CS 577 **Assignment 12 – More Intractability** Spring 2023

Answer the questions in the boxes provided on the question sheets. If you run out of room for an answer, add a page to the end of the document.

Name: Kayley Seow Wisc id: kseow

Reductions

1. *Kleinberg, Jon. Algorithm Design (p. 512, q. 14)* We've seen the Interval Scheduling Problem several times now, in different variations. Here we'll consider a computationally much harder version we'll call *Multiple Interval Scheduling*. As before, you have a processor that is available to run jobs over some period of time.

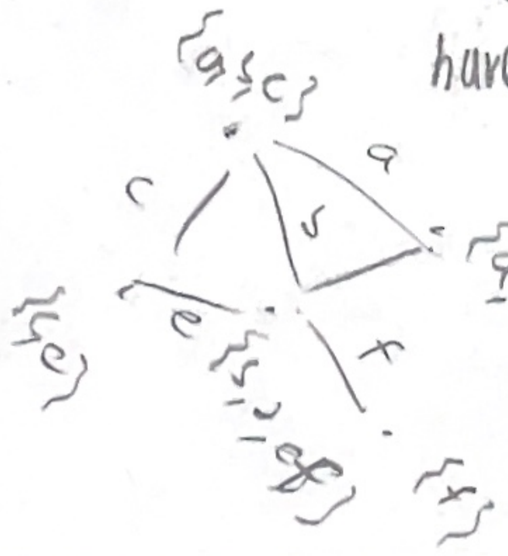
   People submit jobs to run on the processor. The processor can only work on one job at any single point in time. Jobs in this model, however, are more complicated than we've seen before. Each job requires a *set* of intervals of time during which it needs to use the processor. For example, a single job could require the processor from 10am to 11am and again from 2pm to 3pm. If you accept the job, it ties up your processor during those two hours, but you could still accept jobs that need time between 11am and 2pm.

   You are given a set of n jobs, each specified by a set of time intervals. For a given number k, is it possible to accept at least k of the jobs so that no two accepted jobs overlap in time?

   Show that Multiple Interval Scheduling is NP-Complete.

Input: Jobs with their time intervals, K at least for number of jobs. Processor to run on.
Certificate: Returns jobs that can run together, a Yes/No depending on if at least k jobs are able to run on the processor in that given time period.
Certifier: We would have a nested for loop that loops through all the possible pairs of jobs' intervals, and if the jobs overlap, we don't add it, and we check if there is at least K jobs left after the operations.
→ Problem is in NP

Reduction: Using an independent set, we would have nodes represent the jobs, and then the edges connect iff they share an interval in the set of intervals of each job.



Forwards: If we have an IS, we will still have an IS after reduction.

Backwards: If we have k jobs scheduled, between any two of them they don't share an edge, we have an IS.

Reduced problem to IS. have NP-Complete.

F: if IS → valid jobs

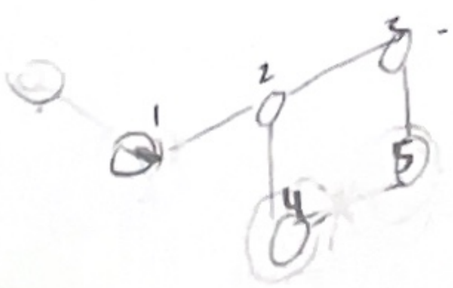
B: If k jobs scheduled, between any two of them they don't share an edge, thus be an independent set

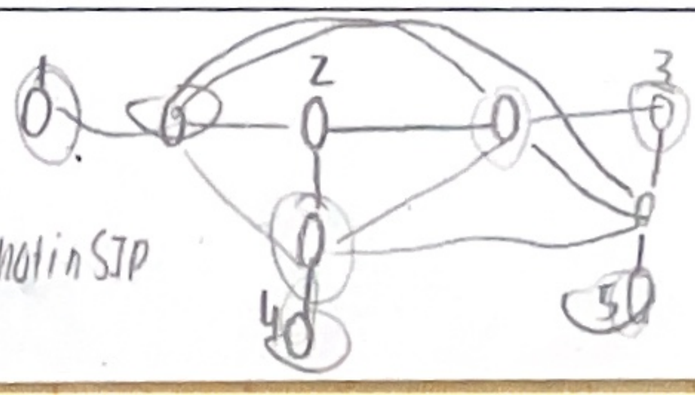
- You must solve NP-Complete problems optimally

2. *Kleinberg, Jon. Algorithm Design (p. 519, q. 28)* Consider this version of the Independent Set Problem. You are given an undirected graph G and an integer k. We will call a set of nodes I "strongly independent" if, for any two nodes $v, u \in I$, the edge (v, u) is not present in G, and neither is there a path of two edges from u to v, that is, there is no node w such that both (v, w) and (u, w) are present in G. The Strongly Independent Set problem is to decide whether G has a strongly independent set of size at least k.

Show that the Strongly Independent Set Problem is NP-Complete.

Input: Nodes and edges to a graph where we need to determine if we can have SISP.
Certificate: A yes if the graph has a SISset of at least k, and no if not.
Certifier: We would have a nested for loop which iterates through possible combinations of nodes and edges, to check if we can build a Strongly Independent Set. In this instance, if we have a SISrelationship, we would add to the set and then compare the size of the set to our K.
→ Problem is in NP, we have a certificate/certifier, and our runtime is polynomial.
Reduction: Per the Strongly Independent Set Problem, we would modify the Independent Set (NP-Hard) problem. In order to guarantee a match in solutions between the SISP and IS, we would add a node between each current node in the IS, and draw edges between each of these newly-added nodes. As such, by adding the extra node we can guarantee a Strongly IS relationship by having ≥ 2 nodes separating our nodes to include in the set, while connecting the added nodes cancels out the previous operation, making sure that it is no better to choose the added nodes than to go with our original choices.
Forward: If we have an IS, after the reduction, we will have a SIS, as we are maintaining the relationships as we add nodes and edges between the nodes, so the path could only potentially get smaller and not bigger.
Backwards: By removing the extra nodes and edges, we would revert back to a IS structure, where nodes connected directly are considered adjacent and not counted.
→ Reduce problem to NP-Hard, showing that our SISProblem is NP-Complete.





