CS6841: Approximation Algorithms

Assignment 1

Name: E Santhosh Kumar

Roll number: CS16B107

Instructions

- **Deadline**: 20 Feb (in class during Midsem)
- **References:** Williamson and Shmoys (`https://www.designofapproxalgs.com/book.pdf`), Approximation Algorithms by Vijay Vazirani, CLRS

1. (5 points) **k-suppliers Problem** The input to the problem is a positive integer k, and a set of vertices V. Let $d(u, v)$ represents the shortest path between any two vertices $u, v \in V$ and obey the same properties as in the Facility Location Problem. The vertices are partitioned into suppliers $F \subseteq V$ and customers $C = V \setminus F$.

   The goal is to find k suppliers such that the maximum distance from a supplier to a customer is minimized. In other words, the problem is to find $S \subseteq F$, $|S| \leq k$, that minimizes $\max_{j \in C} d(j, S)$.

   (a) (2.5 points) Give a 3-approximation algorithm for the k-suppliers problem.

   (b) (2.5 points) Prove that there is no α-approximation algorithm for $\alpha < 3$ unless P = NP.

Solution:

(a) For any customer $c_i \in C$, let $f_i \in F$ denote its nearest supplier (can be found in polynomial time). Also, let OPT denote the optimal solution value to the problem. Then, by problem definition, we have

$$d(c_i, f_i) \leq OPT \ \ \forall c_i \in C$$

We have already seen an approximation algorithm for the Facility Location Problem in class. We proved that this was a 2-approximation algorithm. We use this algorithm to define an algorithm for the given k-suppliers problem as follows

- Case 1 - if $k \geq |C|$: simply return $S = \{f_i | c_i \in C\}$ (you can add other random suppliers to this set if size has to be exactly k).
- Case 2 - if $k < |C|$: Let G_C be the graph obtained by considering only the consumer vertices of G. Let C^* be the set of chosen consumers obtained on applying the facility location problem on G_C (with same k). We return $S = \{f_i | c_i \in C^*\}$.

We now show that this is a 2-approximation algorithm.

- In case 1, we see that S is one of the optimal solutions as per problem definition.
- Consider any $v \in C$ and let c_j be the vertex closest to it in C^*. If $d(v, c_j) \leq 2OPT$, then $d(v, f_j) \leq d(v, c_j) + d(c_j, f_j) \leq 2OPT + OPT = 3OPT$. Thus it is a 3-approx. We now show that $d(v, c_j) \leq 2OPT \ \ \forall v \in C$.

  Proof by contradiction. Assume there exists a $v \in C$ such that $d(v, c_i) > 2OPT \ \ \forall c_i \in C^*$. Let c_x and c_y be any two vertices in C^* such that c_x was chosen before c_y during the run of the facility location algorithm. Then, in the iteration when c_y was chosen (let the already chosen vertices at this point be S_2), c_y was chosen over v. Thus as per algorithm,

$$d(v, S_2) \leq d(c_y, S_2) \implies \min_{c_p \in S_2} d(v, c_p) \leq \min_{c_p \in S_2} d(c_y, c_p) \implies \min_{c_p \in S_2} d(v, c_p) \leq d(c_y, c_x)$$

$$\implies 2OPT < d(c_y, c_x)$$

  Thus any 2 vertices in C^* are at least distance $2OPT+1$ apart. Also, vertex v is at least distance $2OPT + 1$ apart from all vertices of C^*. We now have $k + 1$ vertices that are distance at least $2OPT+1$ from each other. Thus we cannot get an optimal value of OPT for the facility location problem on C with k facilities. Hence, contradiction. Hence proved.

