CS 577 **Assignment 12 – More Intractability** Spring 2024

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: ______________________

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.

First, we explain that Multiple Interval Scheduling (MIS) belongs to NP.
To do this, a schedule that fits at least k jobs without overlapping is considered proof, and this proof isn't too long.
To check this schedule, we first count the jobs listed. Next, we ensure that no more than one job occurs at the same time.
Finally, we verify that each job covers all the intervals it needs.
This checking process takes a reasonable amount of time O(mn), based on the number of jobs and intervals. So, we can confirm MIS is in NP.

Next, we show that MIS is NP-hard by adapting a problem known as Independent Set (IS).
We start with an IS problem, which involves a set of points connected by lines (a graph) and a number k.
We transform this into a MIS problem as follows:
Each point in the graph becomes a job, and each connecting line becomes a time slot that certain jobs need.
We keep the same number k. This transformation is straightforward and quick.

We then establish that if the graph has a group of k points that don't connect directly,
we can arrange k corresponding jobs without any scheduling conflicts. Conversely, if we can schedule k jobs without conflicts,
then the original k points in the graph must not be directly connected.

Therefore, MIS is confirmed as NP-complete, meaning it's both in NP and as tough as the hardest problems in NP.

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 confirm membership in NP, we simplify the process by relating it to the independent set problem.
Suppose we have a graph G and a number k from an independent set problem.
We modify this to create a new problem for the Strongly Independent Set.
For every edge e in G, we add a new vertex ve in the middle, splitting the edge into two: (u, ve) and (ve, v).
Additionally, we link all these new vertices ve to form a tight group

We then form a new graph, let's call it G0. Now, if there's a group of vertices S that's independent in G,
this same group will also be independent in G0, but more strictly so (a strongly independent set).
Conversely, if there is a strongly independent set S in G0, first consider that k must be greater than 1 for this to matter.
In this set S, none of the newly added vertices ve can be included because they are all closely connected to another vertex in G0,
which is just two steps away. Thus, S only includes original vertices from G. We then verify that these vertices still form an independent set in G.

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 $(s_1, t_1), (s_2, t_2), \dots, (s_k, t_k)$. The problem is to decide whether there exist node-disjoint paths $P_1, \dots, P_k$ so that P_i goes from s_i to t_i.

   Show that Directed Disjoint Paths is NP-Complete.

A problem is in NP if a solution to the problem can be verified in polynomial time given a certificate (a possible solution).
For the DDPP, the certificate would be the k paths (P1, P2,..., Pk) connecting each pair ((si, ti)). Verifying that these paths:
- Are indeed paths in the graph (G),
- Start at (si) and end at (ti) for each (i),
- Are node-disjoint,

can be done in polynomial time. You can check each path's validity by ensuring all its edges are present in the graph and follow consecutively, which is (O(|E|)) where (|E|) is the number of edges. You also need to ensure no node is repeated across different paths (other than potentially shared start or end nodes if allowed by problem specification). This can be done in polynomial time by marking nodes as visited. Thus, DDPP is in NP.

Step 2: DDPP is NP-hard

To show that a problem is NP-hard, we need to show that any problem in NP can be reduced to this problem in polynomial time, or reduce a know NP-complete problem to this problem. For the DDPP, we use the reduction from the NP-complete problem, Directed Hamiltonian Path Problem (DHPP).

Directed Hamiltonian Path Problem (DHPP) to DDPP

DHPP asks whether there exists a directed path in a directed graph that visits each vertex exactly once. Here's how we can reduce DHPP to DDPP:

1. **Construction**: Given a directed graph (G = (V, E)) for DHPP, construct a new instance of DDPP as follows:
   - Use the same graph (G).
   - Define k = 1, with the single pair ((s, t)), where (s) and (t) are any two nodes in (V).

2. **Reduction Logic**:
   - If there exists a Hamiltonian path in (G) from (s) to (t), this path is also a valid solution to DDPP since it connects (s) to (t) and visits all nodes, ensuring they are trivially node-disjoint (as there is only one path).
   - Conversely, if there exists a solution to the DDPP instance, it must be a path from (s) to (t) that uses all nodes in (V) exactly once (otherwise, it wouldn't use all nodes, violating the Hamiltonian path's requirement).

This reduction is polynomial in time since we only specify a single source and target, and use the original graph without modification.

Since DHPP is a known NP-complete problem, and we can reduce DHPP to DDPP in polynomial time, DDPP is NP-hard.

