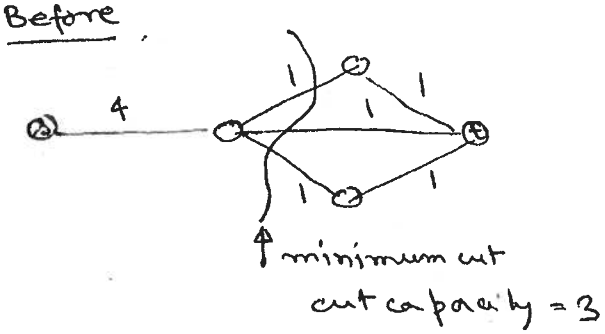
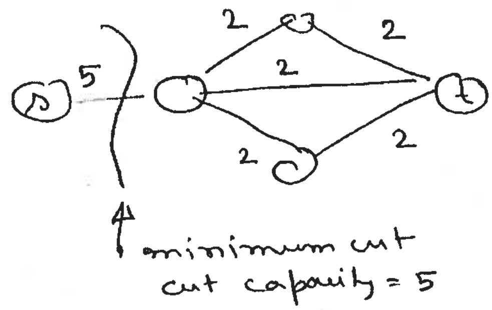
HW#6
Solution

(1) NO (A,B) may not ~~be~~ remain to be a minimum cut after capacity of each edge is increased by 1. Here is an example:

Before



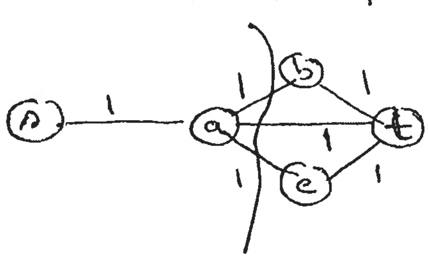
After



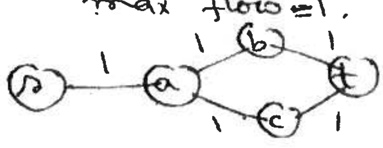
(2) Ford-Fulkerson algorithm can be used to compute max flow ~~in~~ of this network.

Let the identical capacity of each edge be c. The source node can be connected to at most $|V|-1$ other nodes. Therefore maximum possible flow is $c \times (|V|-1) = O(|V|)$. Therefore Ford-Fulkerson algorithm must run at most $O(|V|)$ time. In fact bottleneck capacity (residual capacity) of any augmenting path in the residual network is c and thus Ford-Fulkerson will run at most $|V|$ iterations. Each iteration takes $O(|E|)$ time. Thus total running time is $O(|V||E|)$.

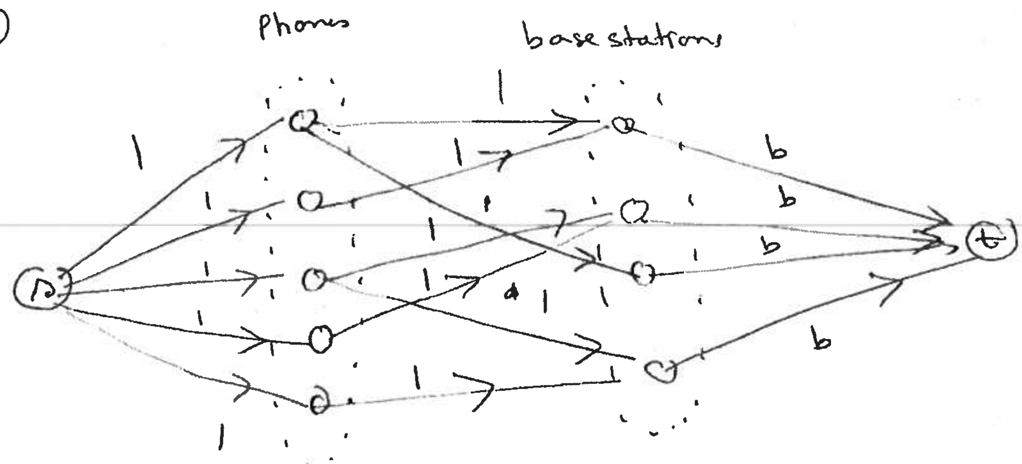
(3) NO the maximum flow of the resulting graph will not necessarily decrease. Consider the following example where ~~each~~ capacity of each edge is 1.



