CS218 Assignment 3

190050004 - Adarsh Raj
190050041 - Gudipaty Aniket
190050102 - Sahasra Ranjan
190050104 - Sambit Behera

May 11, 2021

Exercise 1

a **Given:** There are n jobs and m persons. We need to assign people to each job, considering their willingness to do the job, given by the list of pairs $(J_i,\ P_j)$. We need to design a flow network corresponding to this problem.

Construction as a Max Flow:
We consider a directed graph $G(V, E)$ with set of vertices $V = \{s, t\} \cup \bigcup_{i=1}^{n}\{J_i\} \cup \bigcup_{j=1}^{m}\{P_j\}$, s being the source and t being the sink, and the following edges:-

- Edges (s, J_i) with capacity 1, $1 \leq i \leq n$
- Edges (J_i, P_j) with capacity 1, corresponding to each pair (J_i, P_j) in willingness list
- Edges (P_j, t) with capacity 1, $1 \leq j \leq m$

If the max flow value of the network equals n, then we have an assignment for all jobs. Note that max flow value cannot exceed n, as total capacity of edges exiting source s is n. Also, if m < n, then we cannot have a satisfying assignment for all jobs and can return "no" directly.

This problem is a special case for part b, where the value is $d_j = 1$ for all persons. Hence, proof for this part is provided in part b.

b **Given:** There are n jobs and m persons. We need to assign people to each job, considering their willingness to do the job, given by the list of pairs $(J_i,\ P_j)$. Person i is willing to do at max d_i jobs. We need to design a flow network corresponding to this problem.

Construction as a Max Flow:
We consider a directed graph $G(V, E)$ with set of vertices $V = \{s, t\} \cup \bigcup_{i=1}^{n}\{J_i\} \cup \bigcup_{j=1}^{m}\{P_j\}$, s being the source and t being the sink, and the following edges:-

- Edges (s, J_i) with capacity 1, $1 \leq i \leq n$
- Edges (J_i, P_j) with capacity 1, corresponding to each pair (J_i, P_j) in willingness list
- Edges (P_j, t) with capacity d_j, $1 \leq j \leq m$

We say that it is possible to assign all n jobs to people, if and only if the max flow value of this flow network equals n. Note that max flow value cannot exceed n, as total capacity of edges exiting source s is n.

Equivalence between flow network and the given problem:
(Proof for parts a and b):
Now we need to show equivalence between the flow network and the given problem. We make the following observations :-

- Each edge from J_i to P_j of capacity 1 represents if the person P_j is willing to do J_1.
- If there is a non zero flow from J_i to P_j, it implies person P_j has been assigned job J_i.
- **Assignment $\implies$ Flow:** For a certain solution to the given problem, we need to show there exists an equivalent and satisfying flow in our network. If P_j is assigned k jobs, then in our network, $f^{\leftarrow}(P_j) = k$ from the corresponding k J_i nodes. A person can accept a maximum of d_j jobs, which is represented by $f^{\rightarrow}(P_j) \leq C(P_j, t) = d_j$. By flow conservation, $f^{\leftarrow}(P_j) = f^{\rightarrow}(P_j) \leq d_j$ hence satisfying the constraint that k

cannot exceed d_j.
A job can only be assigned to one person, and that is represented in the flow, as $f^{\rightarrow}(J_i) = f^{\leftarrow}(J_i) \leq C(s, J_i) = 1$. Hence, one job cannot have more than 1 edge exiting with positive flow.
If job J_i is assigned then there is a flow of 1 through the node, which means a flow of 1 from the source s, implying that if k jobs are assigned then we have a flow of k in the graph.

- **Flow $\implies$ Assignment**: We need to show that for any flow in our graph, there exists an equivalent job assignment.
In the flow, if $f(J_i, P_j) = 1$ then assign job J_i to P_j. For any node P_j, max inflow can be d_j by flow conservation and hence satisfies the limitation for person P_j.
If flow of value 1 passes through node J_i it implies that job J_i has been assigned.
Hence, if the flow value in the network is k, then a total of k jobs have been assigned.
This is true because each unit flow starting from source passes through a single node J_i.

For each flow k in the network, we can find assignment for k jobs, and for any type of k jobs assignment, we have a flow of k in the network. Hence, max flow in the network will correspond to maximum job assignments. If all jobs have been assigned, then the max flow would be equal to n. Hence, if Ford Fulkerson algorithm returns value n as the max flow, then we have one such assignment, else we can conclude that we cannot have any complete assignment. In the given problem, **(a)** part is a special case where $d_i = 1$ for all persons P_i. Hence, the above construction and algorithm is also sufficient for it.

The time complexity of this algorithm is equivalent to he time complexity of Ford Fulkerson algorithm ie $\mathcal{O}(C.|E|) = \mathcal{O}(n|E|)$.

c **Given:** There are n jobs and m persons. We need to assign people to each job, considering their willingness to do the job, given by the list of pairs $(J_i,\ P_j)$. Also, each job is assigned to be boring or interesting. We need to design a flow network corresponding to this problem.

Construction of an equivalent flow network
We consider a directed graph $G(V, E)$ with set of vertices $V = \{s, t\} \cup \bigcup_{i=1}^{n}\{J_i\} \cup \bigcup_{j=1}^{m}\{P_j\} \cup \bigcup_{j=1}^{m}\{B_j\}$, s being the source and t being the sink, and the following edges:-

- Edges (s, J_i) with capacity 1, $1 \leq i \leq n$
- Edges (J_i, P_j) with capacity 1, if J_i is an interesting job and P_j is willing to do J_i.
- Edges (J_i, B_j) with capacity 1, if J_i is a boring job and P_j is willing to do J_i.
- Edges (B_j, P_j) with capacity b_j, $1 \leq j \leq m$
- Edges (P_j, t) with capacity d_j, $1 \leq j \leq m$

If the max flow value returned by this network is n, then we have a satisfying assignment of all jobs to persons. If max flow is less than n, there is no such assignment.