If you can't choose anything else if you have reached too far connect all the nodes you added in

3. *Kleinberg, Jon. Algorithm Design (p. 527, q. 39)* The *Directed Disjoint Paths Problem* is defined as follows: We are given a directed graph G and k pairs of nodes $(s_1, t_1), \ldots, (s_k, t_k)$. The problem is to decide whether there exist node-disjoint paths $P_1, \ldots, P_k$ so that P_i goes from s_i to t_i.

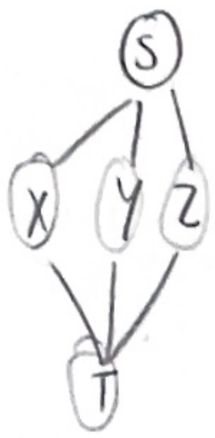
   Show that Directed Disjoint Paths is NP-Complete.

Certificate: A set of nodes where none of the paths overlap.

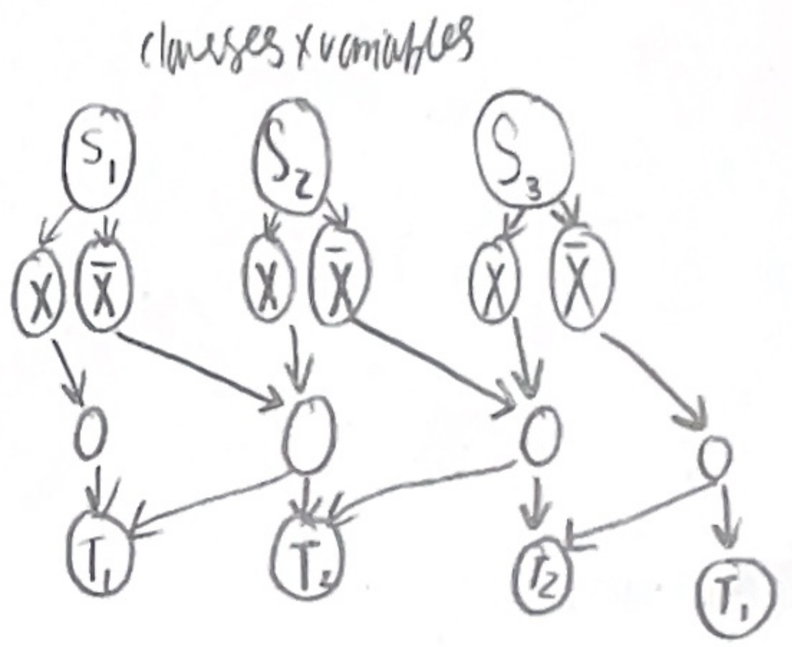
Certifier: We would have a nested for loop, where we iterate through the pairs to make sure that none of the pairs overlap.

→ Problem is in NP. We have a certificate/certifier, runtime is polynomial.

Reduction:



we need to make sure that $\bar{X}, \bar{Y}$, and $\bar{Z}$ do not satisfy



Reduction: We added intermediate nodes to signify both x and $\bar{x}$, our two options, and we add another node to our graph in order to direct the flow back to the correct T_i.

Forwards: From the 3SAT, we know there are three clauses with different options where we would add extra nodes to match S to T.

Backwards: From our modified graph to 3 SAT, we remove intermediate nodes and will return back to 3-SAT

→ NP Complete, after proving NP Hard

4. *Kleinberg, Jon. Algorithm Design (p. 508, q. 9)* The *Path Selection Problem* may look initially similar to the problem from the question 3. Pay attention to the differences between them! Consider the following situation: You are managing a communications network, modeled by a directed graph G. There are c users who are interested in making use of this network. User i issues a "request" to reserve a specific path P_i in G on which to transmit data.

   You are interested in accepting as many path requests as possible, but if you accept both P_i and P_j, the two paths cannot share any nodes.

   Thus, the Path Selection Problem asks, given a graph G and a set of requested paths $P_1, \ldots, P_c$ (each of which must be a path in G), and given a number k, is it possible to select at least k of the paths such that no two paths selected share any nodes?

   Show that Path Selection is also NP-Complete.

Input: graph G and Paths in G, to see if it is possible to select at least k.
Certificate: A yes if at least k of paths is possible, no if k is not possible.
Certifier: We are given a set P, with k paths, we get the count of each vertex used in the path → O(kV). then, we check whether the count is ≤ 1 or not O(V).
→ Problem is in NP, because of the certificate and certifier, and runtime is polynomial.
Reduction: Create graph G', where G' has vertices corresponding to each edge in G. We then make G' a complete directed graph. For each vertex V in G, there is a path P.
Forward: if we find an independent set of set k in G', it means that the paths, vertices in G', don't share any vertices. the edges. thus, we have found k paths, if we have found k parts.
Backward: If we find k-paths, that means we can find the vertices corresponding to these paths, once they don't share common vertices, then we found an independent set of size of k.
→ We reduced problem to NP-Hard, showing our PSP is NP-Complete through the IS.