ASSIGNMENT COL351

Harikesh(2019CS10355),Sourav(2019CS10404)

November 2021

1 Problem

Given: G(V,E) is a directed graph with source = s and T = $t_1, t_2, \ldots\ldots, t_k \subseteq$ V is the set of terminals. For any X $\subseteq$ E, r(X) is the number of vertices $v \in T$ which are reachable from s in G-X i.e. G(V, E \X).
Need an O($|T| * |E|$) time algorithm to find a set X of edges that minimizes r(X) + $|X|$.
Algorithm intuition: As here we are given a graph G(V,E) along with a single source and many terminals. So, we have to somehow make it a problem of single source and single sink. For that one of the ways is to assume the set $t_1, t_2, \ldots\ldots, t_n$ a single vertex and then doing the minimization of r(X) + $|X|$ but this will cause a problem in calculating r(X) as we have to find the reachable terminals and that will become reductant if we have only assumed 1 terminal set of all the terminals. One of the other methods is to extend our graph G by adding some vertices and edges to it. Let G(V,E) be the given graph. We will be adding a new vertex q to the graph and then add n new edges to our graph i.e. (t_1, q), (t_2,q), ... ,(t_k,q). We then call it G' and the purpose of sink in our new graph G' is served by our new vertex q. Assume that each of the edges in G'(V', E') has a constant weight and let that be w. Now, we have to find the set X which kind of minimizes the r(X) +$|X|$. Also, the maximum possible flow from s to q can be w*k because each edge has weight/capacity w and there can be at max k flows to our new sink q.
Constraints : s != t_i for $i \in (1, k)$
Flow in an edge can be at max = the capacity of the edge.
Incoming flow at vertex = outgoing flow for vertex hold for all vertices except s and q.
Algorithm:

```
Def func opt_setX(Graph G(V,E), source s, list[terminals] t) do
    V' $\leftarrow$ V + {q}
    E' $\leftarrow$ E
    Opt_set = []
    For terminal $in$ t do
        E'$\leftarrow$ E' + {terminal, q}
    End For
    For edge $\in$ E' do
```

```
        weight(edge)←w #weight is same as capacity for the graph
    End For
    Graph G'(V',E') # this is the new auxiliary graph which we wanted
    For each edge (u,v) ∈ E' do
        f(u,v) ← 0
        f(v,u) ← 0
    end For
    While ∃ an augmenting s,t path p ∈ G' do
        m ← MAX_INTEGER
        For edge x ∈ path p do
            m ← min(m , weight(x))
        end For
        For each edge e ∈ p do
            f(u,v) ← f(u,v) + m
            f(v,u) ← -f(u,v)
        end For
        new_edges = []
        For each edge u,v ∈ G' do
            new_edges ← new_edges + {u,v} + {v, u}
            weight(u, v) ← weight(u,v) -f(u,v)
            weight(v,u) ← f(u, v)
        end For
        Graph G' = new Graph G'(V, new_edges)
    End While
    # now we have divided all the vertices in two halves one reachable and other ones non_reachables
    Reachables = vertices reachable from s ∈ G'
    Non_reachables = V  Reachables
    For edge e =(u, v) ∈ p do
        If u and v both ∈ reachables then
            Opt_set = Opt_set + {u ,v}
        End if
    End For
    Return Opt_set
  End func
```

Time Complexity: Initially we are forming new graph by adding 1 new vertex q which will take constant time and after that we are adding some k new vertices which will take O(k) time. After that, we are basically defining values like f(u,v) for all edges and then we are defining the weight of all the edges and all those things just take O($|E|$ + k) time as we have added some new edges also to our new auxiliary graph. Apart from that we are taking O($|V'|$ +$|E'|$) for finding the optimal cut which we are finding by iterating through the graph G'. Also, we are using a while loop till there is a s,t path p in G' and it is basically Ford-Fulkerson algorithm which run in O(max_flow*($|E|$ + k)) and we know that at a max we can have a flow of O(w*k) here w is a constant ⇒ we have a flow of O(k) because we have only k possible flows to the sink q. Hence,