If the edge (a,t) is carrying flow 1 and we remove it the resulting graph will still have max flow = 1.



(4)



Construct a flow network as follows

Assign a node for each cell phone and each base station.

Assign an additional source node (s), an an additional sink node (t).

Source node s has an edge to each cell-phone with capacity 1.

Each base station has an edge to sink node with capacity b.

If a cell phone is within a distance d of any base station then assign an edge from this cell phone to the base station with capacity 1.

~~Note this~~

Due to flow conservation, this construction will ensure that ~~the~~ each cell phone will be assigned to at most one base station and each base station can handle at most b cell-phones.

Now we can use Ford-Fulkerson algorithm to solve the bipartite matching problem.

Note that max flow is at most $k \times 1 = k$.

Thus ford-fulkerson algorithm will run at most k iterations.

Now, the total # of edges in the network is at most

$$p + kp + p = O(kp)$$

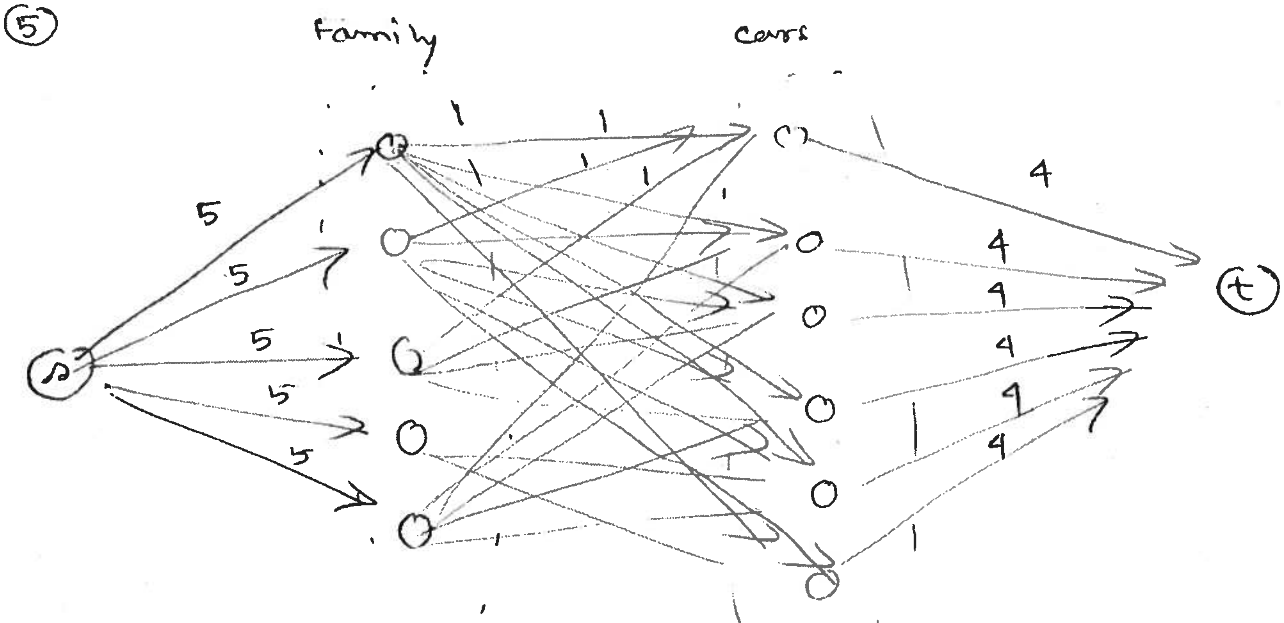
where the first term is due to edges between s and cellphones. The second term is due to all possible edges between cellphone and base stations. The third term is due to edges between base stations and t.

Therefore we have shown $|E| = O(kp)$.

$\therefore$ Running time of ford-fulkerson is $O(f^*|E|) = O(k.kp)$
$= O(k^2 p)$.

where f^* is value of maximum flow.

(5)



Construct a flow network as follows:

Assign a node for each family and each car and also an additional source node s and an additional sink node t.

Source node s has an edge to each family node with capacity 5.

Each car node has an ~~capacity to~~ edge to sink node t

with capacity 4.

Assign edges ~~to~~ from each family node to each car node with capacity 1.