(b) We prove this by reduction from the known NP-complete problem - Dominating Set Problem (Given a graph G(V,E) and a positive integer k, decide if there is a subset $S \subseteq V$ of size k such that every vertex in V is either present in S or is adjacent to a vertex in S.)
Given an instance $G(V, E)$ of the dominating set problem, we convert it to an instance $G_2(V_2, E_2)$ of the k-suppliers problem as follows:- For every vertex $u \in V$, create 2 vertices f_u, c_u in V_2, where $f_u \in F$ is a supplier and $c_u \in C$ is a customer. The distances between the vertices are given as

- $d(f_u, f_v) = 2$ and $d(c_u, c_v) = 2$ forall pairs u,v.
- $d(f_u, c_v) = 1$ if $u = v$ or $(u, v) \in E$(u and v are adjacent in G). Else, $d(f_u, c_v) = 3$.

We can verify that these distances satisfy the triangle inequality. We observe that any solution to the k-supplier problem has value either 1 or 3.

- If there exists a dominating set S of size k for G, the corresponding supplier set $S_2 = \{f_u | u \in S\}$ to the k-supplier problem has value 1 (as for each c_v, either $v = u$ or v is adjacent to u for some u in S).
- Similarly, let optimal value for the k-supplier problem be 1 and let one such solution be S_2. Then for all c_v in C, there exists a f_u in S_2 such that $d(f_u, c_v) = 1$. Thus the corresponding set $S = u | f_u \in S_2$ is a dominating set of G.

Thus G has a dominating set if and only if the corresponding k-facility problem has an optimal solution of 1. Let us now assume we have an α-approx algorithm A for k-facility problem with $\alpha < 3$. Then whenever the problem I has $OPT(I) = 1$, we have $A(I) \leq \alpha OPT(I) = \alpha < 3$. Thus, whenever $OPT(I) = 1$, we have $A(I) = 1$ (as we earlier observed that A(I) can only be 1 or 3). Thus $A(I) = 1$ iff $OPT(I) = 1$. Hence, we have a polynomial time algorithm for the dominating set problem. But,

we know that dominating set is an NP-complete problem. Hence, $P = NP$. Thus there is no α-approximation algorithm with $\alpha < 3$ unless P = NP.

2. (5 points) **Job-Scheduling with Precedence** This problem is a variant of the job scheduling (in multiple processors) problem, that was discussed in class. Here, along with the n jobs and m processors, we also have precedence constraint between the jobs.

   Given are n jobs $\mathcal{J} = \{j_1, j_2, \cdots, j_n\}$, with processing times $t_1, t_2, \cdots, t_n$ and m processors $P_1, P_2, \cdots, P_n$. The precedence constraints between jobs is given in the form of a partial order $(\mathcal{J}, <)$ on the jobs i.e. if $j < j'$, then job j has to be completed before job j' has started. The problem is to find the shortest schedule that satisfies the precedence constraints.

   (a) (2.5 points) Prove that the naive job scheduling algorithm (first algorithm discussed in class) yields a 2-approximation algorithm for this problem.

   (b) (2.5 points) For any general value of n and m, give an instance to prove that $2.OPT$ is a tight bound. Assume $t_i = 1 \forall i \in [1, n]$.

Solution:

(a) Let OPT be the optimal/minimal time required to schedule all these jobs. Let A be the total time taken by the naive scheduling algorithm. This can be split as $A = B + C$, where B is the time duration for which all processors are busy and C is the time duration for which at least one and at most $m-1$ processors are free. We get bounds on each of these now.

- Since each processor is busy for the time duration B, we have

$$\sum_{i=1}^{n} t_i \geq Bm$$

$$\implies B \leq \sum_{i=1}^{n} t_i/m \leq OPT$$

- The precedence conditions among the jobs can be used to creates chains of the form $j_{x_1} < j_{x_2} < \cdots < j_{x_k}$ where every two consecutive chain elements are arranged by partial ordering. Let Q be the maximum sum of times of jobs of one such chain. Since OPT has to be longer than all such chains, we have $Q \leq OPT$.

