CSCE 629-601 Homework 7
26th October, 2016

Name: Adil Hamid Malla **UIN:** 425008306

1. Exercise 26.2-11, on page 731

INPUT: We have a Undirected $Graph = G(V, E)$.
The problem is asking to calculate the edge connectivity of the graph using the maximum flow algorithm on at most V flow networks each having the $O(V)$ vertices and $O(E)$ Edges.

Solution: We are given an undirected graph. Now according to the question we have to apply the maximum-flow algorithm on at most $|V|$ flow networks and show that we can use this method or algorithm to get the edge connectivity of the undirected graph, where the constraint is of having the $O(V)$ vertices and $O(E)$ Edges . For transforming the undirected graph to the directed graph we will make the edges between any pair of the vertices as two edges one going in the forward direction and other going in the backward direction. So the flow network graph G' will have V vertices and $2|E|$ edges corresponding to the edges we formulated between two vertices. So the property of the $O(V)$ vertices and $O(E)$ edge is maintained. Let $f(u, v)$ denote the maximum flow in the graph G'
The algorithm for the same will be:

Algorithm 1 Edge Conectivity

```
1: procedure EdgeConnectivity($G$)
2:     Let s be a vertex in V any vertex is fine
3:      define $c(u, v) = 1$ for every $c(u, v) \in E$
4:     foreach $t \in V - \{s\}$
5:     Solve the minimum cut problem or maximum flow problem in nw $(G, s, t, c)$ and let $A_t$ be the minimum cut
6:     Since the maximum flow comes from the minimm cut and vice versa , according to lemma in book
7:     Return the cut $A_t$ of minimum cost among all the minimum cuts or maximum flow $f(u, v)$
8: end procedure
```

This algorithm uses $|V| - 1$ minimum cut computations in networks, each of which can be solved by a maximum flow computation. Since each network can have a maximum flow of cost $|V| - 1$ times and all the capacities are integers, the Ford Fulkerson algorithm finds the each of the maximum flow in time $O(V.opt) = O(V.E)$ and so the overall running time of the algorithm is $O(V.E^2)$.
To see that the algorithm finds the global min cut, let k be edge-connectivity of the graph, E^* be a set of k edges whose removal disconnects the graph, and let A^* be the connected component containing s in the disconnected graph resulting from the removal of the edges in E^*. So A^* is a global minimum cut of cost at most and exactly k , and it contains s.
In at least one iteration, the algorithm constructs a network (G,s,t,c) in which $t \notin A^*$, which means that A^* is a valid cut, of capacity k, for the network, and so when the algorithm finds a minimum capacity cut in the network it must find a cut of capacity at most k indeed, exactly k. This means that, for at least one t, the cut A_t is also an optimal global min-cut.

1. Proof that $k \geq min(f(u, v))$ .Let $m = min(f(u, v))$. Suppose we remove only m-1 edges from G. For any vertex v, by the max-flow min-cut theorem, u and v are still connected. The max flow from u to v is at least m, hence any cut separating u from v has capacity at least m, which means at least m edges cross any such cut. Thus at least one edge is left crossing the cut when we remove m-1 edges. Thus every node is connected to u, which implies that the graph is still connected. So at least m edges must be removed to disconnect the graph i.e., $k \geq min(f(u, v))$

2. Proof that $k \leq min(f(u, v))$. Consider a vertex v with the $f(u, v)$. By the max-flow min-cut theorem, there is a cut of capacity $f(u, v)$ separating u and v. Since all edge capacities are 1, exactly $f(u, v)$ edges cross this cut. If these edges are removed, there is no path from u to v, and so our graph becomes disconnected. Hence $k \leq min(f(u, v))$.

Thus, the claim that $k = min(f(u, v))$, for any $u \in V$ is true.

2. Exercise 26.3-3 on page 735

INPUT: Given a $G = (V, E)$ bipartite graph with vertex partition $V = L \cup R$, and let G' be the corresponding flow network.
OUTPUT: Give an upper bound on the length of any augmenting path found in G' during the execution of the FORD-FULKERSON.
Solution:
The solution to this problem is very trivial. Since we know that each edge has a capacity of 1 in the flow network. Also we know that all the vertices in L can connect to only one vertex in the R, i.e matching has to be one to one.
Now lets assume that we have a flow of one from the s to t via nodes in L and R. Since this flow will result in flow of capacity one from the source s to sink t, as we already defined the highest capacity of each edge to be one. Now if we calculate the residual network after this flow G'_f, we can get the augmenting path ranging from the length of 3 i.e. source s to sink t $s \to u \to v \to t$, here the $u \in L$ and $v \in R$ and the maximum can be $s \to u_1 \to v_1 \to u_2 \to v_2 \cdots \to u_k \to v_k \to t$, where $u_i \in L$ and $v_i \in R$. Here we can not have a repeating vertex as that is the constraint in the bipartite matching that each vertex is used only once. So we can see that the augmentation is happening back and forth from L to R, it can happen as many times as possible without repeating the vertex twice. So if we see that the path contains s, t and equal number of distinct vertices in L and R. So at most the number of vertices involved will be $2 + 2 * min(|L|, |R|)$.
Getting the above in terms of the augmenting path i.e. number of edges will be $2 * min(|L|, |R|) + 1$. Thus we found out that the highest augmenting path length can happen when the flow is zero and the length of the path is $2 * min(|L|, |R|) + 1$. After which the augmenting path constantly decreases when we start using the vertices in increasing the flow.