Due to flow conservation. this construction will ensure that no two family members are assigned in the same ~~car~~ car.

Maximum possible flow of this ~~a~~ network is at most $5m$. Therefore $f^* \leq 5m$.

~~Eg~~ Edges present in the network is atmost

$5m + mn + 4n = O(mn)$ (The reasoning is same as Q # 4)

$\therefore |E| \leq O(mn)$

Therefore running time of ford-Fulkerson algorithm is

$O(f^*|E|) = O(m^2n)$.

(6). (a) (b) TRUE.

The reasoning for (a) and (b) is same. Since 3-COLOR is NP complete, all problems in NP (NP complete or not) can be polynomially reduced to 3-COLOR. Therefore a polynomial time solution to 3-COLOR problem implies every problem in NP can be solved in polynomial time.

(c) FALSE. While any NP complete probem may be polynomially reduced to 3-COLOR problem, size of the problem may increased polynomially during this reduction. In particular, suppose an NP-complete problem X, while reducing to 3-COLOR, expands its size from n to n^2 (polynomial increase). Now ~~each~~

solution of 3-COLOR will require time $O((n^2)^5) = O(n^{10})$ to solve this problem in the worst case.

7 Suppose each potential hire is represented as a node in a graph $G=(V,E)$ where there is an edge between any two nodes $u, v \in V$ if and only if u, and v have overlapping expertise.

Therefore, the decision version of the problem is given an instance graph $G=(V,E)$ and an integer k, is there a subset $V' \subset V$ such that $|V'| \geq k$ no two vertices of V' is connected by an edge?

Clearly, this is an instance of INDEPENDENCE-SET problem.

Our goal is to show INDEPENDENCE-SET is NP-complete.
To do this we will show, (i) INDEPENDENCE-SET is in NP
(ii) we will reduce it from CLIQUE which is an NP-complete problem.

First we need to show INDEPENDENCE-SET is in NP.
Given an instance of INDEPENDENCE-SET problem $G=(V,E)$ and a subset $V' \subset V$, $|V'| \geq k$, it is easy to check in polynomial time (in $|E|$ and $|V|$) whether no edges exist between any two vertices of V'.

Next we will show how to reduce CLIQUE to INDEPENDENCE-SET

CS 721: Advanced Algorithms & Analysis

NP-complete reductions

7. Suppose each potential hire is represented as anode in a graph $G = (V, E)$, where there is an edge between any two nodes $u, v \in V$ if and only if u and v have overlapping expertise. therefore the decision version of the problem is: given a graph $G = (V, E)$ and a positive integer g, is there a subset $V' \subset V, |V'| = g$ such that no two vertices of V' is connected by an edge?

Clearly this is an instance of INDEPENDENCE-SET problem.

Our goal is to prove that INDEPENDENCE-SET is NP-complete. In order to do that, we must show that (i) INDEPENDENCE-SET is in NP and (ii) a known NP-complete problem can reduced to INDEPENDENCE-SET.

First we need to show that INDEPENDENCE-SET is in NP. Given an instance of INDEPENDENCE-SET problem and a candidate solution V' where $V' \subset V$ and $|V'| = g$, it is easy to check in polynomial time (in $|E|$ and $|V)$ whether no edges exists between any pair of vertices from V' or not.

Next we will show how to reduce CLIQUE to INDEPENDENCE-SET.

Recall that the decision problem CLIQUE is defined as follows.

CLIQUE: Given an input graph $G = (V, E)$ and an integer k, is there a set of $V' \subset V$ of k vertices such that all pairwise edges between V' is present in E?

Next we will show how to reduce CLIQUE. to INDEPENDENCE-SET.

Let (G, g) be an instance of CLIQUE. Construct a new graph $G' = (V, E')$, where E' contains precisely those edges that are not present in G. Then a set of nodes S is a clique of G if and only of S is an independence set of G'. Therefore, (G', g) is an instance of CLIQUE. Clearly this construction can be done in polynomial time in $|V|$ and $|E|$.

Next we need to show that, (i) If G' has an independent set of size g, how to efficiently recover a clique of size g from G, and (ii) If G' has no independence set of size g then G has no clique of size g.

- If G' has an independent set of size g, how to efficiently recover a clique of size g from G.

