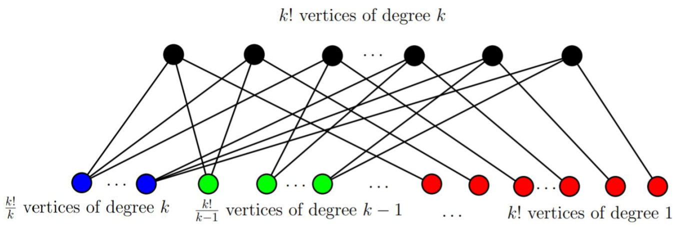
School of Computer Science and Technology
University of Science and Technology of China

Exercise Sheet 3 for
Design and Analysis of Algorithms
Autumn 2022
Solution

Exercise 1 (30 points, graded by Yinhao Dong)

Consider the algorithm AnotherGreedyVC given in Lecture 8 for the minimum vertex cover problem. Use the following example to show that the approximation ratio of AnotherGreedyVC is $\omega_n(1)$, where n is the number of vertices in the input graph.



Solution. The vertices picked by the algorithm form a vertex cover, denoted as C. In the example, the algorithm could possibly pick all the bottom vertices from left to right in order. Hence $|C| = k!(\frac{1}{k} + \frac{1}{k-1} + \cdots + 1) \approx k! \log k$. However, the optimal cover will pick the $k!$ vertices at the top. Hence $\mathrm{OPT} = k!$. Note that in the example, $n = k! + k!(\frac{1}{k} + \frac{1}{k-1} + \cdots + 1) \approx k!(1 + \log k)$. Therefore, the approximation ratio of AnotherGreedyVC is *at least* $\frac{|C|}{\mathrm{OPT}} = \log k = \omega_n(1)$, which becomes arbitrarily large as n approaches infinity.

Exercise 2 (30 points, graded by Di Wu)

Consider the set cover problem. Let U be a set of n elements. Let $\mathcal{S} = \{S_1, \dots, S_m\}$ be a collection of subsets of U such that $\cup_{i=1}^m S_i = U$. Our goal is to select as few subsets as possible from $\mathcal{S}$ such that their union covers U.
Consider the following algorithm SetCover for this problem. The algorithm takes as input U and $\mathcal{S}$, and does the following:

(a) Initialize $C = \emptyset$.

(b) While U contains elements not covered by C:

  (i) Find the set S_i containing the greatest number of uncovered elements
  (ii) Add S_i to C.

To analyze the above algorithm, let $k = \mathrm{OPT}$ be the number of sets in the optimal solution. Let $E_0 = U$ and let E_t be the set of elements not yet covered after step t.

(a) **(15 points)** Show that $|E_{t+1}| \le |E_t| - |E_t|/k$.

(b) **(15 points)** Show that the algorithm SetCover is a $(\ln n)$-approximation algorithm for the set cover problem.

Hint: Show that the algorithm SetCover finishes within $\mathrm{OPT} \cdot \ln n$ steps.

Proof.

(a) Since $k = \text{OPT}$ is the number of sets in the optimal solution, the optimal solution covers every E_t with no more than k sets. In step $t+1$, the algorithm always picks the *largest* set over E_t in $\mathcal{S}$. The size of this largest set must cover at least $|E_t|/k$ in E_t; if it covered fewer elements, no way of picking sets would be able to cover E_t in k sets, which contradicts the existence of OPT. So $|E_t| - |E_{t+1}| \geq |E_t|/k \implies |E_{t+1}| \leq |E_t| - |E_t|/k$.

(b) (i) **Running time:** Each step takes $O(mn)$ time, and we will next show that the algorithm finishes within $\text{OPT} \cdot \ln n$ steps. Therefore, the algorithm runs in polynomial time.

(ii) **Correctness:** According to (a) and $|E_0| = |U| = n$, we have that $|E_t| \leq n\left(1 - \frac{1}{k}\right)^t$ for any $t \geq 1$. (This can be proven by induction: $|E_1| \leq |E_0| - \frac{|E_0|}{k} = n\left(1 - \frac{1}{k}\right), |E_2| \leq |E_1| - \frac{|E_1|}{k} \leq n\left(1 - \frac{1}{k}\right)^2$, and so on.) Consider $t = \text{OPT} \cdot \ln n = k \ln n$, we have $|E_t| \leq n\left(1 - \frac{1}{k}\right)^{k \ln n} < n \cdot \left(\frac{1}{e}\right)^{\ln n} = 1$, which implies that the algorithm finishes within $\text{OPT} \cdot \ln n$ steps. Therefore, $|C| \leq (\ln n)\text{OPT}$.

Therefore, the algorithm SetCover is a $(\ln n)$-approximation algorithm for the set cover problem.