3. Problem 26-2 , Textbook page 761.

INPUT: Given a directed acyclic graph $G = (V, E)$.
OUTPUT: Calculate the minimum path cover of G.

Solution $(a.)$ **:** Lets define our problem first. We have to calculate the minimum path cover. Path cover of a directed graph is a set P of vertex disjoint paths such that ever vertex in V is included in exactly one path. One important point is that the paths can start anywhere. Moreover, we can have a path of length 0, i.e. path consisting of single node. Now our problem statement asks us to get a minimum path cover, which contains the fewest possible paths.
According to the hint given in textbook, lets convert this problem in a bipartite matching by adding the vertices and edges in our graph G to form $G' = (V', E')$. The transformation from the G to G' is given by:

$$V' = \{x_0, x_1, \dots, x_n\} \cup \{y_0, y_1, \dots, y_n\}$$
$$E' = \{(x_0, x_i) : i \in V\} \cup \{(y_i, y_0) : i \in V\} \cup \{(x_i, y_j) : (i, j) \in E\}$$

Where we have $V = \{1, 2, 3, \dots, n\}$ is the original vertices in the graph.
Here, we can assume that x_0 is the source and y_0 is the sink. We have introduced the new set of vertices for each vertex in the graph. Now as in the bipartite graph we can make a bipartite graph from these two set of vertices. So that we have a set L and R.
Now lets run the Maximum Flow Algorithm to calculate the maximum number of the matching. Let this matching me of size $M \subseteq E'$. Suppose this matching has m edges. Now we need to get the path P in G as follows:
The Algorithm is defined as:

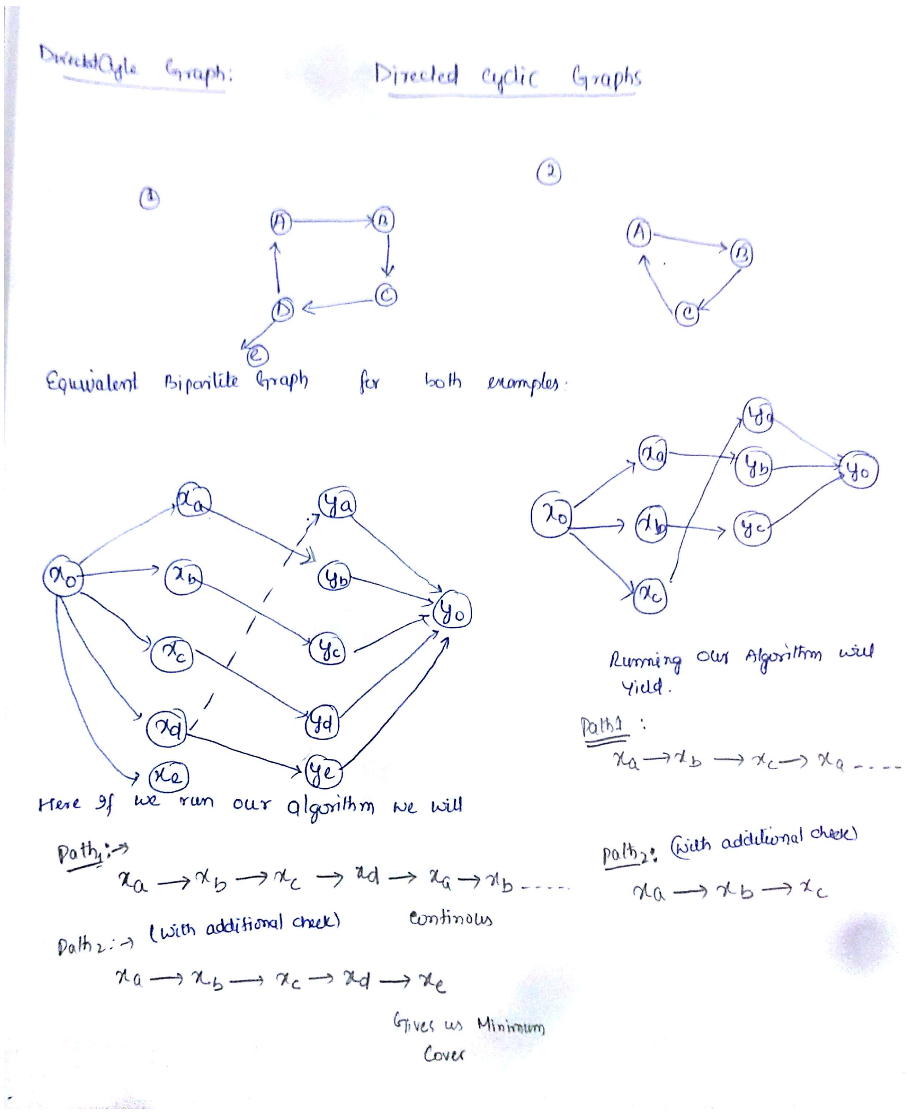
Algorithm 2 Minimum Path Cover

```
 1: procedure MinimumPath($G', M$)
 2:     Let P be set of paths , $P = \phi$
 3:     Repeat until P covers all vertices
 4:      Choose any $x_i \in V - x_0, y_0$ s.t $x_i \notin P$ and $y_i \notin M$
 5:     IncreasePath ($x_i$)
 6:     Return P
 7: end procedure
 8: procedure IncreasePath($x_i$)
 9:     $p = x_i$
10:     While $x_i$ is matched to some vertex $y_j$, set $p = p \to x_j$ and $x_i = x_j$
11:      $P = P \cup \{p\}$
12: end procedure
```

