CS 577 (Epic) # Assignment 12 – More Intractability Spring 2020

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: _____ Wisc id: _____

Reductions

1. *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.

Solution:

1. *Directed Disjoint Paths is in NP.*

   Given a set of paths $P_1, \ldots, P_k$, for each path P_i, iterate through the path and flag each node. If you encounter a node that is already flagged, then the solution is invalid. If you iterate through all k paths without trying to flag the same node twice, then the certificate is valid. This process takes at most time $O(|V|)$, since at worst we flag each node in G once.

2. *DDP is NP-Hard. (3-SAT $\leq_p$ DDP)*

   An arbitrary instance of 3-SAT is defined with variables $x_1, \ldots, x_m$ and k clauses. This reduction is modeled on the reduction from 3-SAT to 3D Matching discussed in class.

   For each variable x_i, create a gadget. This gadget has a core and points. The core consists of k "start" nodes $s_i^1 \ldots, s_i^k$, k "target-adjacent" nodes $u_i^1 \ldots, u_i^k$, and k "target" nodes $t_i^1 \ldots, t_i^k$, for a total of $3k$ nodes. Call the point nodes $p_i^1, \ldots, p_i^{2k}$. Draw paths $s_i^a \to p_i^{2a-1} \to u_i^a \to t_i^a$ and $s_i^a \to p_i^{2a-2} \to u_i^{a-1} \to t_i^a$ for each $a \in \{1 \ldots k\}$.

   For each clause C_j, create a start node s_j and a target node t_j. Then for each x_i in the clause, create a path $s_j \to p_i^{2j-1} \to t_j$. If $\overline{x_i}$ is in the clause, instead create a path $s_j \to p_i^{2j} \to t_j$.

   Given a 3-SAT instance with m variables and k clauses, in polynomial time this transformation generates an instance of DDP with $k(2m+1)$ paths and $|V| = k(5m+2)$.

   ($\Rightarrow$) Suppose we have a satisfying assignment. If a given variable x_i is assigned True in that assignment, let its gadget core use the "even" paths. These are disjoint by construction. If x_i is in a clause, that clause has a path from start to target through an odd point in the x_i gadget, and so this is also disjoint. If x_i is assigned False, then the gadget core can use the "odd" paths by the same logic. Because the assignment is satisfying, at least one of the variables in each clause must match, and so each clause gadget includes at least one $s \to t$ path which is disjoint from the variable gadget it touches. Thus, there exists a set of disjoint paths covering all $s \to t$ pairs in the graph.

   ($\Leftarrow$) If we have k node-disjoint paths through the graph, then for each clause gadget we have a path which selects one of the variables in that clause to hold. Let each of these variable assignments be included in our candidate satisfying assignment. By construction, each of these additions satisfies its clause. It remains to show that we never pick both x_i and $\overline{x_i}$. But because the x_i gadget core requires that we pick "odds" or "evens", we can never have clause paths that include both x_i and $\overline{x_i}$ point nodes. So our candidate satisfying assignment is satisfying, and we may complete it by assigning any unassigned variables arbitrarily.

   We have a polynomial time mapping from 3-SAT to DDP in which we have a yes instance of 3-SAT if and only if we have a yes instance of DDP. Since 3-SAT is NP-complete, DDP is NP-Hard.

2. *Kleinberg, Jon. Algorithm Design (p. 508, q. 9)* The *Path Selection Problem* may look initially similar to the problem from the previous question. 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:

1. *Path Selection is in NP.*

   Given a set of paths $P_1, \ldots, P_k$, for each path P_i, iterate through the path and flag each node. If you encounter a node that is already flagged, then the solution is invalid. If you iterate through all k paths without trying to flag the same node twice, then the certificate is valid. This process takes at most time $O(|V|)$, since at worst we flag each node in G once.

2. *PS is NP-Hard. (Independent Set $\leq_p$ PS)*

   An arbitrary instance of Independent Set is defined by a graph G. We will create a graph G' on which to run Path Selection.

   - For each edge in G, create a node in G'.
   - For each node v_i in G, create a path P_i that visits each node in G' which corresponds to an edge in G that is adjacent to v_i.

   ($\Rightarrow$) If there is a size k independent set, then we can select k nodes from G such that no edge is adjacent to more than one node in our set. Let P_i be in our candidate set of disjoint paths if and only if v_i is in this independent set. By construction, each node e_j in G' corresponds to an edge in G, so only one of its adjacent nodes in G could have been picked, and only the paths corresponding to those nodes include e_j in G'. Since we only picked one of those, our candidate set of paths is indeed disjoint.

   ($\Leftarrow$) If there is a size k set of disjoint paths, then by the same logic as above the set of nodes $\{v_i\}$ corresponding to the disjoint paths $\{P_1\}$ constitute an independent set in G.

   We have a polynomial time mapping from Independent Set to Path Selection in which we have a yes instance of IS if and only if we have a yes instance of PS. Since Independent Set is NP-complete, Path Selection is NP-Hard.

3. *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. *Multiple Interval Scheduling is in NP.*

   Given a set of jobs $J_1, \ldots, J_k$, for each job J_i, iterate through the intervals in the job and flag the processor as occupied during each. If you try to flag the processor during an interval when it is already flagged, then the solution is invalid. If you iterate through all k jobs without trying to flag the same time twice, then the certificate is valid. This process takes at most time equal to the number of available time intervals, since at worst we flag each interval once.

2. *MIS is NP-Hard. (Independent Set $\leq_p$ MIS)*

   An arbitrary instance of Independent Set is defined by a graph G.

   - For each edge e_j in G, create a time interval t_j.
   - For each node v_i in G, create a job J_i which uses the intervals t_j which correspond to edges adjacent to v_i.

   ($\Rightarrow$) If there is a size k independent set, then we can select k nodes from G such that no edge is adjacent to more than one node in our set. Let J_i be in our candidate set of non-conflicting jobs if and only if v_i is in this independent set. By construction, each time interval t_j corresponds to an edge in G, so only one of its adjacent nodes in G could have been picked, and only the jobs corresponding to those nodes require t_j. Since we only picked one of those, our candidate set of jobs is non-conflicting.

   ($\Leftarrow$) If there is a size k set of non-conflicting jobs, then by the same logic as above the set of nodes $\{v_i\}$ corresponding to the jobs $\{J_1\}$ constitute an independent set in G.

   We have a polynomial time mapping from Independent Set to Multiple Interval Scheduling in which we have a yes instance of IS if and only if we have a yes instance of MIS. Since Independent Set is NP-complete, Multiple Interval Scheduling is NP-Hard.

4. *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. 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. *Strongly Independent Set is in NP.*

   Given a set of nodes I, we can check whether any nodes $v, u \in I$ are too close together in polynomial time. For each $u \in I$, we can use BFS to see if any of the other nodes are distance 1 or 2 edges away. The requires $O(nm)$ time in total.

2. *SIS is NP-Hard. (Independent Set $\leq_p$ SIS)*

   An arbitrary instance of Independent Set is defined by a graph G. We will construct a graph G' as an instance of Strongly Independent Set.

   - For each node v_i in G, add v_i to G'.
   - For each edge $e_j = (u, v)$ in G, add a node w_j to G' and add edges (u, w_j) and (w_j, v).
   - Finally add an edge between each pair of added w nodes.

   ($\Rightarrow$) If there is a size k independent set, then the same set of nodes is a size k strongly independent set in G'. Because we have subdivided each edge in G into two edges in G', if two nodes were not adjacent in G (and both had at least one adjacent edge), the distance between them is now 3 ($u - w_i - w_j - v$).

   ($\Leftarrow$) If there is a size k strongly independent set in G', that doesn't use any of the added w nodes, then it follows by construction that they constitute an independent set in G. It remains to show that no nodes w can be part of a strongly independent set. Without loss of generality, consider a particular such node w^*. Because w^* is adjacent to all other w nodes, it can only be part of a strongly independent set if no other w node is, and additionally no other node adjacent to a w node is. In other words, w^* can only be part of a strongly independent set if it is the only node in the set (excluding nodes with no adjacent edges at all). Note that if there is a strongly independent set which includes w^*, there must exist a second one that includes no w nodes by replacing w^* with any non-w node that has at least one adjacent edge.

   We have a polynomial time mapping from Independent Set to Strongly Independent Set in which we have a yes instance of IS if and only if we have a yes instance of SIS. Since Independent Set is NP-complete, Strongly Independent Set is NP-Hard.