Note that max flow cannot exceed n, as total flow capacity originating from source s is n.

This problem is a special case of part d, where each person has the same consideration :

of a job being boring or interesting. Hence, proof for this part is provided in part d.

d **Given:** There are n jobs and m persons. We need to assign people to each job, considering their willingness to do the job, given by the list of pairs $(J_i,\ P_j)$. Also, each job is assigned to be boring or interesting according to a person. We need to design a flow network corresponding to this problem.

Construction of an equivalent flow network
We consider a directed graph $G(V, E)$ with set of vertices $V = \{s, t\} \cup \bigcup_{i=1}^{n}\{J_i\} \cup \bigcup_{j=1}^{m}\{P_j\} \cup \bigcup_{j=1}^{m}\{B_j\}$, s being the source and t being the sink, and the following edges:-

- Edges (s, J_i) with capacity 1, $1 \le i \le n$
- Edges (J_i, P_j) with capacity 1, if P_j is willing to do job J_i and also considers it to be interesting
- Edges (J_i, B_j) with capacity 1, if P_j is willing to do job J_i but considers it to be boring
- Edges (B_j, P_j) with capacity b_j, $1 \le j \le m$
- Edges (P_j, t) with capacity d_j, $1 \le j \le m$

Equivalence between flow network and the given problem
(Proofs for parts c and d):
Now we need to show equivalence between the flow network and the given problem. We make the following observations :-

- **Assignment $\implies$ Flow**: We have a set of job assignment, let a person P_j be doing k boring jobs and l interesting jobs. Then there are k non zero flow edges to B_j from k corresponding jobs, and l non zero flow edges to P_j from l corresponding jobs. Also, as it is a satisfying assignment, we have $k \le b_j$ and $k + l \le d_j$. Hence, flow conservation occurs at node B_j and $f^{\rightarrow}(B_j) = k \le b_j$ and capacity constraint is satisfied for edge (B_j, P_j). Similarly, we have $f^{\leftarrow}(P_j) = l + f^{\rightarrow}(B_j) = k + l \le d_j$, hence $f^{\rightarrow}(P_j) = k + l \le d_j$ satisfying the capacity constraint for the edge.
  For each node J_i, if it is assigned to some person as a boring job, then there is a flow of capacity 1 to B_j, else if interesting, P_j. Also, in our assignment a job can be assigned to only person, and hence we assign $f^{\leftarrow}(J_i) = f^{\rightarrow}(J_i) = 1$ for assigned job J_i. Also, if z total jobs have been assigned, then a total flow of z originates from source to the jobs and hence, to the sink.
  In our above flow network, we have flow conservation at all nodes and capacity constraints are satisfied at all edges. Hence the implication holds.
- **Flow $\implies$ Assignment**: If we have a flow in the network, we can get a satisfying and valid assignment of jobs to persons.
  Consider a flow k in the network. If there is a non zero flow edge from J_i to P_j, assign job J_i to P_j as interesting job, and if a non zero flow edge from J_i to B_j, assign it as a boring job to P_j. Note that a job is either interesting or boring to a person, but not both.
  For a node B_j, if there is a total x incoming edges, then person P_j has been assigned x corresponding boring jobs, and by flow conservation and capacity constraint, $x \le b_j$ which satisfies the maximum capacity of P_j to do b_j boring jobs.
  For each node P_j, if there is a x flow from B_j and y total non zero flow edges from y J_i, jobs, then it implies P_j does x boring and y interesting jobs, ie a total of $x + y$

jobs, and $f^{\leftarrow}(P_j) = x + y = f^{\rightarrow}(P_j) \leq C(P_j, t) = d_j$. Hence, P_j does not do more than d_j total jobs.

Each non zero flow edge originating from source s is incoming to J_i, implying that J_i has been assigned, hence total k flow in the network implies k total jobs have been assigned to people.

Hence for each flow k in the network, we can find assignment for k jobs, and for any type of k jobs assignment, we have a flow of k in the network. Hence, max flow k in the network will correspond to maximum job assignments k. Maximum flow can be found by the Ford Fulkerson algorithm in time $(O)(C.|E|) = (O)(n|E|)$.

Exercise 2

a. Minimum Path Cover of a DAG

Problem: To find the minimum number of paths in the graph such that every vertex is contained in at least one path.

Let $G = (V, E)$ be the given directed acyclic graph, such that $|V| = n$ and $|E| = m$. For simplicity let us denote the vertices of the graph $(a_1, a_2, a_3, a_4, \dots, a_n)$.

For general graphs, finding a minimum vertex-disjoint path cover converges to finding a Hamiltonian path which makes the problem an NP-hard problem. But, for directed acyclic graphs, we can solve this problem by reducing to bipartite maximum matching, which can be solved using a max flow formulation, as shown in the class.

Now, since the above problem is not vertex-disjoint path cover as every vertex can be contained in more than one path. We can reduce the given problem as vertex disjoint path cover and then formulate it as Bipartite Matching Problem which can be converted into a flow problem. To do the above, we have to calculate the **transitive closure of the graph** and let us call it G^+. So, it turns out that solving for the case of vertex disjoint paths is easier and the solution for Q2(b) can be treated as a simplification of this solution, since it doesn't require the transitive closure of the graph in consideration.

Transitive Closure of G:

The transitive closure of a graph G is $G^+ = (V, E^+)$; namely, an edge $(v, w) \in E^+$ if and only if the vertex w is reachable in the graph G from the vertex v.

