Amaiya Singhal 2021CS50598

# COL754 Assignment 1

**1.1 Partial Cover Problem:** Given a collection of $n$ elements $E$ and $m$ subsets $S_j$ of $E$, each with an associated cost $w_j$, find the collection of subsets indexed by $I$ that minimizes $\sum_{j\in I} w_j$ such that

$$\left|\bigcup_{j\in I} S_j\right| \geq p|E|$$

where $0 < p < 1$ is some constant.

**1.1 a)**

**Algorithm 1** Greedy Algorithm for Partial Cover

$I \leftarrow \phi$
$\hat{S}_j \leftarrow S_j \ \ \forall j$
**while** $\left|\bigcup_{j\in I} \hat{S}_j\right| > |E| - p|E|$ **do**
  $l \leftarrow arg\ min_{j:\hat{S}_j \neq \phi} \frac{w_j}{|\hat{S}_j|}$
  $I \leftarrow I \cup \{l\}$
  $\hat{S}_j \leftarrow \hat{S}_j - S_l \ \ \forall j$
**end while**

**Claim 1:** Algorithm 1 is a polynomial time algorithm.

*Proof.* At the beginning of each iteration of the algorithm, $\hat{S}_j = \{e \in S_j \mid e \notin \bigcup_{i\in I} S_i\}$ represents the elements of $S_j$ that have not been covered yet. Since in each iteration we choose a $l$ such that $\hat{S}_l$ is non-empty and add it to $I$, we increase the size of $\left|\bigcup_{j\in I} \hat{S}_j\right|$ by atleast 1. Hence, the total number of times the loop runs is at max the number of elements which is $n$.

Each iteration of the loop involves finding $arg\ min_{j:\hat{S}_j \neq \phi} \frac{w_j}{|\hat{S}_j|}$ which takes as much time as the number of sets which is m.

Hence, the running time of the algorithm is $\mathcal{O}(n.m)$ □

**Claim 2:** Algorithm 1 finds a solution to the partial cover problem in which the value is no more than $\left(\ln \frac{1}{1-p} + 2\right)$.OPT, where OPT is the value of the optimal solution to the set cover problem.

*Proof.* Let $n_k$ denote the number of elements that remain uncovered at the start of the $k$th iteration. Suppose the algorithm takes $l$ iterations then $n_1 = n$. Pick an arbitrary iteration k. Let $I_k$ denote the indices of the sets chosen in iterations 1 through $k-1$, and for each $j = 1, \ldots, m$, let $\hat{S}_j$ denote the set of uncovered elements in $S_j$ at the start of this iteration; that is $\hat{S}_j = S_j = \bigcup_{p\in I_k} S_p$.

**Claim 2.1:** For the set $j$ chosen in the $k$th iteration $w_j \leq \frac{n_k - n_{k+1}}{n_k}.\text{OPT}$

*Proof.* Using the fact

$$\min_{i=1\ldots k}\frac{a_i}{b_i} \leq \frac{\sum_{i=1}^{k} a_i}{\sum_{i=1}^{k} b_i} \leq \max_{i=1\ldots k}\frac{a_i}{b_i} \qquad \text{[Ch1 Fact1.10, Williamson and Shmoys]}$$

Let O contain the indices of the sets in an optimal solution of the set-cover problem, then using the above fact

$$\min_{j:\hat{S}_j \neq \phi}\frac{w_j}{|\hat{S}_j|} \leq \frac{\sum_{j\in O} w_j}{\sum_{j\in O}|\hat{S}_j|} = \frac{\text{OPT}}{\sum_{j\in O}|\hat{S}_j|} \leq \frac{OPT}{n_k}$$

where the last inequality follows from the fact that since $O$ is a set cover, the set $\bigcup_{j\in O}\hat{S}_j$ must include all remaining $n_k$ uncovered elements. Let j index a subset that minimizes this ratio, so that $\frac{w_j}{|\hat{S}_j|} \leq \frac{OPT}{n_k}$. if we add the subset $S_j$ to our solution, then there will be $|\hat{S}_j|$ fewer uncovered elements, so that $n_{k+1} = n_k - |\hat{S}_j|$. Thus,

$$w_j \leq \frac{|\hat{S}_j|.\text{OPT}}{n_k} = \frac{n_k - n_{k+1}}{n_k}.\text{OPT}$$

□

Let I contain the indices of the sets in our final solution. Then

$$\begin{aligned}
\sum_{j\in I} w_j &\leq \sum_{k=1}^{l}\frac{n_k - n_{k+1}}{n_k}.\text{OPT} && \text{[from Claim 2.1]}\\
&\leq \text{OPT}.\left(\sum_{k=1}^{l-1}\left(\frac{1}{n_k} + \frac{1}{n_k - 1} + \cdots + \frac{1}{n_{k+1}+1}\right) + \left(\frac{n_l - n_{l+1}}{n_l}\right)\right) && \left[\frac{1}{n_k} \leq \frac{1}{n_{k-i}} \forall 0 \leq i \leq n_k\right]\\
&\leq \text{OPT}.\left(\left(\frac{1}{n_1} + \frac{1}{n_1 - 1} + \cdots + \frac{1}{n_l+1}\right) + \left(\frac{n_l - n_{l+1}}{n_l}\right)\right)\\
&\leq \text{OPT}.\left(\left(\frac{1}{n} + \frac{1}{n-1} + \cdots + \frac{1}{n_l+1}\right) + \left(\frac{n_l - n_{l+1}}{n_l}\right)\right) && [n_1 = n]\\
&= \text{OPT}.\left(\left(H_n - H_{n_l}\right) + \left(\frac{n_l - n_{l+1}}{n_l}\right)\right)\\
&\leq \text{OPT}.\left(\left(H_n - H_{n_l}\right) + \left(\frac{n_l}{n_l}\right)\right) = \text{OPT}.\left(\left(H_n - H_{n_l}\right) + 1\right)
\end{aligned}$$

Since we have assumed that the algorithm takes $l$ iterations to terminate, from the condition of termination of the algorithm we can conclude that $n_1, \ldots n_l > n - np$ and $n_{l+1} \leq n - np$ and since

$H_i$ increases with $i$, $H_{n_l} > H_{n-np}$

$$\begin{aligned}
\sum_{j \in I} w_j &\leq \text{OPT.}\left(\left(H_n - H_{n-np}\right) + 1\right) && [H_{n_l} \geq H_{n-np}] \\
&\leq \text{OPT.}\left(\ln(n) - \ln(n-np) + 1\right) && [H_n \approx \ln(n)] \\
&\leq \text{OPT.}\left(\ln\left(\frac{1}{1-p}\right) + 1\right)
\end{aligned}$$

Hence, proved. We have shown a polynomial-time algorithm to find a solution to the partial cover problem in which the value is no more than $c(p)$.OPT with $c(p) = \ln \frac{1}{1-p} + 1$, a constant that depends on p as required in the problem. □

### 1.1 b)

**Claim:** Algorithm 1 is an $H_{\lceil np \rceil}$ approximation for the partial cover problem.

*Proof.* Let O be an optimal solution to the partial cover problem with value OPT. Proceeding similar to Claim 2.1 from the previous part, we have

$$\min_{j:\hat{S_j} \neq \phi} \frac{w_j}{|\hat{S_j}|} \leq \frac{\sum_{j \in O} w_j}{\sum_{j \in O} |\hat{S_j}|} = \frac{\text{OPT}}{\sum_{j \in O} |\hat{S_j}|}$$

Since O is an optimal solution, $\left|\bigcup_{j \in O} |\hat{S_j}|\right|$ must be atleast $n_k - (n - \lceil np \rceil)$ since O covers more than $\lceil np \rceil$ (ceiling since number of elements is integral) elements. So, we get

$$\sum_{j \in O} |\hat{S_j}| \geq \left|\bigcup_{j \in O} |\hat{S_j}|\right| \geq n_k - (n - \lceil np \rceil)$$

If $S_j$ is the subset chosen at the $j$th iteration,

$$w_j \leq |\hat{S_j}| \frac{\text{OPT}}{\sum_{j \in O} |\hat{S_j}|} \leq |\hat{S_j}| \left(\frac{\text{OPT}}{n_k - (n - \lceil np \rceil)}\right) = \text{OPT}\left(\frac{n_k - n_{k+1}}{n_k - (n - \lceil np \rceil)}\right)$$

Defining $n'_i = n_i - n - \lceil np \rceil \quad \forall i \in [l+1]$, we can rewrite the above equation in a form similar to the previous part as

$$w_j \leq \text{OPT}\left(\frac{n'_k - n'_{k+1}}{n'_k}\right)$$

Let I contain the indices of sets in our final solution. Then proceeding similar to the previous part,

$$\begin{aligned}
\sum_{j\in I} w_j &\leq \sum_{k=1}^{l} \frac{n'_k - n'_{k+1}}{n'_k}.\text{OPT} && \text{[from above]} \\
&\leq \text{OPT}.\left(\sum_{k=1}^{l-1}\left(\frac{1}{n'_k} + \frac{1}{n'_k - 1} + \cdots + \frac{1}{n'_{k+1}+1}\right) + \left(\frac{n'_l - n'_{l+1}}{n'_l}\right)\right) \\
&\leq \text{OPT}.\left(\left(\frac{1}{n'_1} + \frac{1}{n'_1 - 1} + \cdots + \frac{1}{n'_l+1}\right) + \left(\frac{n'_l - n'_{l+1}}{n'_l}\right)\right) \\
&\leq \text{OPT}.\left(\left(\frac{1}{\lceil np\rceil} + \frac{1}{\lceil np\rceil - 1} + \cdots + \frac{1}{n'_l+1}\right) + \left(\frac{n'_l - n'_{l+1}}{n'_l}\right)\right) && [n'_1 = \lceil np\rceil \because n_1 = n]
\end{aligned}$$

Since we have already argued in the previous part that $n_1, \ldots n_l > n - \lceil np\rceil$, we get $n'_1, \ldots n'_l > 0$. This means that $n'_l \geq 1$ since each $n_i$' is an integer (denotes number of elements which can only be integers). Hence $n'_l + 1 \geq 2$ and $\frac{1}{n'_l+1} \leq \frac{1}{2}$. We use this result in the previous inequality to get

$$\begin{aligned}
\sum_{j\in I} w_j &\leq \text{OPT}.\left(\left(\frac{1}{\lceil np\rceil} + \frac{1}{\lceil np\rceil - 1} + \cdots + \frac{1}{2}\right) + \left(\frac{n'_l - n'_{l+1}}{n'_l}\right)\right) \\
&\leq \text{OPT}.\left(\left(\frac{1}{\lceil np\rceil} + \frac{1}{\lceil np\rceil - 1} + \cdots + \frac{1}{2}\right) + \left(\frac{n'_l - n'_{l+1}}{n'_l}\right)\right) \\
&\leq \text{OPT}.\left(\left(\frac{1}{\lceil np\rceil} + \frac{1}{\lceil np\rceil - 1} + \cdots + \frac{1}{2}\right) + \left(\frac{n'_l}{n'_l}\right)\right) \\
&= \text{OPT}.\left(\left(\frac{1}{\lceil np\rceil} + \frac{1}{\lceil np\rceil - 1} + \cdots + \frac{1}{2}\right) + 1\right) \\
&= \text{OPT}.\left(\sum_{i=1}^{\lceil np\rceil} \frac{1}{i}\right) \\
&= \text{OPT}.\left(H_{\lceil np\rceil}\right)
\end{aligned}$$

Hence, we have shown an $H_{\lceil np\rceil}$-approximation algorithm for the partial cover problem. Also $f(p) = H_{\lceil np\rceil}$ and $f(1) = H_{\lceil n\rceil} = H_n$ which satisfies the constraint in the question that $f(1) \leq H_n$. □

**1.4 Uncapacitated Facility Location Problem:** We have a set of clients D and a set of facilities F. For each client $j \in D$ and facility $i \in F$, there is a cost $c_{ij}$ of assigning client $j$ to facility $i$. Furthermore, there is a cost $f_i$ associated with each facility $i \in F$. The goal of the problem is to choose a subset of facilities $F' \subseteq F$ so as to minimize $\sum_{i \in F'} f_i + \sum_{j \in D} \min_{i \in F'} c_{ij}$.

**1.4 a)**

**Claim 1.4**: There exists some $c$ such that there is no $c \ln |D|$-approximation algorithm for the uncapacitated facility location problem unless P = NP.

*Proof.* We show this by construction an instance of the set cover problem into an instance of the uncapacitated facility location problem. Suppose $I = (S, E, w)$ is an instance of the set cover problem. We construct an instance of the facility location problem such that $D = E$ (the set of clients is the set of elements in the set cover problem) and $F$ represents the set of facilities such that $i \in F$ represents $S_i$ $\forall i \in [m]$ (there is a facility corresponding to each set in the set cover problem). We define $f_i = w_i$ and define $c_{ij}$ as follows

$$c_{ij} = \begin{cases} 0 & \text{if } e_i \in S_j \\ \text{INF} & \text{if } e_i \notin S_j \end{cases}$$

where $e_i$ is the element corresponding to the $i$th client and $S_j$ is the set corresponding to the $j$th facility.

**Claim 1.4.1:** There exists a solution of cost $C(<$ INF) to an instance of the set cover problem iff there exists a solution of cost $C$ in the corresponding instance of the uncapacitated facility location problem formed by the above method.

*Proof.* ( $\implies$ ) Let $O$ (set of indices of selected subsets) be a solution of cost $C = \sum_{i \in O} w_i$ to the vertex cover problem. The corresponding solution to the uncapacitated facility location problem is $O' = \{i : i \in O\} = O$ (set of facilities chosen). The cost $C'$ to this solution is

$$\begin{aligned} C' &= \sum_{i \in O'} f_i + \sum_{j \in D} \min_{i \in F'} c_{ij} \\ &= \sum_{i \in O} f_i + \sum_{j \in D} \min_{i \in F'} c_{ij} \\ &= \sum_{i \in O} w_i + \sum_{j \in D} \min_{i \in F'} c_{ij} && [O' = O,\ f_i = w_i] \\ &= C + \sum_{j \in D} \min_{i \in F'} c_{ij} \\ &= C + 0 = C \end{aligned}$$

The last statement follows from our construction of $c_{ij}$. Since $O$ is a set cover, there is atleast 1 set in the solution which contains the $j$th element and hence the facility corresponding to that set, say $i$ provides $c_{ij} = 0$. This is true for every element since $O$ is a set cover. Hence $\min_{i \in F'} c_{ij} = 0 \ \ \forall j \in D$.

($\impliedby$) Let $O'$ be a solution (set of facilities) to the uncapacitated facility location problem with cost $C'$ ($<$ INF) where $C' = \sum_{i \in O'} f_i + \sum_{j \in D} \min_{i \in F'} c_{ij}$. Since $C' <$ INF and by our definition, $c_{ij}$ can only take values 0 or INF, it must be the case that $\min_{i \in F'} c_{ij} = 0 \ \ \forall j \in D$. Hence $C' = \sum_{i \in O'} f_i$.

