Assignment 1

Problem 1 (15 points) In the MAX-COVERAGE problem, we are given m sets $S_1, \cdots S_m$, each a subset of a universe U. We are also given an integer parameter $k \geq 1$. The objective is to pick k out of these m sets, indexed by I with $|I| = k$ such that $\bigcup_{j \in I} S_j$ is maximized. Describe a natural greedy algorithm for this problem and prove that is an $(1 - 1/e)$-approximation.

Given $U = \{e_1, e_2, \cdots, e_n\}$

$\mathcal{F} = \{S_1, S_2, \cdots, S_m\}$ and $k \geq 1$

Greedy Algo

```
Greedy_Max-Coverage(U, F, k):
    X ← U
    I ← ϕ
    while k > 0:
        pick S_j s.t. |S_j ∩ X| is max
        X ← X - S_j
        I ← I ∪ {j}
        k = k - 1
    return I
```

To Prove: $(1 - 1/e)\,OPT \leq T$

Notations:

OPT = no. of elements covered by the optimal solution

T = no. of elements covered by greedy

n_j = no. of elements covered in the j^{th} iteration

x_j = no. of elements remaining to be covered at the beginning of the j^{th} iteration

Z_j = no. of elements covered after j rounds

$x_{j+1} = x_j - n_j$ and $n_j \geq \frac{x_j}{k} \geq \frac{OPT - \sum_{i=0}^{j-1} n_i}{k}$ — (1)

[no. of elements left to be covered from $U \geq$ no. of ——— from OPT since $OPT \leq |U|$]

Also, $x_j \leq n(1 - 1/k)^{j-1}$ — (2) (from SET-COVER)

$Z_j = |U| - x_{j+1}$ (n)

$= n - x_{j+1}$

$\leq n - n(1 - 1/k)^j = n\left(1 - (1 - 1/k)^j\right)$ (using (2)) — (3)

Also, $Z_j = \sum_{i=1}^{j} n_i$

$\geq \sum_{i=1}^{j} \frac{OPT - Z_{i-1}}{k}$

$\geq \sum_{i=1}^{j} \frac{OPT - n[1 - (1-1/k)^{i-1}]}{k}$ (using (3))

$= \frac{j}{k} OPT - \frac{n}{k}\left[j - \sum_{i=1}^{j} (1-1/k)^{i-1}\right]$

Replacing j with k,

$Z_k \geq OPT - \frac{n}{k}\left[k - \left[\frac{1 - (1-1/k)^k}{1 - (1-1/k)}\right]\right]$ (sum of GP)

$$\geq OPT - \frac{n}{k}\left[k + k(1 - 1/e)\right]$$

$$\geq OPT - \underbrace{n}_{\geq OPT}\left[\underbrace{2 - 1/e}_{\geq 1/e}\right]$$

$$\geq OPT - OPT \cdot 1/e \qquad \geq (1 - 1/e)\, OPT$$

$$\Rightarrow \boxed{Z_k (= T) \geq (1 - 1/e)\, OPT}$$

Problem 2 (15 points) We consider the makespan minimization problem with an additional constraint. The basic setting is the same - you have a set of n jobs with processing times $p_j = 1$ for every job j (yes, this is an easier setting with all jobs of unit size) and m identical parallel machines. In addition, now you are given *precedence constraints* among jobs - $j > j'$ means that j' can be processed only after processing j. Note that precedence constraints are transitive. We can think of the precedence constraints as inducing a collection of directed acyclic graphs on the jobs. The task is to find a schedule of the jobs to minimize makespan. Note here that now it is not enough to just assign a job to a machine (like we did for basic makespan in class), rather you need to specify exact time-points where to process the jobs (you are not allowed to break jobs).

Extend the basic greedy algorithm to give a 2-approximation for this problem.