Let G^+ be the transitive closure of $G = (V, E)$. Now, consider the following set of arguments:

- If we represent a path as an ordered set of vertices, then a collection of paths $\{P_1, P_2, \dots, P_n\}$ in G can be converted to a collection of vertex-disjoint paths $\{Q_1, Q_2, \dots, Q_n\}$ in G^+ (which cover the same set of vertices in V as P_is) as follows (similar to Gram-Schmidt orthogonalisation procedure):
  1. Let $Q_1 := P_1$
  2. For $i = 2$ to n : $Q_i := P_i - \bigcup_{j=1}^{i-1} Q_j$
- In the above process, certain Q_is may be equal to the null-set, which means that the path P_i in G is redundant, in terms of covering all the vertices in G. So, number of distinct vertex-disjoint paths in G^+ corresponding to a collection of n paths in G will be at most n.
- Using the recursive use of the definition of transitive property, we can easily see that any path in G^+ corresponds to a path in G.
- Also, two vertex-disjoint paths in G^+ correspond to two distinct paths in G.
- So, we can claim that the minimum number of vertex-disjoint paths in G^+ required to cover V is equal to the minimum number of paths required in G to cover V. We can prove this using the following 2 arguments. Let k be that minimum number corresponding to G^+. Then:

1. Since there are k vertex disjoint paths in G^+ covering V, we can get k corresponding distinct paths in G which cover V.
2. Proof by contradiction: If it was possible to cover V using less than k distinct paths in G, let's consider $k-1$ such paths in G. Then we would have gotten a corresponding set of vertex-disjoint paths in G^+ of size at most $k-1$, which also cover V. This is a contradiction to the assumption that the minimum number of vertex-disjoint paths required in G^+ to cover V is k.

- Solving the case of vertex-disjoint path cover can be done using maximum matching in a particular bipartite graph, which can be converted to a max flow problem, as shown below.

Formulation as a Maximum Bipartite Matching Problem:

- Construct a new bipartite graph $G' = (X \cup Y, E')$ from transitive closure of graph G, i.e G^+.
- We will split every node of graph G^+ into two. One of these nodes will preserve outgoing edges from original node and other will preserve incoming edges.
- We can split each node $a_i \in V$ to a_{ix} and a_{iy}, such that $a_{ix} \in X$ and $a_{iy} \in Y$.
- Now, for all (i, j) $s.t., 1 \leq i, j \leq n$, if $(a_i, a_j) \in E^+$, then $(a_{ix}, a_{jy}) \in E'$. This completes the construction of E'.
- Now, we have a bipartite graph $G' = (X \cup Y, E')$ constructed from G^+.
- It can be shown that G' has a matching M of size m if and only if G^+ has a vertex-disjoint path cover of size $n - m$, where n is the number of vertices in G. This is proved below as **Lemma 2.1**.
- Now, the minimum number of vertex disjoint paths in G^+ required to cover V is equal to the minimum number of paths required in G to cover V. This is proved above in the **Transitive Closure** section.
- Hence, to find the minimum number of paths to cover all vertices in G, we have to find the maximum matching in the corresponding bipartite graph G'.

Lemma 2.1 — *If G' is a bipartite graph constructed from a graph G, using above procedure (as above construction from G^+ to G'), then vertices of G can be covered by k vertex-disjoint paths if and only if graph G' has a matching of size $|V| - k$.*

Proof. We prove the proposed equivalence of lemma in two stages:

- **k disjoint paths $\implies$ matching of size** $(|V| - k)$:

  If graph G has a vertex disjoint cover C with k paths, we can see C as a subgraph of graph G. Every node in C has in-degree either 0 and 1. It can also be concluded that there is exactly one vertex with in-degree 0 in each path in C. Hence, one can conclude that C has exactly $|V| - k$ edges (number of edges is same as the number of vertices in V which have in-degree 1 in C). Now, we can define a subset of edges of G' as follows:

$$M = \{(a_{ix}, a_{jy}) \in E' | (a_i, a_j) \in C\}$$

Now, By definition of disjoint-path cover, every vertex of G has at most one incoming edge in C and at most one outgoing edge in C. We conclude that every vertex of G' is incident to at most one edge in M; that is, M is a matching of size $|V| - k$.

- **matching of size** $(|V| - k) \implies$ **k disjoint paths**:

  If graph G' has a matching M of size $|V| - k$, then we can define a new subgraph $C = (V, M')$ from M and project it on the original graph.M' is defined from M as follows:

  $$M' = \{(a_i, a_j) \in E | (a_{ix}, a_{iy}) \in M\}$$

  Now, since M is a matching, every vertex of G has at most one incoming edge in C and at most one outgoing edge in C. We can conclude that C is a collection of disjoint directed paths in G. Also, since C includes every vertex, C defines an disjoint path cover with $|V| - k$ edges. Also, the number of paths in C is equal to the number of vertices in G that have no incoming edge in M'. Hence, it can be concluded that C contains exactly k paths.

□

Formulation as a Maximum Flow Problem:

The bipartite graph $G' = (X \cup Y; E')$ can be represented as a graph with a sink and source and capacities on the edges. In this new graph G'', we can solve the maximum flow problem to find the maximum matching for the corresponding bipartite graph.

- Given $G' = (V' = (X \cup Y), E')$. We construct $G'' = (V'', E'')$ as follows:
- $V'' = V' \cup \{s, t\}$, where s,t are two new vertices.
- For each $u \in X$, add a directed edge from s to u in G''.
- For each $v \in Y$, add a directed edge from v to t in G''.
- Let the capacity of every edge in G'' be equal to 1.