The corresponding collection of sets O in the set cover problem are $O = \{i : i \in O'\} = O'$ with cost $= C$ equal to

$$\begin{aligned} C &= \sum_{i \in O} w_i \\ &= \sum_{i \in O'} w_i = \sum_{i \in O'} f_i \\ &= C' \end{aligned}$$

The collection of sets also forms a set cover. This can be observed from the fact that the collection $F'$ of facilities chosen ensured that $\min_{i \in F'} c_{ij} = 0$, this is only possible when each element $j$ is present in atleast one of the subsets $S_i$ (corresponding to the chosen facilities). This is from the definition of $c_{ij} = 0$ only when this condition holds. Hence for every element in the set cover instance, atleast one of the chosen sets must contain it, hence forming a set cover. □

**Theorem 1.4.1:** The optimal solution of the set cover instance is the same as the optimal solution of the uncapacitated facility location instance. [Directly entails from Claim 1.4.1]

**Theorem 1.4.2:** There exists some constant $c > 0$ such that if there exists a $c \ln(n)$-approximation algorithm for the unweighted set cover problem, then P = NP. [Theorem 1.14 Williamson and Shmoys]

Let $c'$ be the constant from Theorem 1.4.2. Suppose we have a $c'$-approximation algorithm for the uncapacitated facility location problem. Since, we have shown that we can model an instance of a vertex cover into an instance of uncapacitated facility location with the same optimal value [Theorem 1.4.1], we can use this algorithm to find the optimal solution to the set cover problem. But, from the above theorem we know that if this is true then P = NP. Hence, if P $\neq$ NP then such an algorithm cannot exist. $\therefore$ There exists some $c$ $(= c')$ such that there is no $c \ln(n)$-approximation algorithm for the uncapacitated facility location problem unless P = NP. □

### 1.4 b)

We will formulate the problem as an instance of the set cover problem and use the greedy algorithm for set cover to obtain an $\mathcal{O}(\ln |D|)$-approximation algorithm for the uncapacitated facility location problem.

Given an instance $(D, F, c, f)$ of the uncapacitated facility location problem, we model it as an instance $(S, E, w)$ of the set cover problem where

- $E = D$
- $S_{i,V} = \{i, V\}, i \in F, V \subseteq D$ covers elements of set $V$ and has cost $w_{i,V} = f_i + \sum_{j \in V} c_{ij}$

The greedy algorithm for set cover is an $\ln |D|$-approximation algorithm (used as a result here, can be proved similar to 1.4). Suppose $G$ is a solution found by the greedy algorithm and OPT be the cost of the optimal solution to the set cover.

**Algorithm 2** Greedy Algorithm for Set Cover

$I \leftarrow \phi$
$\hat{S}_j \leftarrow S_j \ \ \forall j$
**while** I is not a set cover **do**
  $l \leftarrow arg\ min_{j:\hat{S}_j \neq \phi} \frac{w_j}{|\hat{S}_j|}$
  $I \leftarrow I \cup \{l\}$
  $\hat{S}_j \leftarrow \hat{S}_j - S_l \ \ \forall j$
**end while**

Let $I = \{i : \exists V \subseteq E \text{ s.t. } (i,V) \in G\}$, $J_i = \{j \in V : (i,V) \in G\}$. Then the cost of our greedy solution $G$ is

$$\begin{aligned} C &= \sum_{i \in I} f_i + \sum_{i \in I} \sum_{j \in J_i} c_{ij} \leq ln|E|.\text{OPT} && [ln|E| - approximation] \\ &= \sum_{i \in I} f_i + \sum_{j \in D} \sum_{i \in I:(i,V) \in G, j \in V} c_{ij} \leq ln|E|.\text{OPT} \end{aligned}$$

Since the solution is a set cover, there is a $c_ij$ term $\forall \ \ j \in D$ in the cost above. The corresponding cost of the facility location with set of selected facilities as G is equal to

$$\begin{aligned} C' &= \sum_{i \in I} f_i + \sum_{j \in D} \min_{i \in I} c_{ij} \\ &\leq \sum_{i \in I} f_i + \sum_{j \in D} \sum_{i \in I:(i,V) \in G, j \in V} c_{ij} && [\text{there is a term } \forall j \ \in D] \\ &\leq C && \left[ \sum_{i \in I:(i,V) \in G, j \in V} c_{ij} \geq \min_{i \in I} c_{ij} \right] \\ &\leq \ln |E|.\text{OPT} = \ln |D|.\text{OPT} \end{aligned}$$

Let OPT' be the cost of the optimal solution to the uncapacitated facility location problem then to show that our greedy algorithm is a $\ln|D|$-approximation algorithm, we need to show $C' \leq \ln|D|.\text{OPT}$'. We claim this by showing that OPT = OPT'.

**Claim 1.4.2:** If there is a solution of cost $C$ to an instance of uncapacitated facility location problem corresponding to an instance of the set cover problem then there is a set cover of the same cost in that instance of the set cover problem.

*Proof.* Let $F'$ denote the set of facilities chosen in a solution to the uncapacitated facility location problem. The cost of this solution is $C = \sum_{i \in F'} + \sum_{j \in D} \min_{i \in F'} c_{ij}$. The corresponding solution B in the set cover instance with the same cost is

- $S_{i,V} \in B$ if $i \in F'$ and $V = \{j \in D : c_{ij} = \min_{i \in F'} c_{ij}\}$. In case for multiple $i$ with $c_{ij} = \min_{i \in F'} c_{ij}$, we take $j$ in only one such $V$ to ensure that each element is in exactly 1 $S_{i,V}$ set.
- Let $I = \{i : \exists V \subseteq E \text{ s.t. } (i,V) \in F'\}$, $J_i = \{j \in V : (i,V) \in F'\}$. The cost of this solution is

$$
\begin{aligned}
C' &= \sum_{i \in I} f_i + \sum_{i \in I} \sum_{j \in J_i} c_{ij} \\
&= \sum_{i \in I} f_i + \sum_{j \in D} \min_{i \in I} c_{ij} \qquad \text{[Each element in exactly 1 } S_{i,V} \text{ and } c_{ij} = \min_{i \in F'} c_{ij}] \\
&= C
\end{aligned}
$$

Hence, we have shown a solution of the same cost $C$ in the set cover instance. □

**Claim 1.4.3:** If OPT is the cost of the optimal solution to an instance of the set cover problem then the cost of the corresponding solution of the uncapacitated facility location problem is also OPT.

*Proof.* Let $O$ denote an optimal solution to an instance of the set cover problem. We argue that $O$ must be such that if $S_{m,V} \in O$ and $S_{n,V'} \in O$ then $V \cap V' = \phi$. If this was not the case then we could construct a better solution by removing common elements from the sets $V, V'$ such that each element remains in exactly 1 of the 2.

Let $I$ denote $\{i : \exists V \subseteq D, (i, V) \in O\}$ We also argue that $\forall\ j \in D$, if $j \in V : \exists i \in D, S_{i,V} \in O$ then $c_{ij} = \min_{i \in I} c_{ij}$. Suppose $i'$ is the facility such that $c_{i'j} = \min_{i \in I} c_{ij}$ and $i' \neq i$ then we can construct a better solution than $O$ as $O' = O - S_{i,V} - S_{i',V'} + S_{i,V-j} + S_{i',V'+j}$. This has a better cost than O because $c_{ij} \geq \min_{i \in I} c_{ij} = c_{i'j}$.

Using the above 2 arguments, we can easily show that there is a facility location represented by $I$ (defined above) which has cost = OPT. The cost of the facility location represented by I is

$$
\begin{aligned}
C' &= \sum_{i \in I} f_i + \sum_{j \in D} \min_{i \in I} c_{ij} \\
&= \sum_{i \in I} f_i + \sum_{i \in I} \sum_{j \in V} c_{ij} \\
&= \sum_{i \in I} \Big( f_i + \sum_{j \in V} c_{ij} \Big) \\
&= \sum_{(i,V) \in O} w_{i,V} = OPT
\end{aligned}
$$

□

**Claim 1.4.4:** An instance of the set cover problem and the corresponding instance of the uncapacitated facility location problem have the same cost optimal solution.

*Proof.* Suppose the optimal solution of the instance of the uncapacitated facility location problem corresponding to a set cover instance is OPT then the set cover instance also has a solution of cost OPT [Claim 1.4.2]. The set cover instance cannot have a better cost than OPT because if it did then [Claim 1.4.3] the uncapacitated facility location instance also has a solution with that cost which violates the fact that OPT was the optimal solution to the facility location instance. □

$C' \leq \ln |D|.OPT = \ln |D|.OPT'$ $[Claim 1.4.4]$

Hence, proved that this is an $\ln |D|$-approximation algorithm for the uncapacitated facility location problem.