2. First, we perform some preprocessing as follows:-

1. Create a DAG based on the precedence constraints
   - each node represents a set of jobs (will be used later) and initially has only 1 job
   - each node has a weight = sum of p_j's in the node
   - each node has a min. starting time attribute (MST)
     (initially 0 for all nodes with no incoming edges and ∞ for the rest)
2. Compress the DAG
   - if there are 2 nodes which have only 1 incoming edge and 1 outgoing edge, AND have an edge between them
     THEN combine them

     New weight = sum of the weights of two nodes
     New MST = min of MST of the 2 nodes
   - repeat till no nodes can be combined

eg.

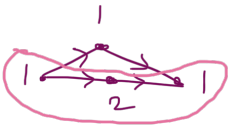
Greedy Algorithm:

1. Let the min MST = t $\forall$ nodes in DAG
2. Sort all nodes with MST = t in decreasing order of wt
3. Schedule them based on List Scheduling
   - while scheduling make sure the start time $\geq t$
   - once a node is scheduled, delete it and update the next node in the graph with
     $$MST = wt_{node} + start\ time_{node}$$
4. Repeat till all nodes are scheduled (i.e. DAG is empty)

* Due to the MST update step, we ensure that solution given by greedy is feasible.

To prove: $T \leq 2\,OPT$

Proof: Let d^* be the wt of the max wt path in the DAG.

e.g.: 

$d^* = 4$

then $OPT \geq d^*$ — (1) lower bound for OPT

and $OPT \geq n/m$ — (2) lower bound for OPT

Let's assume $T > 2OPT$ and consider the first node j^* due to which the makespan exceeded $2OPT$.

[collection of jobs in DAG]

Since each job has processing time $p_j = 1$

$\Rightarrow \geq 2OPT$ jobs are scheduled before j^*

Consider 2 cases :-

(i) Based on the DAG, the MST of $j^* = 2OPT$

i.e. its chain in the DAG has $2OPT$ jobs before it (this is irrespective of m)

which means that $d^* >$ MST of j^*

$\Rightarrow d^* > 2OPT$

Not possible, violates (1)

(ii) Based on DAG, MST of $j^* < 2OPT$

which means that $\exists$ a node i^* (MST $i^* \leq$ MST j^*) s.t. it was scheduled later than its MST in the DAG.

→ because all m machines were not available

→ At this stage List Scheduling is used. And we know that LS is 2 approx.

So it is not possible for $T > 2OPT$

Problem 3 (10 points) In the K-CENTER problem, we are given a set of n points P. We are also given a distance function $d(u, v) \geq 0$ between every pair of points $u, v \in P$. Further, the distance function is a *metric*, meaning they satisfy the following properties :

- (reflexive) $d(u, u) = 0$
- (symmetric) $d(u, v) = d(v, u)$
- (triangle inequality) $d(u, v) \leq d(u, x) + d(x, v)$ for all $x \in P$

The third property is very crucial to the problem (and so is the second !). Now, the goal is to determine a set of k centers $C \subseteq P$. Each point in P will now assign themselves to the closest selected center to form a *clustering*. The goal is to select C in such a way that minimizes the maximum distance from any point to its cluster center.

a) Suppose we know the value of OPT $= \Delta$. Here is an algorithm to solve the problem. Pick any arbitrary point, say v as the first center. Assign every point within distance of 2Δ from v to the first center v (including v itself) and delete them. Continue the same procedure on the remaining instance until all points are either a center or assigned to some center.

Prove the given Δ is the correct value of OPT, the above algorithm does not need more than k iterations.

b) How does the above imply a polynomial time algorithm to solve the actual problem (where you do not know Δ) ?

a) First, we will try to prove why the algo works.

Let K = be the set of centers chosen by OPT

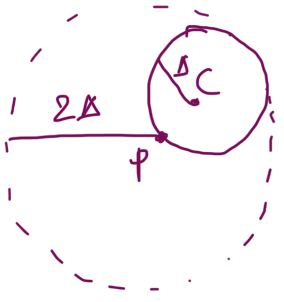
For any arbitrary point p, the distance for some $c \in K$ is $d(c, p) \leq \Delta$

So if we choose p as a center in the algorithm, c and all the other points covered by c in OPT will be covered by p.