Flow Network: G'' is the flow network created from bipartite graph G', in which s and t are source and sink respectively. Now, we apply **Ford Fulkerson Algorithm** for finding maximum flow in this network.

Now, according to lemma given in the **slides**, we have:

— Let G be a bipartite graph. Let G' be the flow network constructed from G as above.

- Then, If M is a matching in G, then there is an integer valued flow f in G' such that $|f| = |M|$.
- Conversely, if f is an integer valued flow in G', then there is a matching M in G such that $|M| = |f|$.

Note: We are not proving the lemma here, as it has been referred from slides of the course.

Now, due the above lemma, maximum flow of graph G'' will correspond to the maximum matching of graph G', which will be of value (n – |min Path Cover|). Since, we know n, |min Path Cover| can be calculated.

Hence, the original problem is formulated as a maximum flow problem.

Time Complexity

We can do step by step analysis to find the time complexity of overall problem, as follows:

- Given a directed acyclic graph, we can find its transitive closure using different algorithms, each having a different upper bound for time complexity. Some of them are listed below:
  1. Floyd Warshall Algorithm - $O(n^3)$
  2. Using DFS - $O(n^2)$
  3. Purdom's Algorithm - $O(n^2)$
- We conclude that an efficient implementation for finding transitive closure will be bounded by complexity $O(n^2)$.
- Now, since transitive closure have the same number of vertices as of original graph and the worst case will be upper half of adjacency matrix of transitive closure will have all values 1. Hence, worst case complexity for constructing bipartite graph from it will be $O(n^2)$.
- Formulation of Maximum Flow problem from bipartite graph can be done in linear time.
- Solving the max-flow problem using **Ford Fulkerson Algorithm**, will have complexity $O(nm)$, where n is the number of vertices and m is the number of edges.
- The total time complexity for the given problem will be $O(n^2 + nm)$ or $O(|V|.(|V| + |E|))$

b. Minimum Vertex Disjoint Path Cover of a DAG

Problem: To find the minimum number of paths in the graph such that every vertex is contained in exactly one path.

Let $G = (V, E)$ be the given directed acyclic graph, such that $|V| = n$ and $|E| = m$. For simplicity let us denote the vertices of the graph $(a_1, a_2, a_3, a_4, ..., a_n)$.

For this sub part, **we can directly construct the bipartite graph G' from graph G, without considering transitive closure** of the given graph. Construction of bipartite graph is given below.

Formulation as a Maximum Bipartite Matching Problem:

- Construct a new graph $G' = (X \cup Y, E')$ from G.
- We will split every node of graph G into two. One of these nodes will preserve outgoing edges from original node and other will preserve incoming edges.
- We can split each node $a_i \in V$ to a_{ix} and a_{iy}, such that $a_{ix} \in X$ and $a_{iy} \in Y$.
- Now, for all $(i, j), 1 \leq i, j \leq n$, if $(a_i, a_j) \in E$, then $(a_{ix}, a_{jy}) \in E'$. This completes the construction of E'.
- Now, we have a bipartite graph $G' = (X \cup Y, E')$ constructed from G.
- Now, It can be shown that if G' has a matching M of size m if and only if G has a vertex-disjoint path cover of size $n - m$, where n is the number of vertices in G. This is proved above as **Lemma 2.1**.

- Hence, to find the minimum number of vertex disjoint paths to cover all vertices in G, we have to find the maximum matching in the corresponding bipartite graph G'.

Formulation as a Maximum Flow Problem:

As we did for the part (a) of the problem, we can reduce the bipartite matching problem to flow problem by following the same steps. The steps are also listed below:

- Given $G' = (V' = (X \cup Y), E')$. We construct $G'' = (V'', E'')$ as follows:
- $V'' = V' \cup \{s, t\}$, where s,t are two new vertices representing source and destination.
- For each $u \in X$, add a directed edge from s to u in G''.
- For each $v \in Y$, add a directed edge from v to t in G''.
- Let the capacity of every edge in G'' be equal to 1.

Flow Network: G'' is the flow network created from bipartite graph G', in which s and t are source and sink respectively. Now, we apply **Ford Fulkerson Algorithm** for finding maximum flow in this network.

Now, according to lemma given in the **slides**, we have:

— Let G be a bipartite graph. Let G' be the flow network constructed from G as above.

- Then, If M is a matching in G, then there is an integer valued flow f in G' such that $|f| = |M|$.
- Conversely, if f is an integer valued flow in G', then there is a matching M in G such that $|M| = |f|$.

Now, due the above lemma, maximum flow of graph G'' will correspond to the maximum matching of graph G', which will be of value (n – |min Path Cover|). Since, we know n, |min Path Cover| can be calculated.

In this way, vertex disjoint path cover problem can be formulated as maximum flow problem.

Time Complexity

We can do step by step analysis to find the time complexity of overall problem, as follows:

- Given a graph G, construction of a bipartite graph G' can be done in linear time in terms of vertices and edges if original graph, i.e complexity is bounded by $O(n + m)$.
- Formulation of Maximum Flow problem from bipartite graph can also be done in linear time.
- Solving the max-flow problem using **Ford Fulkerson Algorithm**, will have complexity $O(nm)$, where n is the number of vertices and m is the number of edges.
- Now, since $O(n + m)$ is bounded by $O(nm)$, the total time complexity for the given problem will be $O(nm)$ or $O(|V|.|E|)$

c. Every Edge is in exactly one path

We'll try to convert this problem into that of sub-part b. Note that if we have a graph G in which every node has at most one incoming edge and at most one outgoing edge, then the

answers to problems in parts b and c of this question become the same. So, if G has a node which has more than one incoming or outgoing edges, then we will try to create a new graph G' from G, such that the minimum size of edge-disjoint path covering of G' and G is the same.