the overall time complexity of the algorithm is O(k*($|E|$ +k))as k*($|E|$ +k) > k and k*($|E|$ +k)> $|E|$ and k*($|E|$ +k)> ($|E|$ +k) and hence this is the dominant term.
Now, k*($|E|$ +k) can be simplified as k*$|E|$ + k^2 and we know that k =$|T|$ and $|T| < |E|$ (total edges are more than terminal edges). So, final time complexity of this algorithm is O($|T| * |E|$).
Proof of Correctness: We will start by stating that the algorithm terminates because in the first three for loops we are just iterating through terminals and edges which will take only at max O($|E|$) time. After that we are having a while loop which is just checking if there is a path from s to t in the new augmented graph and this is the general Ford-Fulkerson and graph iterating problem and we know that both of these algorithm terminates after a certain period of time because every time we are removing the min weight graph from the cycle and then are adding a backward edge. So, one of the forward path vanishes and after a certain period of time all the forward paths vanishes and hence the algorithm terminates.
Now we have to find the Set X for which r(X) + $|X|$ minimizes and lets say C$_{min}$ is the minimum s,t cut in the graph G'. Lets start by taking any s,t cut in the graph and we can say that if that s,t cut contains let's say r of the terminals where $r \leq k$, then $|X|$ will be number of edges in that s,t cut − the edges that are terminals and are there in the cut we chose. Now, we have then divided the graph in two parts and only these r of the terminals lie in the reachable vertices from s because if any other would have been reachable we would have been added it to the reachable set. This way we are minimizing the r(X)+$|X|$ and finally got an intuition that it is related to the min s,t cut cut for maximum flow in the augmented graph. This concludes our correctness proof of the algorithm.

2 Problem

2.1 Part A:

Given: A set U = {u$_1, u_2, u_3, \ldots\ldots\ldots\ldots, u_n$}
Collection of subsets A = {A$_1, A_2, A_3, A_4, \ldots\ldots\ldots, A_m$} where A$_i$ ⊆ U.
HITTING SET: A set S is a hitting set for our collection A only if it is a such a subset of S such that $S \cap A_i \neq \phi \ \forall\ 1 \leq i \leq m$.
Prove: Hittting Set Problem is in NP class.
Hitting Set Problem: The Hitting-Set Problem (HS) for the input (U, A$_1, ..., A_m$) is to decide if ∃ a hitting-set $S \subseteq U$ of size at most k.
Proof: To show that Hitting set lies in NP class we have to just show that there exists a polynomial time algorithm which checks whether a set S of an instance I is a solution or not.
Step 1: We will first check whether the set S is of size at most k or not and for that we will just iterate through the set and increment our counter by 1 each time we hit a element. This way in O(k) we can check whether it's size is at

most k or not.
Step 2: Now we will have to check another condition which is whether S is a subset or not. For that we will iterate all the elements of S one by one and then for each of them we will search it in our set U. If we are unable to find even a single element in U then it is not a subset otherwise S is a subset of U. This step will take O(k*n) time as for every element in S we are iterating all the elements in U.
Step 3: In this step we have to check whether $S \cap A_i$ is non empty or not $\forall i$ ∈(1,m). For this we have to iterate through all the elements of S and for every element we will consider one subset of A and will check whether this element is in that subset or not (basically we have to check whether $S \cap A_i = \phi$ or not). So, for checking it for a single subset take time O(k*$|A_i|$) and also we have to check with each of the subsets of m and hence to verify all of them it will take a time O(k*$|A_i|$*m). Also , we can say that $|A_i|$ can be at most n and hence the time can be reduced to O(k*n*m).
So, now finally we have to reduce it to a polynomial time complexity in instance I i.e. (m+n). Now, $k \leq k * n \leq k * n * m$ and hence the overall time complexity is of the order of O(k*n*m). Also, we can say that k can be at most n for it to hold true because max subset possible for a set is itself of the size of the set. So, time complexity is O(n*n*m) which can be of the order of O($(m+n)^3$) and is polynomial in size of instance i.e. m+n. Therefore HS problem lies in NP class.
Algorithm:

```
Def func check_instance(S, U, A, k) do
    A = $\{A_1, A_2, \ldots\ldots\ldots\ldots, A_m\}$
    Count $\leftarrow$ 0
    For i $\in$ A do
        Count $\leftarrow$ count +1
    End for
    If count $>$ k then
        Return False
    End if
    For ele $\in$ S do
        If ele $\notin$ U then
            Return false
        End if
        If ele $\notin S \cap A_i$ then
            Return false
        End if
    End for
    Return true
End func
```