∵ if for some pt v s.t. $d(c, v) \leq \Delta$ then

$d(v, p) \leq d(v, c) + d(c, p)$

$\leq 2\Delta$



thus each arbitrary selection will atleast cover one of the centres c (which was included in OPT).

there can be k such centres, so the algorithm takes atmost k iterations (here it's crucial that covered points be deleted)

* Also its not possible that the centre of an arbitrary point (in OPT) is not covered when the arbitrary point is chosen.

b) K-CENTER(k, P):

$LB = \min d(u,v) \quad \forall u,v \in P$

$UB = \max d(u,v) \quad \forall u,v \in P$

while $LB < UB$:

set $\Delta \leftarrow \frac{LB+UB}{2}$

isFeasible = KCENTER_Dec(P, k, Δ) → decides based on if algo in (a) ends after k iterations

if isFeasible = True:

$UB = \Delta$

else

$LB = \Delta$

Problem 4 (10 points) Suppose we are given a *directed acyclic graph* with edge costs c_e. Further, there is a length ℓ_e associated with every edge - assume all c_e, ℓ_e are non-negative. Given two vertices s and t, we need to find the $s-t$ path of minimum *cost* under the constraint that the *length* of the path does not exceed a given budget L. Design a polynomial time approximation scheme for this problem.

(Will design a PTAS using the same concepts as in KP)

General DP:

$OPT(v, l)$ = min cost path from s to v of length atmost l

$$= \min \begin{cases} OPT(v, l-1) \\ \min\limits_{u \in V} \{ OPT(u, l-l_{uv}) + c_{uv} \text{ ; if } l-l_{uv} \geq 0 \end{cases}$$

Answer = $OPT(t, L)$

this gives us a time complexity of $O(n^2 L)$

which is pseudopolynomial due to dependence on L which can be arbitrarily large.

Let's define a different DP based on cost

$LC(v, c)$ = min length of a path from s to v with cost atmost c

$$= \min \begin{cases} LC(v, c-1) \\ \min\limits_{u \in V} \{ LC(u, c-c_{uv}) + l_{uv}, \quad c-c_{uv} \geq 0 \end{cases}$$

$$= \infty \quad \text{, if } c = 0 \text{ and } v \neq s \quad \text{—— (1)}$$

$$= 0 \quad , \quad v = s$$

Time complexity: $O(n^2 C) \Rightarrow O(n^3 c_{max})$ $\qquad C = \sum\limits_{e \in E} c_e$

Space complexity : $O(nC)$

Answer: c s.t. $LC(t,c) = \min_{1 \le c' \le c} LC(t,c')$

[if $LC(t,c) > L$, then there is no feasible solution]

Scaling the instance

Scale each c_e s.t. $\hat{c}_e \in [1, n/\epsilon]$

Scaling factor, $k = \frac{c_{max}\,\epsilon}{n}$ $(n = |V|)$

PTAS algorithm:

1. Choose $\epsilon > 0$
2. Scale $\hat{c}_e = \left\lceil \frac{c_e}{k} \right\rceil$
3. Solve scaled instance using (1)

Approximation

$$cost(LC) \le k \boxed{\hat{cost}(LC)} \quad \text{→ scaled instance}$$

$$\le k\ \hat{cost}(opt)$$

$$= k \sum_{c_e \in O} \hat{c}_e$$

$$= k \sum_{c_e \in O} \left\lceil \frac{c_e}{k} \right\rceil$$

$$\le \sum_{c_e \in O} c_e + (n)k$$

(n) → shortest path will have atmost n (actually n-1) edges

$$\le OPT + \epsilon C_{max}$$

$$\le OPT(1+\epsilon)$$

Runtime w.r.t. scaled instance $O(n^3 \times n/\epsilon) \sim O(n^4/\epsilon)$