**2.1** $k - suppliers$ **problem:** Input is a positive integer $k$, and a set of vertices $V$, along with distances $d_{ij}$ between any 2 vertices $i, j$. The vertices are partitioned into *suppliers* $F \subseteq V$ and *customers* $D = V - F$. The foal is to find $S \subseteq F, |S| \leq k$ that minimizes $\max_{j \in D} d(j, S)$.

**2.1 a)**

**Algorithm 3** Greedy algorithm to find set of suppliers such that $\max_{j \in D} d(j, S) \leq r$

```
S ← φ
while ∃v ∈ V : d(v, S) > r do
    l ← j ∈ D : d(j, S) > r
    if min_{i∈F−S} d_il ≤ r then
        S ← S ∪ {i'} (where i' is such that d_i'l = min_{i∈F−S} d_il)
    else(return False)
    end if
end while
```

**Greedy Algorithm for $k$-suppliers**: Binary search on the value of $r$ using algorithm 3. If the algorithm returns False (infeasible) or return $S$ such that $|S| > k$, we increase $r$, otherwise we have a feasible solution and we decrease $r$. At the end of the algorithm we have the least $r$ such that there is a feasible solution with $\leq k$ suppliers.

**Claim 2.1.1:** The greedy algorithm runs in polynomial time.

*Proof.* Binary search takes polynomial time in the maximum distance between a supplier and a customer. For each value of the radius $r$ in the binary search instance the time taken is $\mathcal{O}(|F|^2.|D|)$. We first find out a $j \in S$ for which $d(j, S) > r$ which takes $\mathcal{O}(|F||D|)$ time (iterate over all in $D$ and compute distance from each in $S$ whose size is atmost $|F|$). After finding $j$, finding a facility within $r$ distance takes $\mathcal{O}(|F|)$ time by iterating over all facilities. Hence, the overall running time of the algorithm is polynomial in the size of the input. □

**Claim 2.1.2:** The greedy algorithm is a 3-approximation algorithm for the $k$-suppliers problem.

*Proof.* Assume $r^*$ is the optimal radius of an instance of the $k$-suppliers problem. We prove that the algorithm is a 3-approximation by showing that $r = 3r^*$ is a feasible solution to the instance and hence the solution returned by the binary search would atmost be $3r^*$.

We show that for $r = 3r^*$, our algorithm returns a feasible solution such that $|S| \leq k$. Suppose the optimal solution is $O$ (set of facilities). Let $j \in D$ be a customer that gets covered by

$i \in S$ at the $i$th iteration of our greedy algorithm.

| | |
|---|---|
| $d_{j,i} \leq r^*$ | [since $r^*$ is optimal, there is some facility at distance $\leq r$] |
| $d_{j,i'} \leq r^*$for some $i' \in O$ | [since $r^*$ is optimal] |
| $d_{i,i'} \leq 2r^*$ | [Triangle Inequality] |
| $d_{i,j} \leq 3r^*$ $\forall j : d_{i',j} \leq r$ | [Triangle Inequality] |

The above analysis shows that for each $i' \in O$, there exists a corresponding $i \in S$ such that it covers every point covered by $i'$ within a distance of $3r^*$ (since each point is within $r^*$ of some facility, the above analysis holds). And because $O$ is an optimal solution, it covers every point in $D$ within a distance $r^*$ and $|O| \leq k$. This shows that $S$ covers every point within a distance of $3r^*$. Also $|S| \leq k$ because we add a facility to $S$ corresponding to an uncovered vertex and this facility covers (within $3r^*$), all points covered by the facility that covered this vertex in the optimal solution. Hence if $|S| > k$ then this is a contradiction since it would imply that $|O| > k$. Hence we have shown that this is a feasible solution.

Since $3r^*$ is a feasible solution and we are running binary search on $r$, the solution is returned has a cost at most this value. Hence, proved that the algorithm is a 3-approximation for $k$-suppliers. □

**2.1 b)**

**Claim 2.1:** There is no $\alpha$-approximation algorithm for $\alpha < 3$ unless P = NP.

*Proof.* To prove this, we consider the dominating set problem which is NP-complete. In the dominating set problem, we are given a graph $G = (V, E)$ and an integer $k$, and we must decide if there exists a set $S \subseteq V$ of size $k$ such that each vertex is either in $S$, or adjacent to a vertex in S. Given an instance of the dominating set problem, we can define an instance of the $k$-suppliers problem as follows

- The set of vertices $V'$ is $V$ and another copy of $V$, call it $V_{\text{copy}}$ (make a copy of each vertex in $V$).
- $F = V$ and $D = V_{\text{copy}}$.
- The distance function $d_{i,j}$ between $i \in F$ and $j \in D$ is defined as

$$d_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \cup i = j \\ 3 & \text{if } (i,j) \notin E \end{cases}$$

**Claim 2.1.3:** There is a dominating set of size $k$ if and only if the optimal radius for the corresponding $k$-supplier instance is 1.

*Proof.* ($\Longrightarrow$) Suppose there is dominating set $S$ of size $k$ in the graph $G = (V, E)$. In the corresponding $k$-suppliers instance we choose the suppliers $F' = S$. Since $S$ forms a dominating set, every vertex in $V$ either $\in S$ or is adjacent to a vertex in $S$. This means that $\forall j \in D, d(j, F') = 1$ since each $j \in D$ represents a vertex of $V$ and has an adjacent vertex which corresponds to a supplier in $F'$, hence by definition of $d_{ij}$, we have $d(j, F') = \min_{i \in F'} d_{ij} = 1$.

($\Longleftarrow$) Suppose that the optimal radius of the corresponding $k$-supplier instance to a dominating set instance is 1. Then the dominating set is the set of vertices corresponding to the suppliers chosen in the $k$-supplier instance. It is easy to show that this set is infact a dominating set because all the vertices that are not in this set are atmost a distance of 1 away from their nearest supplier, by definition of $d_{ij}$, this is only possible when the $(i, j) \in E$ which means that they are adjacent in the graoh $G$. Hence every vertex is either in the set or adjacent to a vertex in the set, thus forming a dominating set. □

Suppose we have a $(3 - \epsilon), \epsilon > 0$ approximation for the $k$-suppliers problem. Then the solution we get after running that algorithm on this instance of the $k$-suppliers has cost

$$\begin{aligned} C &\leq (3-\epsilon).\text{OPT} \\ &= (3-\epsilon) && [\text{OPT} = 1 \text{ for this instance}] \\ &< 3 \\ &= 1 && [\text{by definition } d_{ij} \text{ is either 1 or 3}] \end{aligned}$$

We choose the vertices corresponding to the $k$ suppliers chosen in this algorithm and since the cost is 1, the $k$ vertices form a dominating set [Claim 2.1.3]. This would imply that the algorithm is a polynomial time algorithm to solve the dominating set problem. However, this is only possible if P = NP since Dominating-Set is NP-Complete. Hence, there is no $\alpha$-approximation algorithm for $\alpha < 3$ unless P = NP. □

**2.11 Maximum Coverage Problem:** We are given a set of elements $E$, and $m$ subsets of elements $S_1, \ldots S_m \subseteq E$ with a nonnegative weight $w_j \geq 0$ for each subset $S_j$. We would like to find a subset $S \subseteq E$ of size $k$ that maximizes the total weight of the subsets covered by $S$, where $S$ covers $S_j$ if $S \cap S_j \neq \phi$.

**2.11a)**

Refer to Problem 5.9 c) (solved here) for a $\left(1 - \frac{1}{e}\right)$-approximation algorithm for the maximum coverage problem.