2.2 Part B:

Prove: Hitting set is NP complete by reducing vertex cover to hitting set.
Proof: We have to prove that Hitting set is NP complete problem by reducing vertex cover to hitting set. (VC $->$ HS)
Step 1: We have to first of all map an instance of vertex cover problem to Hitting set problem in polynomial time.
Vertex Cover problem: For a graph G(V,E) we have to find a subset of V i.e. S such that size of the set S is at most k and for all edges (v1, v2) in graph at least one v1 or v2 $\in$ S.
Now let's assume that $|V|$ = n and $|E|$ = m. Now we have to find a polynomial time algorithm to convert an instance to VC to HS. Let's take a function f that changes G(V,E) from into sets U, A$_1, \ldots\ldots., A_m$ such that U = V i.e. U contains all the vertices in the graph and for each of the edge in graph we are adding the pair {v1, v2} in one of the A$_i$'s (which is empty). Hence, for all the A$_i$'s from 1 to m we have taken all the pair of vertices {v1, v2} which form an edge in our graph and we can clearly see that all the A$_i$ are subsets of U only. This way we have transformed an instance of vertex cover problem to hitting set problem in O(m) time which is polynomial.
Step 2: The graph G(V,E) has a vertex cover of size k iff there is a hitting set of size k for the instance (U, A$_1, A_2, \ldots\ldots\ldots., A_m$) which we have converted earlier.
First we will try to prove forward equation by assuming that there is a vertex cover V' of size at most k. Clearly V' is a subset of V and hence subset of U. Now lets take some i in range of 1 to m and we know that we have constructed A$_i$ by adding some pair of vertices of some i^{th} edge lets say (v$_j, v_k$) for some j , k in 1 to n. And thus $V' \cap A_i \neq \phi$ for that i. This holds true for all the i's in 1 to m as we have formed our A$_i$'s for all the edges in 1 to m. Hence for all I in 1 to m $A_i \cap V' \neq \phi$. Hence we have found a vertex cover V' of size at most k such that it intersects all the sets A$_1, \ldots\ldots..A_m$. Thus this V' is equivalent to our Hitting Set. That is thus the solution of HS problem.
Now, we will try to prove it conversely by assuming that there is a hitting set of size at most k for the instance above. Hence we have a set S of size at most k such that $S \cap A_i \neq \phi$ $\forall$ $i \in$ (1,m). And we already know that we have taken the A_i's as the pair of vertices which also form edge in our graph. Thus, we have a set which has non-empty intersection with all such pair of vertices or we have at least one vertex of each edge G in that set S. This proves that there is a vertex cover of size at most k for graph G.
Thus, solution to VS exists iff solution to HS exists.
Step 3: Now finally we have to map the solution obtained by HS problem to VC problem and for that we know that hitting set problem gives us a Set H such that $\forall$ $i \in$ (1,m) $H \cap A_i \neq \phi$ and this H can be also assumed as the set of vertices containing at least one of the vertices of each edge in the graph which is just the vertex cover of our graph. Thus we can directly return H in our VC problem and that doesn't take more than constant time.
This all proves that $VC \leq p$ HS problem and from the lectures we know that

vertex cover is NP complete problem and every other problem from the NP class can be reduced to vertex cover problem. Now, above we have proved that the Vertex cover problem can be reduced to Hitting Set problem using a polynomial function. This, proves that Hitting set is also NP complete problem.

3 Problem

3.1 Part A:

Given: An undirected graph G = (V,E).
Feedback Set: A set X which is subset of V such that G-X which is G'(V, $E \setminus X$) has no cycle.
Prove: Undirected feedback set problem is in NP class.
The Undirected Feedback Set Problem (UFS): Given G and k, does there exist a feedback set of size at most k.
Proof: To show that Undirected Feedback set problem lies in NP class we have to just show that there exists a polynomial time algorithm which checks whether a set X of an instance I is a solution or not.
Step 1: We will start by checking whether the size of set X is atmost k or not and if it is more than k then we will just return false i.e. it will not be a valid UFS. This algorithm will take O(k).
Step 2: In second step we will be first modifying our original graph and then will try to form the new graph G' from it. So, now assume initially we have $|V|$ vertices and $|E|$ edges. Vertices after removing the set X are $|V| - |X|$. Also, set X contains at max k elements and so, the number of vertices in our graph are in the range $|V|$ - k to $|V|$.
Step 3: In this step we will just check whether there is a cycle or not in our modified graph and for that we will use standard DFS algorithm. Now, our modified graph has $|V|$- k vertices and number of edges are in range 0 to $|E|$. And we know that for checking cycle using DFS in an undirected graph will take time O($|V| + |E|$). And also, in our new graph we know that number of vertices $V' \leq V$ and number of edges $E' \leq E$. So, time taken to check cycle in our new graph is O($|V'| + |E'|$) which is basically O($|V| + |E|$).
From both the steps we can see that the time taken by our algorithm is O($|V| + |E|$) as in step one it was O(k) and $k \leq |V|$ because at a max we can remove all the vertices of our graph. Hence, we have found a polynomial time algorithm in $|I|$ to check whether s satisfies the given instance I or not. Hence, UFS problem is in NP class.
Algorithm:

```
Def func check_ufs_instance (G, X, k) do
    count $\leftarrow$ 0
    For i $\in$ X do
        count $\leftarrow$ count+1
    end for
    if count > k then
        Return false
    end if
    Graph G'($|V'|$, $|E'|$) $< - G \setminus X$
    Return if_cycle(G')
End func
Def func if_cycle(G') do
    m ← $|V'|$
    n ← $|E'|$
    visited = {false}* $|n + 1|$ # array of size n+1 initialized with false
    for vertex ← 1 to n do
        if visited[vertex] = false then
            if dfs(G', vertex, -1, visited) = true
                Return false #cycle present
            end if
        end if
    end for
    Return true #cycle not present
End func
Def func dfs(G', vertex, parent, visited) do
    Visited[vertex] = true
    For ver ∈ neighbour of vertex do
        If visited[ver] = false then
            If dfs(G' , ver, vertex, visited) then
                Return true # cycle present
            End if
        End if
        Else if ver ≠ parent then
            Return true
        End else if
        Return false
    End for
End func
```

