CS 577 # Assignment 12 – More Intractability Spring 2025

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: Max Kamachee Wisc id: kamachee

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.

Solution:

1. Let's create an efficient certifier to show that this problem can be verified in Polynomial time:

Given an instance of jobs, extract the time intervals of each one, sort them by start-time, and check that no 2 intervals overlap with each other. This takes poly time.

2. Let's reduce from the Independent Set Problem:

- Each edge in the input graph G for the IS problem corresponds to a 1 unit timeslot such that no two distinct edges share the same timeslot.

- Each vertex corresponds with a job and is connected to the associated edges that the job needs for the job to be processed

3. Prove that this holds iff:

a. If we have a yes instance to IS where we have a subset S with |S| = k, this means that that no two vertices in S share an edge. Since we map the edges to timeslots and vertices to jobs, this means that there are k jobs in S that have no interval overlap and thus is a yes instance to Multiple Interval Scheduling.

b. If we have a yes instance to Multiple Interval Scheduling where we have a set of >=k jobs where no two of them overlap, this means that no two vertices that represent these jobs have an edge between them. This allows us to build an IS in the graph with size >=k.

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.

Solution:

1. Efficient Certifier:

- For a solution S with |S| >= k, check that every pair of nodes u,v does not share an edge and a neighbour. This can be done in Poly time.

2. Reduction from Independent Set to Strongly Independent Set:

- From the original IS graph G, add an edge between edges u and v if they are 2 edges apart. This directly connects nodes that are 2-distance in graph G

3. Prove reduction holds iff:

a. If there is a yes instance to the IS problem, that means that there are no edges between two nodes u and v. Therefore, when we construct the new graph with 2-distance edges, we cannot assign one between u and v because they are not connected whatasoever. Thus, the strongly independent set also contains the pair u,v.

b. If there is a yes instance to the Strongly Independent Set problem, this means that not only are pairs of nodes not connected to each other, but also do not share a neighbour. By virtue of the strength of its independence, this also results in a yes instance for the Independent Set problem.

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 $(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)$. 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.

Solution: 1. Directed Disjoint Paths is in NP because you can guess a list of k paths and in polynomial time verify each goes from its source to its sink and that no two share a node.

2. To show NP-hardness, we reduce from 3SAT by building a graph with one "variable" gadget per Boolean variable such that each gadget has two node-disjoint rails you choose (true or false) and one "clause" gadget per clause that has arcs feeding in from the rails corresponding to its three literals.

Then:

a. If the formula is satisfiable, pick each variable's rail to match a true/false assignment; every clause gadget will have at least one unused "true" rail to route its path, so you get all n+m disjoint paths.

b. Conversely, any solution of n+m disjoint paths must choose exactly one rail per variable (giving a truth assignment) and each clause path must hook into one of its three literal-arcs—so that literal's rail was set true—making the assignment satisfy every clause.

4. *Kleinberg, Jon. Algorithm Design (p. 508, q. 9)* The *Path Selection Problem* may look initially similar to the *Directed Disjoint Paths Problem*, but 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.

Solution: Directed Disjoint Paths is in NP because you can guess a list of k paths and in polynomial time verify each goes from its source to its sink and that no two share a node. To show NP-hardness, we reduce from 3SAT by building a graph with one "variable" gadget per Boolean variable. Each gadget has two node-disjoint rails you choose (true or false) and one "clause" gadget per clause that has arcs feeding in from the rails corresponding to its three literals. Then:

a. If the formula is satisfiable, pick each variable's rail to match a true/false assignment; every clause gadget will have at least one unused "true" rail to route its path, so you get all n+m disjoint paths.

b. Conversely, any solution of n+m disjoint paths must choose exactly one rail per variable (giving a truth assignment) and each clause path must hook into one of its three literal-arcs—so that literal's rail was set true, making the assignment satisfy every clause.

Solution: