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: Carson Miller Wisc id: 9081473028

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.

To show that MIS is in NP, we will show that it has an efficient certifier. The certificate would be the set of accepted jobs. You could iterate through each job, checking against all other jobs that it doesn't overlap. This would take at worst $O(n^2)$ time. MIS is in NP.

To reduce 3SAT to MIS, we start with n variables and m clauses. We can construct an MIS instance with n jobs and 2m intervals. Each job represents a variable and each interval represents a clause. For a given variable, we create two jobs (one for i, one for i-bar). For a given clause, we create two intervals for each job corresponding to variables in the clause. The first being from 0 to 1 and the second from 1 to 2, representing the variable being true and false, respectively.

For a yes 3SAT, if a variable is true, we select the job corresponding to i and don't select the job corresponding to i-bar, and vice versa. This way, for each clause, we accept exactly one job corresponding to the variables in the clause and there is no overlap between jobs. This would result in a yes MIS instance.

For a yes MIS, if a job is accepted, we set it's corresponding variable to true, otherwise set to false. For each clause, there must be at least one job corresponding to the variable in the clause that is accepted, resulting in a yes 3SAT instance.

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.

To show that SIS is in NP, we will show that it has an efficient certifier. The certificate would be the set of nodes that are strongly independent. You could iterate through each edge, checking against all nodes that no two nodes share an edge to a common node. This would take at worst $O(mn^2)$ time. SIS is in NP.

To reduce Vertex Cover to SIS, we start with a graph G and integer k. For each node n in G, we create two nodes n_1 and n_2 in G'. We add an edge from n_1 to n_2, and then add edges from n_1 to all of the n's neighbors (from G). Then, also add edges from n_2 to all nodes in G that weren't neighbors of n (from G).

For a yes VC, we have a set of nodes of size k that "cover" all edges from graph G. We can construct a strongly independent set of size n-k in G'. For each node in G, if n is in the VC, we add n1 to the independent set, otherwise we add n2 to the independent set. This results in a yes SIS.

For a yes SIS, we can construct a VC in G. For each node n in G, if n1 is in the strongly independent set, add n to the VC set. Otherwise, we do not add n to the VC set. This results in a yes VC.

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.

To show that DDP is in NP, we will show that it has an efficient certifier. The certificate would be the set of k paths. You could iterate through each path, checking if it goes from s to t and is node-disjoint. This would take at worst $O(n^2)$ time. DDP is in NP.

To reduce Hamiltonian Path to DDP, for each node n in we create two nodes n1 and n2 in G'. We add a directed edge from n1 and n2. Also, for every directed edge in G (u, v), we add a directed edge from u2 to v1 in G'.

For a yes Hamiltonian Path P in G, we can construct k node-disjoint paths in G'. For each node n in P, we add a path from n1 to n2 to the path P1. Then repeat adding the path from the last node of path to the first node of the next path.

For a yes DDP, we have k node-disjoint paths. We can construct a Hamiltonian Path in G. Let P be the path formed by concatenating the paths from n1 to n2 in each path P. Since the paths are node-disjoint, it follows that P is a path that visits each node in G exactly once.