**2.11b)**

**Claim 2.11.1:** If an approximation algorithm with performance guarantee better than $1 - \frac{1}{e} + \epsilon$ exists for the maximum coverage problem for some constant $\epsilon > 0$, then every NP-complete problem has an $\mathcal{O}(n^{\mathcal{O}(\log \log(n))})$ time algorithm.

*Proof.* We use the following theorem:

**Theorem 2.11:** If there is a $c \ln(n)$-approximation algorithm for the unweighted set cover problem for some constant $c < 1$, then there is an $\mathcal{O}(n^{\mathcal{O}(\log \log(n))})$-time deterministic algorithm for each NP-complete problem. [Theorem 1.13, Williamson and Shmoys]

To prove the claim, we construct an instance of maximum coverage $(E, S_j, w_j)$ from an instance of the unweighted-set cover problem $(E', S'_j)$ as follows:

- The sets from the set cover instance become the elements $E$ in the maximum coverage instance.

- Each of the elements $e_i \in E'$ from the set cover instance correspond to a subset ($S_i$) in the maximum coverage instance such that each of these subsets contains all the elements (from $E$) representing those sets in the set cover instance which contained the element $e$.
- $w_j = 1$ corresponding to each $S_j$.

We run the algorithm for maximum coverage (with all values of $k$ varying from 1 to the number of sets $S'_j$ in the set cover instance) on this instance corresponding to a set cover instance. For each value of $k$, we run the algorithm multiple times until we obtain a coverage of $|E'|$ (i.e. we cover all elements).

Suppose we have an $x = 1 - \frac{1}{e} + \epsilon$-approximation algorithm for the maximum coverage problem. Suppose OPT is the optimal solution to the unweighted set cover problem.Let $l$ denote the number of iterations we have to run this algorithm to obtain a coverage of size $|E'|$. In each iteration of the algorithm we took OPT elements (hence OPT subsets in the set cover instance). So this would be an $l$-approximation algorithm for the set cover problem.

Assume that we started with $n$ elements. We also have $x > (1 - \frac{1}{e})$ and hence $ln(1-x) < -1$ so $ln(\frac{1}{1-x}) > 1$ and $\ln(\frac{1}{1/(1-x)}) < 1$. Using this fact,

$$
\begin{aligned}
n(1-x)^l &\geq 1 && \text{[otherwise we would have terminated]} \\
(1-x)^l &\geq \frac{1}{n} \\
l.\ln(1-x) &\geq -\ln(n) \\
l &\leq \frac{\ln(n)}{\ln\left(\frac{1}{1-x}\right)} < \ln(n) && \text{[using above analysis]}
\end{aligned}
$$

This shows that if we have an $1-\frac{1}{e}+\epsilon$-approximation algorithm for the maximum coverage problem then we have a $c\ln(n),\ c < 1$-approximation algorithm for the unweighted set cover problem. But from theorem 2.11, if such an algorithm exists then every problem NP-complete problem has an $\mathcal{O}(n^{\mathcal{O}(\log\log(n))})$ time algorithm which is what we wanted to show. □

**3.3 Job Scheduling Problem:** There are n jobs to be scheduled on a single machine, where each job $j$ has a processing time $p_j$, weight $w_j$ and a due date $d_j$, $j = 1, \ldots n$. The objective is to schedule the jobs so as to maximize the total weight of the jobs that complete by their due date.

**Claim 3.3.1:** There always exists an optimal schedule in which all the on-time jobs complete before all late jobs, and the on-time jobs complete in an earliest due date order.

*Proof.* Firstly we show that there is an optimal schedule where all on-time jobs complete before all the late jobs. Let $O$ be an optimal scheduling of the jobs. If all on-time jobs complete before all the late jobs then we are done. Suppose some on-time jobs complete after some late jobs. We argue that if we move the late jobs before any on-time job, we cannot get a worse solution than the optimal. This is because if a job $i$ starts at time $t_i$ and is on-time then $t_i + p_i \leq d_i$. If we exchange the late job $j$ which started just before job $i$ (in the queue) at time $t_j$ then the completion time of job $i$ now becomes $t_j + p_i$ and since $t_j \leq t_i$, we get $t_j + p_i \leq d_i$. Hence, by moving a late job to

after an on-time job, the on-time job still finishes before its due date and since we have exchanged consecutive jobs, the completion time of further jobs is unaffected. This proves that we the total weight of the jobs that complete by their due date cannot increase. And since $O$ was an optimal solution, this reordering has total weight equal to that of $O$ (cannot do better than optimal).

Secondly, we argue that if we have an optimal ordering of jobs where all the on-time jobs complete before all late jobs then there is an optimal ordering where all on-time jobs complete before all late jobs in earliest due date order. Let $O'$ be the optimal ordering. If the on-time jobs complete in the earliest deadline order then we are done. Otherwise, suppose $d_i \leq d_j$, but $t_i \geq t_j$ are 2 on-time jobs (consecutive jobs, hence $t_j + p_j = p_i$) in $O'$ which do not complete in the earliest deadline order. Since both jobs are on-time, we also have $t_i + p_i \leq d_i$ and $t_j + p_j \leq d_j$. If we exchange the order of jobs $i, j$, the completion time of job $i$ is now $t_j + p_i \leq t_i + p_i \leq d_i$ since $t_j \leq t_i$ which is still on-time. Similarly completion time of job $j$ is $(t_j + p_i) + p_j = (t_i + p_i) \leq d_i \leq d_j$ which is also on-time. Hence we can keep exchanging consecutive on-time jobs until all follow the earliest deadline order and all on-time jobs remain on-time. Since $O'$ was optimal and we cannot do better than it, the new ordering has the same total weight as $O'$.

The above 2 arguments show that there is always an optimal solution in which all on-time jobs complete before all late jobs in the earliest due date order. □

**Dynamic Programming Solution:** We construct a dynamic programming solution that runs in time $\mathcal{O}(nW)$ where $W = \sum_j w_j$ to solve the problem as follows:

- Order the jobs in the earliest due date first order.
- Let $dp_{i,j}$ denote the least time in which we can run a subset of the first $i$ jobs to get a total weight of on-time jobs as $j$.
- Initialise $dp_{i,j} \forall i \in [n], j \in [0, W]$ as INF (very large number), except $dp_{0,0} = 0$.

**Algorithm 4** Dynamic Programming Solution to Job Scheduling

```
dp_{i,j} ← 0 ∀i ∈ [n], j ∈ [0, W]
dp_{0,0} ← 0
for i = 1, i ≤ n do
    for j = 1, j ≤ W do
        if dp_{i-1,j} ≠ INF and j + w_i ≤ W then
            dp_{i,j+w_i} ← min(dp_{i,j+w_i}, p_i + dp_{i-1,j})
        end if
    end for
end for
```

The optimal answer is $max_{j:dp_{n,j} \neq INF}(j)$ and the optimal schedule is the jobs that were considered to get this total weight (in the order in which they were processed to ensure earliest due date order) followed by the remaining jobs that do not complete in any order. Since we only consider jobs in earliest due date order and this DP approach considers all possibilities of orderings of jobs where on-time jobs finish before all late jobs, we can conclude [from Claim 3.3.1] that this algorithm gives us an optimal solution.

**Fully Polynomial Time Approximation Scheme:** Let $OPT$ be the optimal scheduling solution for the given problem, consider rounding down of weights to multiples of $\delta$ which can be formulated as a new instance for dynamic programming with running time of $O(\frac{nW}{\delta})$, and optimal scheduling $DP$. Now let $OPT'$ be the same schedule as $OPT$ but with new rounded weights, the we claim that

$$DP \geq OPT'$$

Since there can be at maximum $n$ jobs and for each jobs the error in the new solution can be $\delta$, we get

$$OPT' \geq OPT - n\delta$$

Let $W_{max}$ be the maximum weight of any feasible job then

$$OPT \geq W_{max}$$

Now to get $1 - \epsilon$ approximation set $\delta = \frac{\epsilon W_{max}}{n}$

$$DP \geq OPT - \epsilon W_{max}$$

using $OPT \geq W_{max}$ we get

$$DP \geq (1 - \epsilon)OPT$$

Since $W \leq nW_{max}$ , we get running time as $O(\frac{n^3}{\epsilon})$ which is fully polynomial in $n$ and $\frac{1}{\epsilon}$.