  Now, let there be times $t_1 < t_2 < t_3 < A$. Let time interval from t_1 to t_2 be such that at least one processor is free. Also, let job j_x be scheduled at time t_3. Then, it implies there exists a job j_y with $j_y < j_x$ that has been running throughout the interval t_1 to t_2. Based on this argument, there exists a chain whose run-time is longer than the duration C. Thus, $Q \geq C$. Thus, $OPT \geq Q \geq C$.

$$\implies A = B + C \leq OPT + OPT \leq 2OPT \implies \frac{A}{OPT} \leq 2$$

Thus it is a 2 approximation algorithm.

(b) Consider the problem with given n,m and $t_i = 1\ \forall i = 1, \ldots, n$. Let $k = \lceil \frac{n}{m} \rceil$. Let the precedence conditions of the problem be $j_i < j_{i+1}\ \ \forall i = 1, \ldots, k-1$. That is,

$$j_1 < j_2 < \cdots < j_k$$

Consider figure 1 for an example

- Since there is only one chain of precedence and we can pack all other jobs perfectly, we have $OPT = k$.
- One worst case for the order in which our algorithm can schedule the jobs is as follows :- $j_{k+1}, j_{k+2}, \ldots, j_n, j_1, j_2, \ldots, j_k$. This is a valid ordering whose time taken would be

$$A = \left\lfloor \frac{n-k}{m} \right\rfloor + k$$

When assuming that all divisions are exact integers so that floor and int return exact values, we have

$$\frac{A}{OPT} = \frac{\frac{n-k}{m} + k}{k} = \frac{\frac{n}{k} - 1}{m} + 1 = \frac{m-1}{m} + 1 = 2 - \frac{1}{m}$$

As m increases, the approximation ratio gets closer in the limit to 2. Hence it is a tight bound in the limit

3. (5 points) **Vertex Cover – Highest Degree Selection** Consider Algorithm 1 as the solution for the vertex cover problem. Prove that the algorithm does NOT give a constant factor approximation for the vertex cover problem.

Algorithm 1: Algorithm for question 3

Input: Graph $G = (V, E)$

Output: Vertex Cover $S \subseteq V$

$S \leftarrow \phi$

while $\exists e \in E(G)$ **do**

  Remove the vertex v with the highest degree

  Add it to the vertex cover S

end

return S

Solution: For any natural number n, consider the following example of a bipartite graph: $G_n = (L \cup R, E)$, where L is the first partite and has n vertices, and the second partite R is given by $R = \bigcup_{i=2}^{n} R_i$. Each R_i is a set of $\lfloor \frac{n}{i} \rfloor$ vertices that have a degree of i, such that each of them are connected to distinct vertices at L.

The given algorithm runs on the graph as follows

- Initially all vertices in L have degree at most $n-1$ (as each of them is connected to at most one

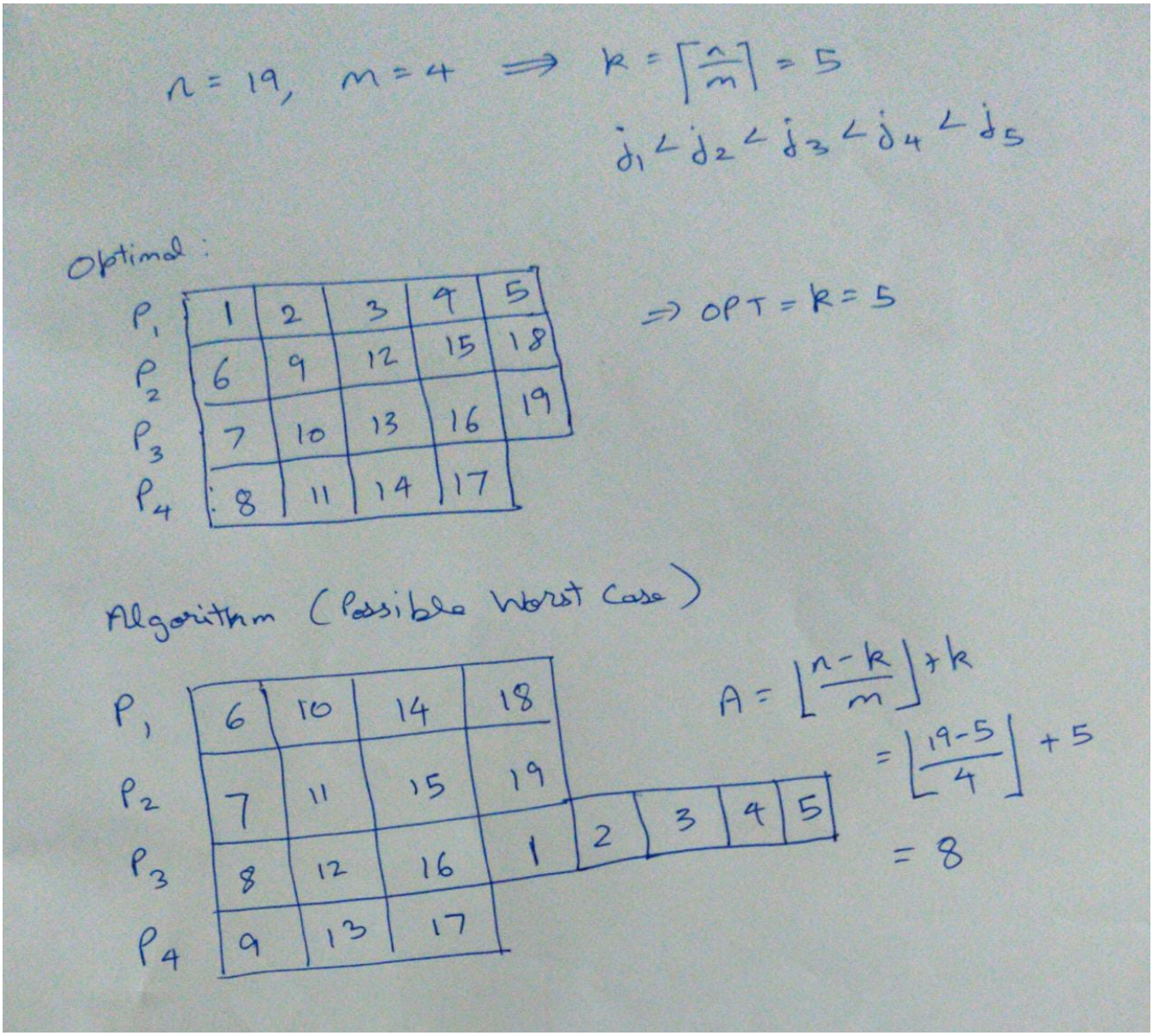


Figure 1: Q2

vertex is each R_i). Hence the algorithm first removes the single vertex in R_n (as it has degree n).

- Now vertices in L have degree at most $n-2$ (as only vertices of R_i, $i = 2, \ldots, n-1$ are left in R). Among the vertices in R, the vertices of R_{n-1} have maximum degree of $n-1$. Hence these are removed.
- Now vertices in L have degree at most $n-3$ (as only vertices of R_i, $i = 2, \ldots, n-2$ are left in R). Among the vertices in R, the vertices of R_{n-2} have maximum degree of $n-2$. Hence these are removed.
- Continuing similarly, we see that the algorithm removes vertices in the order $R_n, R_{n-1}, \ldots, R_2$. That

is, it removes all vertices of R. Thus the total number of vertices removed is given by

$$A(G_n) = |R| = \sum_{i=2}^{n} |R_i| = \sum_{i=2}^{n} \left\lfloor \frac{n}{i} \right\rfloor \geq \sum_{i=2}^{n} \left(\frac{n}{i} - 1) \right) \geq n \sum_{i=1}^{n} \frac{1}{i} - 2n = n(H_n - 2)$$

where $H_n = \sum_{i=1}^{n} \frac{1}{i}$ is the n^{th} Harmonic number and has the asymptotic bound $\mathcal{O}(\log n)$.