  Simply choose the set of vertices that are member of independent set of of G'.

- If G' has no independence set of size g then G has no clique of size g.

  It is easier to prove contrapositive, that is, if G has a clique of size g then G' has an independent set of size g. This is again easy. Let S be a clique of size g in G. By our construction all pairwise edges between vertices in S are removed in G'. Therefore, S is an independent set of size g in G'.

8. First we show that 4-SAT is in NP. This is easy. Given an instance of a 4-SAT problem and possible truth assignment to the variables, we can check in polynomial time if the formula is true or false. Next we will reduce 3-SAT to 4-SAT. Let $\phi = C_1 \wedge C_2 \wedge \ldots \wedge C_n$ be an instance of a 3-SAT problem involving n clauses. We construct an instance $\phi' = C_1' \wedge C_2' \wedge \cdots \wedge C_n' \wedge C_{n+1}'$ of 4-SAT as follows. Introduce a new variable x and for each $i = 1, 2, \ldots n$, set $C_i' = (C_i \vee x)$ and set $C_{n+1}' = (\bar{x} \vee \bar{x} \vee \bar{x} \vee \bar{x})$. Clearly this can be constructed in polynomial time in n.

Next we need to show that, (i) If ϕ' has a satisfying truth assignment, then ϕ also has a satisfying truth assignment, and (ii) If ϕ' has no satisfying truth assignment then ϕ has no satisfying truth assignment.

- If ϕ' has a satisfying truth assignment, then ϕ also has a satisfying truth assignment

  If ϕ' has a satisfying truth assignment then C_{n+1}' must be true. This can be achieved by setting x to be false. Setting x to be false still ensures that each C_i is true. Thus ϕ is true.

* Note that $c'_{n+1}(\bar{x} \vee \bar{x} \vee \bar{x} \vee \bar{x})$ is not really necessary. We could have used $c'_1, c'_2, \ldots, c'_n$, where $c'_n = (c_n \vee x)$, and setting x = false would also work.

- If ϕ' has no satisfying truth assignment then ϕ has no satisfying truth assignment.

  It is easier to prove contrapositive, that is, if ϕ has a satisfying truth assignment then ϕ' also has a satisfying truth assignment. This is again easy. Simply set x to be false. This will ensure ϕ' is a satisfying truth assignment.

9. First we show that JOB-SCHED is in NP. This is easy. Given an instance of JOB-SCHED (J, m, t) and possible assignment of jobs to machines, we can compute total times required for each machines to complete the jobs assigned to this machine and compare the total time in polynomial time in $|J|$ and m.

Next we will show how to reduce it from PARTITION, which is NP-complete. This reduction is straight forward. Given an instance S of PARTITION, for each $x \in S$, we set a job j_x of length x. Then we construct an instance of JOB-SCHED $(S, 2, 1/2 \sum_{x \in S} x)$. Clearly, this construction can be done in polynomial time in number of elements in S.

Next we need to show that, (i) If S jobs can be assigned to 2 machines so that all jobs are finished within time $1/2 \sum_{x \in S} x$ then S can be partitioned, and (ii) If S jobs can not be assigned to 2 machines so that all jobs are finished within time $1/2 \sum_{x \in S} x$ then S can not be partitioned.

- If S jobs can be assigned to 2 machines so that all jobs are finished within time $1/2 \sum_{x \in S} x$ then S can be partitioned.

  Let J_1 and J_2 be the set of jobs assigned to machine 1 and machine 2 such that each machine takes at most $1/2 \sum_{x \in S} x$ time. But the total length of all jobs $J_1 \cup J_2$ is $\sum_{x \in S} x$. So each of these sets must be equal to exactly $1/2 \sum_{x \in S} x$.

- If S jobs can not be assigned to 2 machines so that all jobs are finished within time $1/2 \sum_{x \in S} x$ then S can not be partitioned.

  It is easier to prove contrapositive, that is, if the set S can be partitioned then S jobs can be assigned to 2 machines so that all jobs are finished within time $1/2 \sum_{x \in S} x$. Again, this is easy to show. Suppose $X, S \setminus X$ is the partition of S. For each $x \in X$, assign job j_x to machine 1, assign the remaining jobs to machine 2. Clearly total time requires to finish all the jobs in each machine is $1/2 \sum_{x \in S} x$.