**4.7 Maximum Cut:** We are given as input an undirected graph $G = (V, E)$ with non-negative weights $w_{i,j} \geq 0$ for all $(i, j) \in E$. We wish to partition the vertex set into 2 parts $U$ and $W = V - U$ so as to maximize the weight of the edges whose 2 endpoints are in different parts. Assume we are given an integer $k \leq |V|/2$ and we must find a partition such that $|U| = k$.

**4.7a)**

**Claim 4.7.1:** The following nonlinear integer program models the maximum cut problem with a constraint on the size of the parts:

$$\text{maximize} \sum_{(i,j)\in E} w_{i,j}(x_i + x_j - 2x_i x_j)$$

$$\text{subject to} \sum_{i\in V} x_i = k,$$

$$x_i \in \{0, 1\}, \forall i \in V$$

*Proof.* Firstly, we show that for every cut of size $k$ in the graph, there is a corresponding assignment of variables in the given nonlinear integer program which satisfies the constraints and has the same value of the objective. Let $P$ be a cut of size $k$. The corresponding assignment of variables is:

- The variable $x_i$ is assigned values

$$x_i = \begin{cases} 1 & \text{if } i \in P \\ 0 & \text{if } i \notin P \end{cases}$$

- This clearly satisfies the constraint $x_i \in \{0, 1\}$.
- $\sum_{i \in V} x_i = \sum_{i \in P} 1 + \sum_{i \notin P} 0 = |P| = k$, this constraint is also satisfied.
- The objective value C is

$$\begin{aligned} C &\leq \sum_{(i,j) \in E} w_{i,j}(x_i + x_j - 2x_i x_j) \\ &= \sum_{i \in P, j \in P} w_{i,j}(1 + 1 - 2.1.1) + \sum_{i \in P, j \notin P} w_{i,j}(1 + 0 - 2.1.0) + \sum_{i \notin P, j \notin P} w_{i,j}(0 + 0 - 2.0.0) \\ &= \sum_{i \in P, j \notin P} w_{i,j}(1) = \sum_{i \in P, j \notin P} w_{i,j} \end{aligned}$$

The objective value $C = \sum_{i \in P, j \notin P} w_{i,j}$, the total weight of edges whose two endpoints are in different parts which is exactly the value corresponding to the cut P.

Secondly, we also show that a given solution to the nonlinear integer program with a certain value of the objective corresponds to a cut of the graph $G$ with the same value. Consider some assignment to the variables $x_i$s, suppose $S = \{i : x_i = 1\}$ and $S' = \{i : x_i = 0\}$ where $|S| = k$ from the constraint $\sum_{i \in V} x_i = k$. Consider $U = S$ and $W = S'$. The total weight C of the edges whose two endpoints are in different parts is

$$\begin{aligned} \sum_{i \in S, j \notin S} w_{i,j} &= \sum_{i \in S, j \notin S} w_{i,j}(1 + 0 - 2.1.0) + \sum_{i \in S, j \in S} w_{i,j}(1 + 1 - 2.1.1) + \sum_{i \notin S, j \notin S} w_{i,j}(0 + 0 - 2.0.0) \\ &= \sum_{(i,j) \in E)} w_{i,j}(x_i + x_j - 2x_i x_j) \end{aligned}$$

This shows that the cut has total weight of edges with endpoints in different points equal to the objective function of the nonlinear integer program.

Using the above 2 arguments we can conclude that the given nonlinear integer program models the maximum cut problem with a constraint on the size of the parts since the optimal solution for both is the same. □

**4.7b)**

**Claim 4.7.2:** The following linear program is a relaxation of the problem:

$$\begin{aligned}
\text{maximize} \quad & \sum_{(i,j)\in E} w_{i,j}(z_{i,j}) & \\
\text{subject to} \quad & z_{ij} \leq x_i + x_j, & \forall (i,j) \in E \\
& z_{ij} \leq 2 - x_i - x_j, & \forall (i,j) \in E, \\
& \sum_{i \in V} x_i = k, & \\
& 0 \leq z_{ij} \leq 1, & \forall (i,j) \in E, \\
& 0 \leq x_i \leq 1, & \forall i \in V.
\end{aligned}$$

*Proof.* To show that this problem is a relaxation to the previous nonlinear integer program we first argue that every solution to the nonlinear program satisfies the constraints of this linear program. Since, the constraints on $x_i$ in this program are the same except that the linear program requires $0 \leq x_i \leq 1$ which is true for any assignment of $x_i$ which satisfies $x_i \in \{0, 1\}$ in the non-linear program, for any assignment of $z_{ij}$ which satisfies the constraints we have a valid assignment to all variables in this linear program.

Now, we show that the optimal solution to this linear program is atleast as much as the optimal solution to the nonlinear integer program. For this we need to show that $z_{ij} \geq (x_i + x_j - 2x_i x_j)$ in the optimal solution. We use the fact that one of the constraints on $z_{ij}$ must be tight in the optimal solution (otherwise we could increase $z_{ij}$ until atleast one constraint is tight). There are 3 cases to consider:

- **Case 1:** $z_{ij} = 1$. $x_i + x_j - 2x_i x_j \leq 1$ trivially.
- **Case 2:** $z_{ij} = x_i + x_j \geq x_i + x_j - 2x_i x_j$.
- **Case 3:** $z_{ij} = 2 - x_i - x_j$. We use the fact that $(1 - x_i)(1 - x_j) = 1 - x_i - x_j + x_i x_j \geq 0$ and hence $2 - x_i - x_j \geq x_i + x_j - 2x_i x_j$ as $0 \leq x_i \leq 1$ and $0 \leq x_j \leq 1$.

Since in each of the 3 cases $z_{ij} \geq x_i + x_j - 2x_i x_j$, we get that the optimal value of this linear program is atleast the optimal of this linear program and we had already shown that every assignment in the nonlinear integer program is a valid assignment in this linear program. Therefore this linear program is a relaxation. □

**4.7c)**

**Claim 4.7.3:** For any $(x, z)$ that is a feasible solution to the linear programming relaxation, $F(x) \geq \frac{1}{2} \sum_{(i,j)\in E} w_{ij} z_{ij}$ where $F(x) = \sum_{(i,j)\in E} w_{ij}(x_i + x_j - 2x_i x_j)$ is the objective function from the nonlinear integer program.

*Proof.* We prove this by showing that $x_i + x_j - 2x_i x_j \geq \frac{1}{2} z_{ij}$ or that $x_i + x_j - 2x_i x_j - \frac{1}{2} z_{ij} \geq 0$. Since $(x, z)$ is a feasible solution it satisfies the constraints that $z_{ij} \leq x_i + x_j$ and $z_{ij} \leq 2 - x_i - x_j$. Assume $z = \min(x_i + x_j, 2 - x_i - x_j)$

- **Case 1:** $z = x_i + x_j$.

$$\begin{aligned}
x_i + x_j - 2x_ix_j - \frac{1}{2}z_{ij} &\geq x_i + x_j - 2x_ix_j - \frac{z}{2} && [z_{ij} \leq z] \\
&= z + x_j - 2x_j(z - x_j) - \frac{z}{2} && \text{[Substitution]} \\
&= 2x_j^2 - 2x_jz + \frac{z}{2} \\
&= \frac{4x_j^2 - 4x_jz + z}{2} \\
&= \frac{(2x_j - z)^2 - z^2 + z}{2} \\
&= \frac{z - z^2}{2} && \text{[Square is positive]} \\
&\geq 0 && [z \geq z^2 \text{since} 0 \leq z \leq 1]
\end{aligned}$$