We need to keep two facts in mind : First is that if an edge $u \longrightarrow v$ is part of a path P, then P can go beyond v only through an outgoing edge of v. Similarly , if P has to go beyond u, then it can only do that through an incoming edge of u.

This motivates us to construct a graph G' from G as follows:

- Let G' := (V', E'), given G := (V, E)
- For a vertex u in V. Suppose u has m incoming edges and n outgoing edges in G. Then we insert $r = max(m, n)$ copies of the node u in $G'(V')$. Let's denote them by $u_1, u_2, \ldots, u_r$ The i-th incoming edge (if there is one) of u in G becomes the only incoming edge of u_1. Similarly, the i-th outgoing edge (if there is one) of u in G become the only outgoing edge of u_i in G'.
- So, every edge in G corresponds to an edge in G' and vice-versa. Also, the corresponding edge in G' for $u \longrightarrow v$ is $u_i \longrightarrow v_j$ for some i, j. This means that we can map any edge $u_i \longrightarrow v_j$ in G' back to the edge $u \longrightarrow v$ in G.
- This means that an edge-disjoint path cover in G' corresponds to an edge disjoint path cover in G, because every path in G' can be tracked back to a path in G.
- Now, if we consider the minimum number of edge-disjoint paths in G, then in that case, we would obviously pass $min(m, n)$ paths through a vertex u with in-degree m and out-degree n. The surplus (in or out) edges are used for either terminating a path or starting a path.
- Since u divides every path in G through u into two parts, we can re-shuffle the halves and paths terminating at u and re-shuffle the halves and paths starting at u and re-combine them, so that the total number of paths remains the same. One such recombination has corresponding paths in G' too, for all these paths in G (if this re-shuffling and re-combination is done for every vertex in the graph G).
- Using the above fact, we can easily show that the minimum size of edge-disjoint path cover in G' will be the same as the minimum size of edge-disjoint path cover in G.
- Since in G', every vertex has in-degree and out-degree of at max 1, two paths being vertex-disjoint and edge-disjoint is the same thing (equivalent). So, the minimum size of edge-disjoint path cover in G' is the same as the minimum size of vertex-disjoint path cover in G'.
- So, the final answer will be the minimum number of paths in G', such that every vertex in V' is included in exactly one path. This can be done using the procedure described in part (b) of this question.

Time complexity

Creating G' from G takes $O(|V| + |E|)$ time. After that we again take $O(|V| + |E|)$ time to convert it into the bipartite graph and the flow graph required in part (b). After that, it's just a max flow problem.

d. If the graph contains cycles

If the given graph is not acyclic, then the proposed algorithms will not work. This can be seen by considering an example of such graph and testing our algorithms on it.

Consider the graph $G = (V, E)$, where $V = \{1, 2, 3\}$ and $E = \{(1, 2), (2, 3), (3, 1)\}$. It is a directed cyclic graph.

Now, one can verify that the minimum number of paths such that every vertex is contained in at least one path is equal to 1, and that path cover is $C = \{(1, 2, 3)\}$. For the vertex disjoint case also, the minimum number of paths such that every vertex is in exactly one path, is also equal to 1, and the corresponding path cover is $C' = \{(1, 2, 3)\}$. Also for the edge-disjoint part, the minimum number of paths required is 1, which is the path $(1, 2, 3)$.

Now, we test our algorithm on the given graph for part (a):

- The transitive closure of the graph G is $G^+ = (\{1, 2, 3\}, \{(1, 2), (1, 3), (2, 1), (2, 3), (3, 1), (3, 2)\})$.
- Constructing bipartite graph from G^+, we get $G' = ((X \cup Y), E')$, where $X = \{1_x, 2_x, 3_x\}$ and $Y = \{1_y, 2_y, 3_y\}$, and $E' = \{(1_x, 2_y), (1_x, 3_y), (2_x, 1_y), (2_x, 3_y), (3_x, 1_y), (3_x, 2_y)\}$.
- In the above graph a maximum matching is $M = \{(1_x, 2_y), (2_x, 3_y), (3_x, 1_y)\}$, which is of size 3. According to our algorithm, size of minimum path cover will be $(n - |M|) = 3 - 3 = 0$, which is wrong. As we deduced earlier , the size of minimum path cover will 1.
- Hence, for a graph with cycles, the algorithm of part (a) can give the wrong answer.

Similarly, For part (b):

- The given graph is $G = (\{1, 2, 3\}, \{(1, 2), (2, 3), (3, 1)\})$
- Constructing bipartite graph from G^+, we get $G' = ((X \cup Y), E')$, where $X = \{1_x, 2_x, 3_x\}$ and $Y = \{1_y, 2_y, 3_y\}$, and $E' = \{(1_x, 2_y), (2_x, 3_y), (3_x, 1_y)\}$.
- In the above graph a maximum matching is $M = \{(1_x, 2_y), (2_x, 3_y), (3_x, 1_y)\}$, which is of size 3. According to our algorithm, size of minimum path cover will be $(n - |M|) = 3 - 3 = 0$, which is wrong. As we deduced earlier, the size of minimum path cover will 1, for the vertex disjoint case also.
- Hence, for a graph with cycles, the algorithm of part (b) can give the wrong answer.

For part (c):

- Since in the counter example every vertex has at most one incoming edge and at most one outgoing edge. The construction defined in part (c) will result in same graph, i.e $G' = (\{1, 2, 3\}, \{(1, 2), (2, 3), (3, 1)\})$
- Since, the altered graph is same as original graph, algorithm will follow the same steps as stated for part (b) above, and hence will give the wrong answer.
- Therefore, we can conclude that the proposed algorithm can give incorrect answer, also for the case of edge-disjoint path cover.