Now, for the same graph G_n, we see that one valid vertex cover is the set L. Hence, $OPT(G_n) \leq |L| = n$. Thus the approximation ratio is given by

$$\frac{A(G_n)}{OPT(G_n)} \geq \frac{|R|}{|L|} = \frac{n(H_n - 2)}{n} = \mathcal{O}(\log n)$$

Thus the algorithm does not give a constant factor approximation for vertex cover.

4. (5 points) Show that the following algorithm is a 2-approximation for unweighted vertex cover:

   1. Perform a DFS
   2. Return all internal vertices

   Is this true for BFS?

Solution:

(a) DFS

We prove the result for a single component graph. In order to extended it to a multi-component graph, simply apply to the proof to each component separately and then aggregate the sum on both sides. Let $G(V, E)$ be the given graph with a possible DFS tree T. Let L be the set of leaves (non-internal nodes) of T.

- Any edge $e \in E$ of the graph is an edge between an ancestor node and a descendant node in the corresponding DFS Tree T (i.e, there can't exist an edge (u, v) in G where u is not an ancestor of v and also v is not an ancestor of u). Thus there are no edges in G that connect two leaves of T. Thus any edge of G has at least one end-point in $V - L$. **Hence, $V - L$ is a vertex cover**.
- (a) Any vertex cover of G is also a vertex cover of T.

  (b) Consider the following observations about an optimal vertex cover VC of a tree with n vertices (and hence $|E_t| = n - 1$ edges). For a tree we can assume that none of the leaves are in the optimal VC, it is always more beneficial to take the parent if a leaf is in the cover. We also have $\sum_{u \in VC} deg(u) \geq |E_t|$ (as it covers every edge) and $\sum_{u \in VC} deg(u) + \sum_{u \notin VC} deg(u) = 2|E_t|$ (degree in the tree). Hence we have $\sum_{u \notin VC} deg(u) \leq |E_t|$.

$$\implies \sum_{u \in L} deg(u) + \sum_{u \notin L, u \notin VC} deg(u) \leq |E_t|$$

$$\implies \sum_{u \in L} 1 + \sum_{u \notin L, u \notin VC} 2 \leq n - 1 \ \ (degree\ of\ non-leaves\ is\ at\ least\ 2)$$

$$\implies |L| + 2(n - |L| - VC) \leq n - 1$$
$$\implies VC \geq \frac{n - |L| + 1}{2}$$

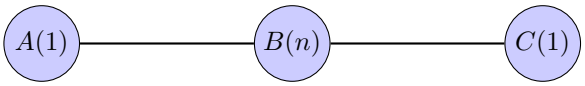
(c) The final statement in (b) is applicable for the optimal VC of a tree. Hence, it is applicable for any VC of a tree. Thus, using (a), it is applicable for any VC of G. But, $n - |L| = |V - L| = A(G)$.

$$\implies OPT(G) \geq \frac{A(G) + 1}{2} \implies \frac{A(G)}{OPT(G)} \leq 2$$

(d) BFS: The set of internal vertices of a BFS tree need not always be a VC of the original graph. A counter-example is a square graph ABCD with edges AB, BC, CD and DA. One possible BFS tree you get when you start from vertex A has the edges AB, AD and BC. Vertices C,D are leaves and the algorithm returns A,B which is not a VC.

5. (5 points) Unlike the unweighted case, show that the maximal matching may not lead to a constant factor approximation for the weighted vertex cover problem.

Solution: Consider the following graph G_n with $V = A, B, C, D$. The values in the brackets represent the weights of the vertices.



- The maximal matching based approximation algorithm chooses either edge AB or BC and picks VC to be both end-points of the edge. Thus, $A(G_n) = n + 1$.
- The optimal vertex cover (for $n \geq 1$) is $\{A, C\}$. Hence, $OPT(G_n) = 1 + 1 = 2$.
- Thus the approximation ratio is $\frac{A(G_n)}{OPT(G_n)} = \frac{n+1}{2}$ and is not upper-bounded by a constant.