- **Case 2:** $z = 2 - x_i - x_j$.

$$\begin{aligned}
x_i + x_j - 2x_ix_j - \frac{1}{2}z_{ij} &\geq x_i + x_j - 2x_ix_j - \frac{z}{2} && [z_{ij} \leq z] \\
&= 2 - z - 2x_j(2 - x_j - z) - \frac{z}{2} && \text{[Substitution]} \\
&= 2x_j^2 - 2x_j(2 - z) + 2 - \frac{3z}{2} \\
&= \frac{4x_j^2 - 4x_j(2 - z) + 4 - 3z}{2} \\
&= \frac{(2x_j - (2 - z))^2 - (2 - z)^2 + 4 - 3z}{2} \\
&= \frac{4 - 3z - (4 + z^2 - 4z)}{2} && \text{[Square is positive]} \\
&= \frac{z - z^2}{2} \\
&\geq 0 && [z \geq z^2 \text{since} 0 \leq z \leq 1]
\end{aligned}$$

We have shown that $x_i + x_j - 2x_ix_j - \frac{1}{2}z_{ij} \geq 0$ which implies $x_i + x_j - 2x_ix_j \geq \frac{1}{2}z_{ij}$. Taking summation over all $(i, j) \in E$, we get the desired result

$$F(x) = \sum_{(i,j)\in E} w_{ij}(x_i + x_j - 2x_ix_j) \geq \sum_{(i,j)\in E} \frac{1}{2}w_{ij}z_{ij}$$

□

**4.7d)**

**Claim 4.7.4:** Given a fractional solution $x$, for 2 fractional variables $x_i$ and $x_j$, it is possible to increase one by $\epsilon > 0$ and decrease the other by $\epsilon$ such that $F(x)$ does not decrease and one of the 2 variables becomes integer.

*Proof.* Given a fraction solution, if all $x_i$ are integers then we are done. If not then there exists at least 2 fractional $x_i$ and $x_j$ because their sum is an integer. Assuming all other values as constants we get,

$$F(x) = a_0 x_i + a_1 x_j - a_2 x_i x_j + a_3$$

where $a_2 = 2w_{ij} \geq 0$, then put $x_i = x_i + \epsilon$ and $x_j = x_j - \epsilon$, then new function is

$$F_{new}(x) = F(x) + constant \cdot \epsilon + a_2 \epsilon^2$$

Since $a_2 \geq 0$, we can choose sign of $\epsilon$ (negative $\epsilon$ would just mean that we decrease $x_i$ and increase $x_j$) according to the *constant* and make either $x_i$ or $x_j$ integer without decreasing $F(x)$. □

**4.7e)**

**Algorithm for Maximum Cut:** Find a solution to the relaxed linear program. Choose any 2 fractional variables and apply the argument of part d) until all variables are integral. Choose $i$ such that $x_i = 1$ as the set $U$.

**Claim 4.7.5** The algorithm terminates and runs in polynomial time.

*Proof.* The linear program can be solved in polynomial time (number of variables is also polynomial). In part d) we have shown that we can choose 2 fractional variables $x_i$ and $x_j$ and convert one of them into an integer. We perform this until all $x_i$s are integers. If we $\geq 2$ fractional variables then we are able to do this and the number of fractional variables decreases by 1 in each iteration. The number of fractional variables cannot be 1 because of the constraint $\sum_{i \in V} x_i = k$. Hence there are always $\geq 2$ fractional variables where we are able to convert atleast 1 of them into an integer and at the end we are left with 0 fractional variables. Hence the algorithm terminates. The number of iterations of this is atmost the number of fractional variables which is polynomial in number. Therefore, the overall running time of the algorithm is polynomial in the size of the input. □

**Claim 4.7.6:** The algorithm is a $\frac{1}{2}$-approximation algorithm for the maximum cut problem with a constraint on the size of the parts.

*Proof.* Let OPT be the value of the optimal solution to an instance of the maximum cut problem. Using part a), we know that this is also the optimal solution of the nonlinear integer program defined previously and using part b) we know that the linear program is a relaxation to this nonlinear integer program. Assume $(x, z)$ is the optimal solution to the linear program. Formally, we can write this as

$$\begin{aligned} \text{OPT} &\leq \sum_{(i,j) \in E} w_{ij} z_{ij} && \text{[Part a) and b)]} \\ \frac{\text{OPT}}{2} &\leq \frac{1}{2} \sum_{(i,j) \in E} w_{ij} z_{ij} \\ &\leq F(x) && \text{[Part c)]} \end{aligned}$$

We convert this fractional solution into an integral solution using the algorithm described above. Say $x^*$ denotes the integral solution obtained. From part d) we know that this process of converting

$x_i$s into integers does not decrease the cost. We can write this as:

$$\frac{\text{OPT}}{2} \leq F(x)$$
$$\frac{\text{OPT}}{2} \leq F(x^*) \qquad \text{[Part d)]}$$

Hence, we have proved that the algorithm described is an $\frac{1}{2}$-approximation algorithm for the maximum coverage problem. □

**5.9 Maximum Coverage Problem:** We are given a set of elements $E$, and $m$ subsets of elements $S_1, \ldots S_m \subseteq E$ with a nonnegative weight $w_j \geq 0$ for each subset $S_j$. We would like to find a subset $S \subseteq E$ of size $k$ that maximizes the total weight of the subsets covered by $S$, where $S$ covers $S_j$ if $S \cap S_j \neq \phi$.

**5.9a)**

**Claim 5.9.1:** The following nonlinear integer program models the maximum coverage problem:

$$\begin{aligned} \text{maximize} \quad & \sum_{j\in[m]} w_j \left(1 - \prod_{e\in S_j}(1-x_e)\right) \\ \text{subject to} \quad & \sum_{e\in E} x_e = k, \\ & x_e \in \{0,1\}, \quad \forall e \in E \end{aligned}$$

*Proof.* We firstly show that for every subset $S \subseteq E$ of size $k$, there is a corresponding assignment of variables in the given nonlinear integer program such that the objective of the linear program has the same value as the total weight of subsets covered by S. The assignment of variables is done as follows:

- The variable $x_e$ is assigned values

$$x_e = \begin{cases} 1 & \text{if } e \in S \\ 0 & \text{if } e \notin S \end{cases}$$

- This clearly satisfies the constraint that $x_e \in \{0,1\}\ \forall e \in E$.
- $\sum_{e\in E} x_e = \sum_{e\in S} 1 + \sum_{e\notin S} 0 = |S| = k$, this constraint is also satisfied.
- The objective value $C$ is

$$\begin{aligned} C &= \sum_{j\in[m]} w_j \left(1 - \prod_{e\in S_j}(1-x_e)\right) \\ &= \sum_{j:S_j\cap S\neq\phi} w_j(1) + \sum_{j:S_j\cap S=\phi} w_j(0) \qquad [(1-x_e) = 1\ \forall e \in S_j \text{ when } S_j \cap S = \phi] \\ &= \sum_{j:S_j\cap S\neq\phi} w_j \end{aligned}$$

The objective value $C$ is equal to the sum of weights of all subsets which are covered by $S$ which is exactly the cost of the solution S of the maximum coverage problem.