Conclusion

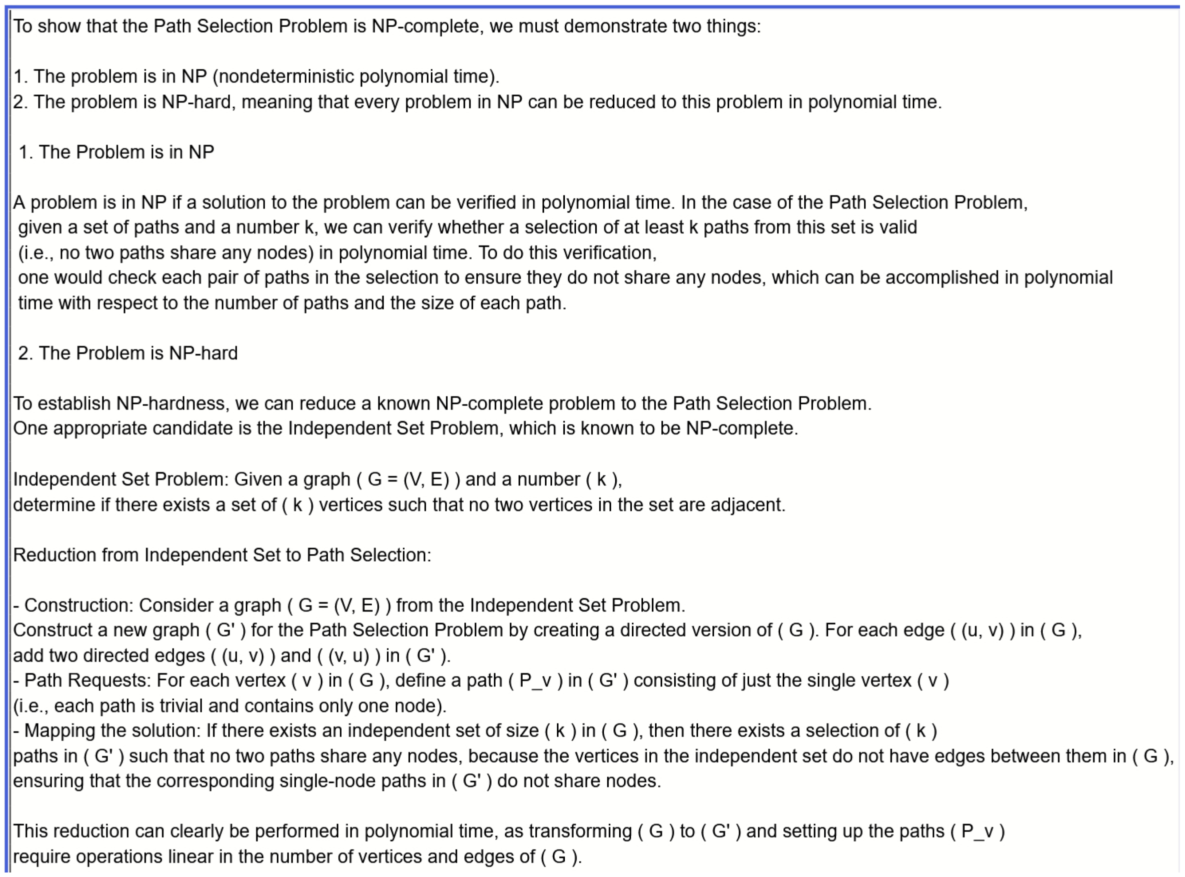
Since DDPP is both in NP and NP-hard, we conclude that the Directed Disjoint Paths Problem is NP-complete.

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.



To show that the Path Selection Problem is NP-complete, we must demonstrate two things:

1. The problem is in NP (nondeterministic polynomial time).
2. The problem is NP-hard, meaning that every problem in NP can be reduced to this problem in polynomial time.

1. The Problem is in NP

A problem is in NP if a solution to the problem can be verified in polynomial time. In the case of the Path Selection Problem, given a set of paths and a number k, we can verify whether a selection of at least k paths from this set is valid (i.e., no two paths share any nodes) in polynomial time. To do this verification, one would check each pair of paths in the selection to ensure they do not share any nodes, which can be accomplished in polynomial time with respect to the number of paths and the size of each path.

2. The Problem is NP-hard

To establish NP-hardness, we can reduce a known NP-complete problem to the Path Selection Problem.
One appropriate candidate is the Independent Set Problem, which is known to be NP-complete.

Independent Set Problem: Given a graph (G = (V, E)) and a number (k),
determine if there exists a set of (k) vertices such that no two vertices in the set are adjacent.

Reduction from Independent Set to Path Selection:

- Construction: Consider a graph (G = (V, E)) from the Independent Set Problem.
Construct a new graph (G') for the Path Selection Problem by creating a directed version of (G). For each edge ((u, v)) in (G), add two directed edges ((u, v)) and ((v, u)) in (G').
- Path Requests: For each vertex (v) in (G), define a path (P_v) in (G') consisting of just the single vertex (v)
(i.e., each path is trivial and contains only one node).
- Mapping the solution: If there exists an independent set of size (k) in (G), then there exists a selection of (k)
paths in (G') such that no two paths share any nodes, because the vertices in the independent set do not have edges between them in (G), ensuring that the corresponding single-node paths in (G') do not share nodes.

This reduction can clearly be performed in polynomial time, as transforming (G) to (G') and setting up the paths (P_v)
require operations linear in the number of vertices and edges of (G).