6. (5 points) Show that the $\log n$-approximation factor is tight for the greedy set cover algorithm by buildling a worst case example.

Solution: (Refer figure 2) For any given natural number n, consider the following set X_n with $2(2^n - 1) = 2^{n+1} - 2$ elements as follows. (let $m = 2^n - 1$)

$$X_n = \{a_1, a_2, \ldots, a_m, b_1, b_2, \ldots, b_m\}$$

The collection of possible subsets $C = \{Q, R, S_0, S_1, \ldots, S_{n-1}\}$ where

$$Q = \{a_1, a_2, \ldots, a_m\}$$

$$R = \{b_1, b_2, \ldots, b_m\}$$

$$S_i = \{a_{2^i}, a_{2^i+1}, \ldots, a_{2^{i+1}-1}, b_{2^i}, b_{2^i+1}, \ldots, b_{2^{i+1}-1}\}$$

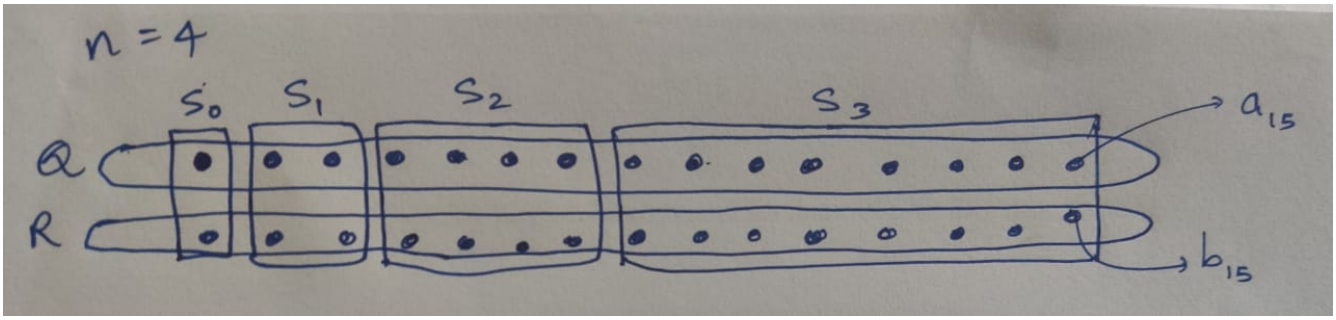


Figure 2: Q6

Thus, we have $|Q| = |R| = m = 2^n - 1$ and $|S_i| = 2^{i+1}$, $\forall i = 0, \ldots, n-1$. It is clear that the optimal set cover is given by $\{Q, R\}$. Thus, $OPT(X_n) = 2$. The greedy algorithm however runs as follows.

- In the first iteration, no elements of any subset are covered already. Q and R have $2^n - 1$ elements remaining each, while S_{n-1} has 2^n elements remaining (maximum). Hence S_{n-1} is chosen.
- Among the remaining subsets, Q and R have $2^{n-1} - 1$ elements remaining each, while S_{n-2} has 2^{n-1} elements remaining (maximum). Hence S_{n-2} is chosen.
- Among the remaining subsets, Q and R have $2^{n-2} - 1$ elements remaining each, while S_{n-3} has 2^{n-2} elements remaining (maximum). Hence S_{n-2} is chosen.
- The algorithm continues till we finally choose subset S_0 is chosen. No more elements remain.

Thus the algorithm chooses $\{S_0, S_1, \ldots, S_{n-1}\}$. Thus $|A(X_n)| = n$. Given that $|X_n| = 2^{n+1} - 2$, we have approximation ratio as

$$\frac{A(X_n)}{OPT(X_n)} = \frac{n}{2} = \frac{\log(|X_n| + 2) - 1}{2} = \mathcal{O}(\log n)$$

Hence it is a tight bound.