Secondly, we show that given an assignment of variables to the nonlinear integer program, there is a corresponding set $S$ which has the sum of weights of subsets covered by $S$ exactly equal to the objective of the nonlinear integer program. Consider some assignment to the variables $x_e$s, suppose $X = \{i : x_e = 1\}$ and $X' = \{i : x_e = 0\}$ where $|S| = k$ from the constraint $\sum_{e \in E} x_e = k$. Consider $S = X$ as the subset for coverage. The total weights of subsets covered by $S$ is

$$\begin{aligned}
\sum_{j:S_j \cap S \neq \phi} w_j &= \sum_{j:S_j \cap S \neq \phi} w_j(1) + \sum_{j:S_j \cap S = \phi} w_j(0) \\
&= \sum_{j:S_j \cap S \neq \phi} w_j(1-0) + \sum_{j:S_j \cap S = \phi} w_j(1-1) \\
&= \sum_{j:S_j \cap S \neq \phi} w_j \left(1 - \prod_{e \in S_j} (1 - x_e)\right) + \sum_{j:S_j \cap S = \phi} w_j \left(1 - \prod_{e \in S_j} (1 - x_e)\right) \\
&= \sum_{j \in [m]} w_j \left(1 - \prod_{e \in S_j} (1 - x_e)\right)
\end{aligned}$$

This is exactly equal to the objective of the nonlinear integer program. Hence, we have shown that for every solution to the maximum coverage problem, there is an assignment of variables in the corresponding nonlinear integer program such that the objective is same as the cost of the solution to coverage instance and vice versa. Therefore the nonlinear integer program models the maximum coverage problem. □

**5.9b)**

**Claim 5.9.2:** The following linear program is a relaxation of the maximum coverage problem:

$$\begin{aligned}
\text{maximize} \quad & \sum_{j \in [m]} w_j z_j \\
\text{subject to} \quad & \sum_{e \in E} x_e \geq z_j, \quad \forall j \in [m] \\
& \sum_{e \in E} x_e = k, \\
& 0 \leq z_j \leq 1, \quad \forall j \in [m] \\
& 0 \leq x_e \leq 1, \quad \forall e \in E
\end{aligned}$$

*Proof.* To show that this problem is a relaxation to the previous nonlinear integer program we first argue that every solution to the nonlinear program satisfies the constraints of this linear program. Since, the constraints on $x_e$ in this program are the same except that the linear program requires $0 \leq x_e \leq 1$ which is true for any assignment of $x_e$ which satisfies $x_e \in \{0, 1\}$ in the non-linear program, for any assignment of $z_{z_j}$ which satisfies the constraints we have a valid assignment to all variables in this linear program.

Now, we show that the optimal solution to this linear program is atleast as much as the optimal

solution to the nonlinear integer program. For this we will show that $z_j \geq \left(1 - \prod_{e \in S_j}(1 - x_e)\right)$ in the optimal solution. We use the fact that one of the constraints on $z_{i,j}$ must be tight in the optimal solution (otherwise we could increase $z_{i,j}$ until atleast one constraint is tight). There are 2 cases to consider:

- **Case 1:** $z_j = 1 \geq \left(1 - \prod_{e \in S_j}(1 - x_e)\right)$, trivially since $\prod_{e \in S_j}(1 - x_e) \leq 1$
- **Case 2:** $z_j = \sum_{e \in E} x_e$. Since $0 \leq x_e \leq 1 \ \ \forall e \in E$, we can write

$$\begin{aligned} z_j = \sum_{e \in E} x_e &\geq \sum_{e \in S_j} x_e \\ &\geq \left(1 - \prod_{e \in S_j}(1 - x_e)\right) \qquad \text{[Holds for } 0 \leq x_e \leq 1\text{]} \end{aligned}$$

Using this, it directly follows that

$$w_j z_j \geq w_j \left(1 - \prod_{e \in S_j}(1 - x_e)\right)$$

$$\sum_{j \in [m]} w_j z_j \geq \sum_{j \in [m]} w_j \left(1 - \prod_{e \in S_j}(1 - x_e)\right)$$

Hence, we have shown that this linear program is a relaxation to the nonlinear integer program. □

**5.9c)**

**Algorithm for Maximum Coverage:** Find a solution to the relaxed linear program. Choose any 2 fractional variables and apply the approach similar to 4.7d) until all variables are integral. Choose $e$ such that $x_e = 1$ as the set $S$.

**Claim 5.9.3:** The algorithm terminates and runs in polynomial time.

*Proof.* Solving the linear program takes polynomial time (the number of variables is polynomial). Similar to Problem 4.7, since here also we have a constraint that $\sum_{e \in E} x_e = k$, we are ensured that the rounding scheme terminates as we saw earlier. Hence the algorithm terminates and the overall running time of the algorithm is polynomial in the size of the input. □

**Claim 5.9.4:** The algorithm is a $\left(1 - \frac{1}{e}\right)$-approximation algorithm for the maximum coverage problem.

*Proof.* Similar to problem 4.7, we show that for any feasible solution to the relaxed linear program, $F(x) \geq \sum_{j \in [m]} w_j z_j$ where $F(x)$ is the objective of the nonlinear integer program. We want to show that $\left(1 - \prod_{e \in S_j}(1 - x_e)\right) \geq z_j \ \ \forall j \in [m]$. Starting with the following equations:

- **Case 1:** $\sum_{e\in S_j} x_e \leq 1$ (this is a tighter constraint for $z_j$)

$$e^{-\sum_{e\in S_j} x_e} = \prod_{e\in S_j} e^{-x_e}$$
$$= \prod_{e\in S_j} (1 - x_e + x_e^2/2 \dots) \geq \prod_{e\in S_j} (1 - x_e) \qquad \text{[Taylor Series]}$$

$$1 - \prod_{e\in S_j} (1 - x_e) \geq 1 - e^{-\sum_{e\in S_j} x_e}$$
$$= \sum_{e\in S_j} x_e . \left( \frac{1 - e^{-\sum_{e\in S_j} x_e}}{\sum_{e\in S_j} x_e} \right)$$
$$\geq \sum_{e\in S_j} x_e . \left(1 - \frac{1}{e}\right) \qquad \left[\frac{1-e^{-t}}{t} \geq 1 - \frac{1}{e} \text{ when } t \in (0,1)\right]$$

Taking summation over $j \in [m]$

$$\sum_{j\in[m]} w_j \left(1 - \prod_{e\in S_j} (1 - x_e)\right) \geq \sum_{j\in[m]} w_j \sum_{e\in S_j} x_e . \left(1 - \frac{1}{e}\right)$$
$$\geq \sum_{j\in[m]} w_j z_j \left(1 - \frac{1}{e}\right) \qquad \text{[constraint } z_j \leq sum_{e\in S_j} x_e\text{]}$$
$$= \left(1 - \frac{1}{e}\right) \sum_{j\in[m]} w_j z_j$$

- **Case 2:** $\sum_{e\in S_j} x_e > 1$, hence $z_j \leq 1$ is a tighter constraint.

$$1 - \prod_{e\in S_j} (1 - x_e) \geq 1 - e^{-\sum_{e\in S_j} x_e} \qquad \text{[similar to previous case]}$$
$$\geq \left(1 - \frac{1}{e}\right)$$

Taking summation over $j \in [m]$

$$\sum_{j\in[m]} w_j \left(1 - \prod_{e\in S_j} (1 - x_e)\right) \geq \sum_{j\in[m]} w_j . \left(1 - \frac{1}{e}\right)$$
$$\geq \sum_{j\in[m]} w_j z_j \left(1 - \frac{1}{e}\right) \qquad \text{[constraint } z_j \leq 1\text{]}$$
$$= \left(1 - \frac{1}{e}\right) \sum_{j\in[m]} w_j z_j$$

Hence, we have shown that $\left(1-\frac{1}{e}\right).\sum_{j\in[m]} w_j z_j \leq F(x)$ and since we showed that the linear program is a relaxation, the optimal value to the linear program is atleast as much as the optimal value of the nonlinear integer program whose optimal value is equal to OPT. Hence we can write $\left(1-\frac{1}{e}\right).\text{OPT} \leq \left(1-\frac{1}{e}\right).\sum_{j\in[m]} w_j z_j \leq F(x)$. Using the rounding scheme (similar to 4.7), we obtain an integer solution which does not increase the cost. Let this be denoted by $x*$. Since this does not increase the cost we can write

$$\left(1-\frac{1}{e}\right).\text{OPT} \leq F(x) \leq F(x^*)$$

Hence, this algorithm is a $(1-\frac{1}{e})$-approximation algorithm for maximum coverage. □