In this algorithms, a vertex x_i is in the set of paths P if there exists a path $p \in P$ such that x_i lies along P. Since the acyclicity is applied on step 4. After finishing the said algorithm we get a path cover P. We have to prove that it is the minimum path cover.
Correctness: In the path we have n-M paths. As we know that the number of the paths in P will be number of times we started with the fresh node, i.e. when we called the IncreasePath on node x_i. Also, we called IncreasePath(x_0) when we didn't have y_0 matched to anyone, so the number of such starting points will be number of unmatched vertices in $y_i \in R$ which is total vertices minus vertices matched i.e. n-M. Now, we showed that if G' has a matching with M edges, then G has a path cover with n-M paths. Is it the minimum path cover. From the previous statement we know that if G has path cover of k paths then G' has a matching with n-k edges. Now if there was a path cover in G with fewer than k = n-M paths, there would have been a matching in G' which is greater than n-k = n- (n-M) = M edges, contradicting that the M is the maximal matching.
Now to show that if G has a path cover with k paths then G' has a matching with n-k edges. Let $P = \{p_1, p_2, ..., p_k\}$ be a path cover of G. We construct a matching M$\subseteq E$ as $\{x_i, y_j\} \in M$ if $x_i \rightarrow y_j$ along a path p_i for some $i \in \{1, 2, ..., k\}$. This is really a matching of G' since any vertex x_i is matched to at most one y_j (the y_j such that $x_i \rightarrow y_j$ lies along a $p_i \in P$) and any vertex y_j is matched to a most one x_i (the x_i such that $x_i \rightarrow y_j$ lies along some $p_i \in P$). We know that M = e, where e is the number of edges in the path cover. Finally, every vertex $v \in V$ (and there are n of them) is either the initial point of a path in the path cover or else it is pointed to by a unique edge of the path cover with path length equal to zero, and so n = k + e = k + M and so M = n k. So our solution is proved. So the path cover will always be minimum as we are maximizing the flow in the bipartite graph which in turn reduces the path length of the cover.

Solution $(b.)$ **:** Now if we have a directed cyclic graphs, will the above algorithm work. The answer is yes but only if we can tweak some what in our algorithm. In the Above algorithm we can get the path cover the only change we have to introduce is one check at Line number 10. Where we say that if x_j already exists in the Path then we have reached the end of the the path and then we return to the original function. In this case the algorithm will give us the minimum cover path. The additional check was not required for the first problem as we knew that there will be no cycle in the graph which will lead us back to the vertex already added. The answer is yes, because we are always making the choice of a vertex at step 4 which may seem like it won't work, but when we have the cycle we will go into the IncreasePath function and then automatically the next vertices on x's can be added to the path since the edge already exists between two different vertices of a cycle. A simple example will help us to prove that this algorithm will work on the cyclic graphs. The intuition comes from the formulation of the graph into bipartite such that we only have an edge between the two corresponding graphs. If we have a cycle in the graph, then we can formulate the graph into the bipartite as we will be able to introduce new vertices on y side which will be able to get set of L and set of R for our bipartite matching. Thus we will always keep in adding the vertices in IncreasePath function and we will be able to terminate the search once there is no path to go or we have a new vertex coming from step 4 in our algorithm. The figure below depicts the situation in better way:

In the figure of you see the figure below, that if we a cycle also then also we will be able to solve the problem since we are making the path extend in our IncreasePath and we are stopping if we see that we are trying to add the vertex again.



~~DirectedCycle~~ Graph:

Directed cyclic Graphs

Equivalent Bipartite Graph for both examples:

Here if we run our algorithm we will

Path1:→ $x_a \to x_b \to x_c \to x_d \to x_a \to x_b$ ....

continous

Path2:→ (with additional check)

$x_a \to x_b \to x_c \to x_d \to x_e$

Gives us Minimum Cover

Running our Algorithm will yield.

Path1: $x_a \to x_b \to x_c \to x_a$ ....

Path2: (with additional check)

$x_a \to x_b \to x_c$

Figure 1: Example of the directed cyclic graph