Why the approach doesn't work with cyclic graphs?

- In part (b), it doesn't work because when we use the fact that in any path P there is a vertex v with in-degree 0 in that path, we assume that its parent in the original graph

either doesn't exist, or is a part of a different path P'. This assumption is not always valid in case of cycles , like in above counter example in which the parent of such node u (= 1) will also be a part of the same optimal path P (= (1, 2, 3)).

- In part (a), it doesn't work because of the same reason as in part (b); and along with that, the use of transitive closure will add more cycles in the graph, and since after that we will use the same max-flow formulation as in (b), we might not get the correct answer due to the same reasons.
- In part (c), if we look at the triangle graph example given above, then using the procedure described in the solution of (c), the graph G' is the same as G. Hence, G' also is a triangle shaped graph. Since after that we use the same max-flow formulation as in (b), we don't get the right answer due to the same reasons.

The above counter example proves that if the graph contains cycles then we cannot formulate the given problem as a maximum flow problem using the procedures we used for part (a), (b) and (c) of this problem.

Exercise 3

Independent Set decision problem is "Does there exist an independent set of size k".

(a) Disjoint Set: Given a set S and a collection $C = \{S_1, S_2, \dots, S_m\}$ of subsets of S, and a number k, are there k pairwise disjoint subsets in C?

Independent Set decision problem to Disjoint Set decision problem: Given graph G, consider $S = \{1, 2, \dots, n\}$ and $C = \{S_1, S_2, \dots, S_n\}$ where $S_i = \{i\}$. Now for each edge e_{ij} in graph G add a new element in S_i, S_j, S which is not already in S (the same new element will be added to all the three sets S_i, S_j and S). This will create a subset (in polynomial time complexity) and a collection such that, S_i, S_j are disjoint iff there is no edge between vertices v_i, v_j in graph G. Thus, finding an independent set of size k in the graph G reduces to finding k pairwise disjoint subsets in C.

Disjoint Set decision problem to Independent Set decision problem: We define a graph G of size m with vertex v_i and edge e_{ij} between v_i, v_j iff $S_i \cap S_j \neq \phi$. For this construction, k disjoint subset in the collection C will imply, there are k vertices in the graph with no edge b/w any of those k vertices, and vice versa . Thus finding an independent set of size k in the graph G will imply that there are k disjoint subsets in C. Thus, finding a pairwise disjoint subsets in C reduces to finding an independent set of size k in the graph G.

For both the sides it is a direct conversion from one representation to other which can be trivially done in polynomial time complexity

(b) Given a graph G and a number k, does G contain a dominating set of size at most k? A dominating set is a subset of vertices such that every vertex not in the subset is adjacent to some vertex in the subset.

Vertex Cover problem to the Dominating Set problem Construct a new graph G' as follows:

- Initialise G' with same vertex set and edge set as G.
- Add a new vertex for each edge in G, such that there now exists a vertex v_{ij} in G' for each edge e_{ij} in G, and connect this vertex to both v_i, v_j.
- So, G' has an extra vertex for each edge in G which is adjacent to the vertices of that edge.

This construction can be trivially done in polynomial time algorithm. Now, we have to prove that our graph G has a vertex cover of size k iff G' has a dominating set of size k.

- (=>): Say our graph has a vertex cover of size k, then for each edge e_{ij}, either v_i or v_j are included in the vertex cover. And since v_{ij} is adjacent to both the vertices, it is also adjacent to a vertex in the vertex cover. Therefore new vertices added in G' are also covered by the vertex cover. Which implies that G' has a dominating set of size k.
- (<=): Say graph G' has a dominating set of size k, then either the vertex in dominating set is an original vertex v_i or it is one of those added in the graph G'. Say, v_{ij} is in DS, then either v_i or v_j can replace v_{ij} and it will still be a dominating set

of same size. This might reduce the size of the set so the size of vertex cover will be atmost k. So there must be a vertex cover of size k.

Using the above argument we can generate the graph G' and find if there exists a dominating set of size k in G' which will imply there exists a vertex cover of size k in G. Or we can say, there is a dominating set of size k in G' iff there is a vertex cover of size k in G. Thus vertex cover decision problem reduces to dominating set decision problem.

Dominating set problem to Vertex Cover problem Construct a new graph G' of size n with vertices as in the original graph G. Add vertices v_{ij} for all $1 \leq i, j \leq n$ and a vertex v'_{ij} if there is no edge b/w v_i, v_j in the original graph. Now, add edges as follows:

- Add edge b/w v_{ij}, v_{ik} for all $j \neq k$. (n cliques of size n)
- Add edge b/w v_i & $v_{i,j}$ and similarly v_i & $v_{j,i}$ if there is an edge b/w v_i, v_j in the original graph G.
- Add edge b/w v_{ij}, v'_{ij} if there is no edge between v_i and v_j in G.

Since we are adding $O(n^2)$ vertices and edges in the graph G', it can be done in polynomial time complexity.

Lemma 3.1 — *G' has a vertex cover of size $n(n-1)+k$ iff G has a dominating set of size k.*

Proof. (<=) Take $n-1$ vertices from each clique formed, in total $n(n-1)$ vertices. Now, select k of the v_i vertices which are in the dominating set. This is a vertex cover for the graph G' of size $n(n-1)+k$.

(=>) If there is a vertex cover of size $n(n-1)+k$ then if it has any of the vertices v'_{ij}, remove it and add a vertex from the adjacent clique. If a clique has all n of its vertices in the vertex cover set then interchange it will a vertex v_i connected to that clique. In this way we will get exactly k of the vertices from v_i, $1 \leq i \leq n$ in the dominating set □

Now given a graph G, construct the graph G' and find if there is a vertex cover of size $n(n-1)+k$, which will imply a dominating set of size k.

For disconnected graphs we can iterate over the connected components and sum them to get a dominating set of size k given VC of size $n(n-1)+k$ and vice-versa.

So Dominating set problem reduces to the vertex cover problem and vice-versa. Now the problem asks for reduction to independent set problem. We will now show the reduction of vertex cover to independent set and vice-versa.

Lemma 3.2 — *Graph has an independent set S of size k iff it has a vertex cover $V-S$ of size $n-k$.*

Proof. (=>) Given a graph G with IS of size k, consider an edge e_{ij}. $v_i, v_j \notin S$ (S is an independent set). So $v_i, v_j \in V-S$. Thus, $V-S$ covers the edge e_{ij}.

(<=) Consider two vertices $v_i, v_j \in S$. Then there can not be an edge e_{ij} since $V-S$ is a vertex cover. Thus, no two nodes in S are joined by an edge, so S is an independent set. □

Therefore, Dominating Set problem $\equiv_p$ Vertex Cover problem $\equiv_p$ Independent set problem.

(c) Given a graph G, and a number k, is there a subset (say 2-Independent Set) of k vertices in G such that there is no path of length at most 2 between any two vertices in the subset.

2-Independent set to Independent set: Construct a new graph G' where each vertex v'_i represents a set S'_i of some vertices from the original graph G and e'_{ij} be edge connecting v'_i, v'_j. Say, $E_G(a, b) = 1$ if there is an edge b/w a and b in graph G else $E(a, b) = 0$. Define,

$$S'_i = \{x | E_G(v_i, x) = 1\} \cup v_i$$

Now add edge e'_{ij} b/w v'_i, v'_j in G' iff there is a vertex in S'_i which is adjacent to v_j in the original graph and similarly for v'_j. More formally,

$$E'_{G'}(v'_i, v'_j) \iff (\exists x \in S'_i.(E(x, v_j)) \wedge \exists x \in S'_j(E(x, v_i)))$$

Note, G and G' are undirected graphs so, $e_{ij} = e_{ji}$ and $E(a, b) \iff E(b, a)$. Also, this construction will expand the graph by a polynomial factor only and thus can be computed in polynomial time complexity.

Lemma 3.3 — *There is a path of length atmost 2 between v_i, v_j in the original graph G iff there is path of length atmost 1 between v'_i, v'_j in graph G'*

Proof. For path length = 0, it is trivially true. For path length = 1, if v_i, v_j are adjacent in graph G, then v'_i, v'_j will be adjacent to each other in graph G' since $v_j \in S'_i$ and $E_G(v_i, v_j) = 1$.

Now for path length = 2, say, $v_i \leftrightarrow x \leftrightarrow v_j$ is the path of length 2 in graph G, then $x \in S'_i, E(x, v_j) = 1$ and $x \in S'_j, E(x, v_i) = 1$. So, $E(v'_i, v'_j) = 1$, or there exists an edge between v'_i, v'_j.

For reverse, if there is a path of length atmost 1 in graph G (between v'_i and v'_j), then for path length = 0, it is trivially true. And for path length = 1 (say v'_i, v'_j are adjacent to each other), it is possible that there is a vertex v, such that $v \in S_i \cap S_j$ but $v \notin \{v_i, v_j\}$. In this case, due to the construction it will be a vertex in the mid of the path from v_i to v_j or $v_i \leftrightarrow v \leftrightarrow v_j$. Now if $v \in S_i \cap S_j$ and v is equal to either of v_i, v_j, in this case path length will be 1 (due to construction). □

Now, if there is an independent set of size k in graph G', then there is a subset of size k in graph G' in which no two vertices are adjacent to each other.

Define $S = \{v_i | v'_i \in S'\}$ where S' is the independent set of size k in the graph G'.

Lemma 3.4 — *Set S in G is a 2-Independent set iff S' in G' is an Independent set.*

Proof. Since S' is an independent set, no two elements in S' are adjacent to each other (or no path of length 1). Using the Lemma 3.3, no two elements in S have path length 1 or 2 (trivially, path length will be 0 only if $v_i = v_j$). So S is 2-Independent set. □

The construction can be done in polynomial time complexity so, the 2-Independent set problem reduces to Independent set decision problem.

Independent set to 2-Independent set Given a graph G, we construct G' as follows:

- Initialise G' as G (vertex set, not edge set).
- Mark all vertices already in the graph as set X

- For each edge e_{ij} in graph G', add a new vertex v_{ij} and connect it to both v_i, v_j. And remove the edge e_{ij} from G'.
- Mark these newly added vertices as set Y.
- If there is a vertex (say v) in X not connected to any other vertex in the graph, add a new vertex in Y and connect it to the vertex v.
- Connect all newly added vertices (set Y) with each other forming a clique.

So now every pair of newly added vertices are adjacent to each other, a pair of original vertices are distance 2 apart in G' if they shared an edge in the original graph else they are distance 3 apart in G'. This construction will take polynomial time complexity so we can use this construction for further reduction to 2-Independent set problem.

Now we find the 2-Independent set algorithm to find such a set. Also, there can't be a vertex in the newly added set Y which is at a distance more than 2 to any other vertex in the graph (due to construction). So the set we will get from 2-Independent set algorithm will have vertices from the original set of vertices only.

Any two of these vertices will be at a distance more that 2 in the graph G' and exactly equal to 3 (due to construction) iff they don't share an edge in the original graph G. And these must not be adjacent in the graph G; if they are, then there must be a vertex $v \in Y$ such that there is a path from say, $v_i \leftrightarrow v \leftrightarrow v_j$ (because of the construction) in G'. Hence, 2-Independent set will return a set of vertices which will form an independent set in the original graph G.