3.2 Part B:

Prove: Undirected Feedback Set Problem is NP-complete by reducing Vertex-cover to Undirected Feedback Set Problem.
Proof: We have to prove that Undirected feedback set problem is NP complete problem by reducing vertex cover to undirected feedback set problem. (VC $\rightarrow$ UFS)
Step 1: We have to first of all map an instance of vertex cover problem to Undirected Feedback set problem in polynomial time.
Vertex Cover problem: For a graph G(V,E) we have to find a subset of V i.e. S such that size of the set S is at most k and for all edges (v1, v2) in graph atleast one v1 or v2 belongs to S.

Now let's assume that $|V|$ = n and $|E|$ = m. Now we have to find a polynomial time algorithm to convert an instance to VC to UFS. Let's take a function f that changes G(V,E) into another graph G'(V',E') which V' = V union { u$_1, u_2, \dots\dots.., u_m\}$ and for each edge er = (v$_j, v_k$) we add two more edges i.e. (v$_j, u_r$) and (v$_k, u_r$). Hence our new graph have $|V'|$ = n+m and $|E'|$ = 3m. This way we have transformed an instance of vertex cover problem to Undirected feedback problem in O(m) time which is polynomial.
Step 2: The graph G(V,E) has a vertex cover of size atmost k iff there is a undirected feedback set of size atmost k for the graph G'(V', E') which we have found earlier.
First we will try to prove forward equation by assuming that there is a vertex cover V' of size at most k. And from the definition of vertex cover we know that atleast one of the vertex of any edge (e$_r$ = (v$_j, v_k$) ∀ r ∈ (1,m) and j ,k ∈ (1,n)) in the graph belongs to the vertex cover set that is V'. Also we can say that if there exists a cycle in G'(V',E') then it must contain an edge let's e$_r$ = (v$_j, v_k$) ∀ r ∈ (1,m) and j ,k ∈ (1,n). And we know that vertex cover contains atleast one of the two vertices of any edge on graph. Hence, on removing that vertex corresponding to the edge er gives us that the corresponding cycle in G'(V',E') is also removed. And this way V' is the undirected feedback set of size atmost k for graph G'(V',E') i.e. V' is equivalent to undirected feedback set. That is thus the solution of our UFS.
Now, we will try to prove it conversely by assuming that there is a undirected feedback set of size atmost k for the instance of G'(V', E'). Hence, we have a set S of size atmost k such that $G' \setminus S$ contains no cycle i.e. for each edge (e$_r$ = (v$_j, v_k$) ∀ r ∈ (1,m) and j ,k ∈ (1,n)) in G, S contains one of the three vertices which are forming a cycle in G' i.e. (v$_j, v_k, u_r$). Now if the set S contains one of the vertices v$_j$ or v$_k$ then it is fine otherwise if S contains v$_r$ we can just exchange it with either of the vertices in v$_j$ or v$_k$ (that will also be undirected feedback set as cycle will be removed by any one of those 3). Thus, by both conditions we make sure that atleast one of the vertices of every edge in G(V,E) is present in the set S. This makes set S as the vertex cover of size atmost k for the graph G(V,E).Thus, solution to VS exists iff solution to UFS exists.
Step 3: Now finally we have to map the solution obtained by UFS problem to VC problem and for that we know that undirected feedback set problem gives us a set S such that for all r in 1 to m one of the vertices of e$_r$ is present in the set S and this S can be also assumed as the set of vertices containing atleast one of the vertices of each edge in the graph G(V,E) which is just the vertex cover of our graph G(V,E). Thus, we can directly return S in our VC problem and that doesn't take more than constant time.
This all proves that VC $\leq p$ UFS problem and from the lectures we know that vertex cover is NP complete problem and every other problem from the NP class can be reduced to vertex cover problem. Now, above we have proved that the Vertex cover problem can be reduced to Undirected Feedback Set problem using a polynomial function. This proves that Undirected Feedback set problem is also NP complete problem.