□

Exercise 3 (40 points, graded by Yudong Zhang)

Consider the max cut problem. Given an undirected n-vertex graph $G = (V, E)$ with positive integer edge weights w_e for each $e \in E$, find a vertex partition $(A, \bar{A})$ such that the total weight of edges crossing the cut is maximized, where $\bar{A} = V \setminus A$ and the weight of $(A, \bar{A})$ is defined to be $w(A, \bar{A}) := \sum_{u \in A, v \in \bar{A}} w_{u,v}$. Consider the following algorithm MaxCut.

(a) Start with an arbitrary partition of V.

(b) Pick a vertex $v \in V$ such that moving it across the partition would yield a greater cut value.

(c) Repeat step (b) until no such v exists.

Now analyze the performance guarantee of the algorithm.

(a) **(10 points)** Suppose that the maximum edge weight is $\lceil n^{10} \rceil$. Show that the algorithm runs in polynomial time.

(b) **(15 points)** Let $(S, \bar{S})$ be partition output by the algorithm MaxCut. Show that for any vertex $v \in S$, it holds that

$$\sum_{u \in \bar{S}, (u,v) \in E} w_{u,v} \geq \frac{1}{2} \sum_{u:(u,v) \in E} w_{u,v}.$$

(c) **(15 points)** Show that the algorithm MaxCut is a 1/2-approximation algorithm for the max cut problem.

Hint: Use the fact that $\sum_{e \in E} w_e \geq \text{OPT}$, where OPT is the total weight of the optimal solution.

Proof.

(a) Each iteration involves examining at most n vertices and selecting one that increases the cut value when moved. This process takes $O(n^2)$ time. Since we assume that edges have positive integer weights, the cut value is increased by at least 1 after each iteration. The maximum possible cut value is $\sum_{e \in E} w_e = O(n^2 \cdot n^{10}) = O(n^{12})$, hence there are at most such number of iterations. The overall time complexity of this algorithm is thus $O(n^2 \cdot n^{12}) = O(n^{14})$, which is polynomial in the input size.

(b) We say that an edge *contributes* to a cut if its endpoints lie in different subsets of the cut. For any vertex $v \in S$, consider the set E_v of edges incident to v. If we move v from S to $\bar{S}$, edges in E_v that contributed to the cut become non-contributing, and vice versa. Edges not in E_v are not affected.
Since $(S, \bar{S})$ is partition output by the algorithm, the cut value is a local optimum, and moving v to $\bar{S}$ does not yield a greater cut value. Therefore, for any vertex $v \in S$, at the time of termination we have

$$\sum_{u \in \bar{S}, (u,v) \in E} w_{u,v} \geq \sum_{u \in S, (u,v) \in E} w_{u,v} \tag{1}$$

(If the above inequality does not hold, moving v to $\bar{S}$ will yield a greater cut value.)

$$2 \sum_{u \in \bar{S}, (u,v) \in E} w_{u,v} \geq \sum_{u \in S, (u,v) \in E} w_{u,v} + \sum_{u \in \bar{S}, (u,v) \in E} w_{u,v} = \sum_{u:(u,v) \in E} w_{u,v} \tag{2}$$

$$\sum_{u \in \bar{S}, (u,v) \in E} w_{u,v} \geq \frac{1}{2} \sum_{u:(u,v) \in E} w_{u,v}. \tag{3}$$

(c) (i) **Running time:** According to (a), the algorithm runs in polynomial time.

(ii) **Correctness:** Similar to (b), for any vertex $v' \in \bar{S}$, we have

$$\sum_{u \in S, (u,v') \in E} w_{u,v'} \geq \frac{1}{2} \sum_{u:(u,v') \in E} w_{u,v}. \tag{4}$$

Adding up (3) for all vertices in S, we have:

$$w(S, \bar{S}) = \sum_{v \in S} \left(\sum_{u \in \bar{S}, (u,v) \in E} w_{u,v} \right) \geq \frac{1}{2} \sum_{v \in S} \left(\sum_{u:(u,v) \in E} w_{u,v} \right) \tag{5}$$

Adding up (4) for all vertices in $\bar{S}$, we have:

$$w(S, \bar{S}) = \sum_{v' \in \bar{S}} \left(\sum_{u \in S, (u,v') \in E} w_{u,v'} \right) \geq \frac{1}{2} \sum_{v' \in \bar{S}} \left(\sum_{u:(u,v') \in E} w_{u,v} \right) \tag{6}$$

Adding up (5) and (6), we obtain the desired result:

$$2w(S, \bar{S}) \geq \frac{1}{2} \left(2 \cdot \sum_{e \in E} w_e \right) = \sum_{e \in E} w_e \geq \text{OPT}$$

$$w(S, \bar{S}) \geq \frac{1}{2} \text{OPT}.$$

Therefore, the algorithm MaxCut is a 1/2-approximation algorithm for the max cut problem.

□