There is an independent set of size k in G iff there is an independent set of size k in the graph G'. Therefore, the independent set problem can be reduced to the 2-independent set problem.

(d) Given a tree T, and a collection $C = \{T_1, T_2, \ldots, T_m\}$ of subtrees of T, are there k edge-disjoint subtrees in the given collection? Show that if the subtrees are required to be vertex-disjoint, the problem can be solved in polynomial-time.

Independent Set decision problem to Disjoint collection of subtrees: Given a graph G we can construct a collection of subtrees C of size m.

Consider $C = \{T_1, T_2, \ldots, T_m\}$ where T_i is a subtree of T. Construction of T and C as follows;

- Initialise T with a single node (say n_0).
- From n_0 add n nodes marked 1 to n and add $\binom{n}{2}$ nodes marked (i, j) for $1 \le i \le j \le n$. Connect all of these to n_0.
- This is our final tree T.
- Now select subtrees. For each T_i, n_i, n_0 is in T_i (all are edge disjoint for now).
- If there is an edge b/w v_i, v_j in G, extend subtrees T_i, T_j by adding n_{ij} (if $i \le j$ else n_{ji}) to both of them. Thus they are not edge disjoint now.
- There is no cycle in the graph due to the construction. Also all the informations are encoded correctly in the construction.

This construction can be trivially done in polynomial time complexity. And there is a k edge-disjoint subtrees in the collection iff there is an independent set of size k in the graph.

Therefore, finding an independent set of size k in the graph G reduces to finding a collection of k disjoint subtrees in C.

Disjoint collection of subtrees to Independent Set decision problem: We define a graph G of size m with vertex $v_i, 1 \leq i \leq m$ and edge e_{ij} between v_i, v_j iff T_i and T_j have atleast one edge in common (trivial to find in polynomial time complexity). Now for graph G, an independent set of size k will exists iff there is a subset of vertices of size k for which no two of them have any edge in between them or we can say there exists a subset $T' = \{T_i | T_i \in C\}$ of size k for which no two of them have an edge in common with each other. Thus finding an independent set of size k in the graph G will imply that there are k edge-disjoint subtrees in C. Therefore, finding a k edge-disjoint subtrees in C reduces to finding an independent set of size k in the graph G.

Vertex-disjoint subtrees:

In this case, we will use recursion through Dynamic Programming to get the **Maximum** number of vertex disjoint subtrees of T in C. In order to use recursion and for dynamic programming, we assign an arbitrary node r as the root of T. Now, using the hierarchy in T defined by r as the root, we can see that every subtree in C has a unique root.

Now, let $T(v)$ be the largest subtree of T that is rooted at v (this is naturally the subtree corresponding to v in the DFS tree rooted at r). Now, let the maximum number of vertex-disjoint subtrees in C which are also in $T(v)$ be $DP(v)$. So, as the name suggests, we will find the value of $DP(r)$ using dynamic programming. The memoization table will be of size $|V|$, which is the number of vertices in T.

The algorithm is as follows for computing $DP(v)$ for a vertex v in T:

I Base case: If v is a leaf in v: then $DP(v) = 1$ if $\{v\}$ is a subtree in C, else $DP(v) = 0$.

II Let $c(v)$ be the set of children of v in T. Assign to the variable $cand_1$ as :

$$cand_1 := \sum_{u \in c(v)} DP(u)$$

$cand_1$ is a candidate value for $DP(v)$.

III Let $R(v)$ be the collection of subtrees in C which are rooted at v.

IV For every subtree t in $R(v)$:

i Let $s(t)$ be the set of vertices in $c(v)$ which are not in t, but adjacent to (i.e. a child of) some vertex in t.

ii Now, asssign to the variable $cand(t)$ as:

$$cand(t) := 1 + \sum_{u \in s(t)} DP(u)$$

V Now, assign to the variable $cand_2$ as:

$$cand_2 := max(0, max_{t \in R(v)} cand(t))$$

max with 0 is to handle the case when $R(v)$ is empty. In that case, $cand_2$ will be 0.

VI The value of $DP(v)$ will be :

$$DP(v) := max(cand_1, cand_2)$$

The above algorithm is correct because we used $cand_1$ and $cand_2$ to cover 2 exclusive and exhaustive cases:

- $cand_1$ is the max number of vertex disjoint subtrees of $T(v)$ in C such that none of the subtrees contains v.
- $cand_2$ is the max number of vertex disjoint subtrees of $T(v)$ in C such that v is present in exactly one subtree of the maximal collection.
- So obviously, $DP(v) = max(cand_1, cand_2)$.

Also, the above algorithm runs in polynomial time. Let $|C|$ be the number of subtrees in the collection C. Then the above algorithm runs in $O(|V|^2 + |V|^2|C|) = O(|V|^2|C|)$ time.

Note that $cand(T_i)$ for every T_i can be stored in a memoization table of size $|C|$, and then used to calculate $cand_2$ Now, we find $DP(r)$ using the above recursive algorithm through dynamic programming. (Note that $T(r) == T$)

Hence, there are k vertex-disjoint subtrees in C if and only if $DP(r) \geq k$.

References:

- https://en.wikipedia.org/wiki/Transitive_closure
- https://en.wikipedia.org/wiki/Dominating_set#L-reductions
- https://en.wikipedia.org/wiki/Ford\T1\textendashFulkerson_algorithm
- https://en.wikipedia.org/wiki/Max-flow_min-cut_theorem
- http://www.cs.kent.edu/~aleitert/iga/slides/13CliqueVerCovIndSet.pdf
- https://en.wikipedia.org/wiki/Independent_set_(graph_theory)
- https://en.wikipedia.org/wiki/L-reduction