1} = \frac{1}{n} - \frac{\varepsilon}{n}$ 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 $S_{n-k+1}, S_{n-k+2}, \dots, S_n$ have already been picked, and thus the only sets that can cover new elements are $S_{n-k-1}, S_{n-k-2}, \dots, S_1$ 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 $i = 1, 2, \dots, n$.

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

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

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: whether 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, \dots, 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, \dots, 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:

$$y_k \geq y_{k-1}(1 - \frac{1}{k}) + \frac{OPT}{k} \geq y_{k-2}(1 - \frac{1}{k})^2 + \frac{OPT}{k}(1 - \frac{1}{k}) + \frac{OPT}{k} \geq \dots \geq y_{k-i}(1 - \frac{1}{k})^i + \frac{OPT}{k} \sum_{j=0}^{i-1} (1 - \frac{1}{k})^j \geq \dots \geq y_0(1 - \frac{1}{k})^k + \frac{OPT}{k} \sum_{j=0}^{k-1} (1 - \frac{1}{k})^j$$

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

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