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: Ki Min Kang Wisc id: 908-404-5662

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.

1. Multiple Interval Scheduling is in NP:

A problem is in NP if a solution can be verified quickly.
For multiple interval scheduling, given a set of jobs and a number K, we can verify whether at least K jobs can be scheduled without overlap by checking each job's time intervals against the others. This verification process is polynomial in terms of the number of jobs and intervals, as it only requires checking each interval against the others.

2. Multiple Interval Scheduling is NP-Hard:

To show NP-Hardness, we need to reduce a known NP-complete problem to multiple interval scheduling in polynomial time.

We can show the reduction from independent set to multiple interval scheduling by:

- Assume we have a graph G for the Independent Set problem.
- For each vertex in G, we assign a job in the multiple interval scheduling problem.
- Each job has intervals corresponding to the edges of the vertex. No two jobs that are adjacent in G should have overlapping intervals.
- A solution to the independent set problem that selects K non-adjacent vertices corresponds to a solution to the multiple interval scheduling problem where at least K jobs are scheduled without overlaps.

If this reduction can be done in polynomial time, which typically involves systematic method to convert the graph into a set of jobs with intervals, we can then say that solving the independent set problem is as hard as solving multiple interval scheduling. Therefore since independent set is NP-complete, so is multiple interval scheduling.

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 demonstrate NP-Hardness, we need a polynomial-time reduction from a known NP-Complete problem to SIS. We typically use the classic Independent Set (IS) problem for this purpose.

We can outline for the reduction from IS to SIS by:

- Consider a graph G from an instance of the IS problem.
- Construct a new graph G' for the SIS problem, where for each edge (u, v) in G, we add a new node w in G', and replace edge (u, v) with two edges (u, w) and (v, w)
- A set of nodes I is an independent set in G if and only if it is strongly independent set in G'. There will be no edges directly between nodes in I, nor will there be a node in G' connected to two nodes in I because we have introduced intermediate nodes that prevent this configuration.

If we can construct G' in polynomial time from G and if a solution to the IS problem in G corresponds to a solution to the SIS problem in G', this proves that any algorithm that could solve SIS in polynomial time could also solve IS in polynomial time, making SIS NP-Hard.

Since SIS is both in NP and NP-Hard, it is NP-Complete.

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.

1. Directed Disjoint Paths in NP:

For a set of paths $P_1, P_2 \cdots P_k$ in a directed graph G, we can verify whether the paths are node-disjoint by checking each path P_i to ensure that no node is repeated on more than one path. This verification can be done in polynomial time because it simply involves scanning through each path and checking for shared nodes, which is a process that can be executed in time proportional to the total number of nodes in all paths.

2. Directed Disjoint Paths in NP-Hard:

The NP-Hardness of DDP can be established by showing a polynomial-time reduction from an NP-Complete problem to DDP. We can use 3-SAT reduction here:

- Assume a 3-SAT problem with variables $x_1, x_2, \cdots, x_m$ and clauses $c_1, c_2, \cdots, c_n$.
- Construct a directed graph G and identify nodes that represent the true and false assignments of each variable, as well as nodes representing each clause.
- For each clause variable x_i, create two paths in G: one representing x_i being true and that other representing x_i being false.
- For each clause c_j, ensure that there are paths from the variable nodes to the clause nodes only if the corresponding variable assignment would satisfy the clause.
- Now, finding node-disjoint paths from the variable assignment nodes to the clause nodes in G corresponds to finding a satisfying assignment for the 3-SAT instance.

If we can establish a mapping between solutions to the 3-SAT problem and node-disjoint paths in G, where each satisfying assignment in 3-SAT corresponds to a set of node disjoint paths in G, and we can do this in polynomial time, we've shown that any algorithm that can solve DDP can also solve 3-SAT. Hence, DDP is at least as hard as 3-SAT, making it NP-Hard.

Combining these two points confirms that the Directed Disjoint Paths Problem is NP-Complete, which also belongs to NP.

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.

1. Path Selection is in NP:
A problem is in NP if a solution to the problem can be verified quickly. In the context of the path selection problem, given a set of requested paths $P_1, P_2, \ldots P_c$ in a graph G and a number k, we can quickly verify whether at least k of these paths can be selected such that none of them share any nodes. This verification involves checking the paths one by one to ensure no nodes overlap, which is doable in the size of the graph.

2. Path Selection is NP-Hard:
To establish that the path selection problem is NP-Hard, we must show that any NP-Complete problem can be polynomially reduced to it. Since we've already established that the Directed Disjoint Paths problem is NP-Complete, we can use it for our reduction.
We can outline the reduction by:
- Assume an instance of the Directed Disjoint Paths problem with a directed graph G and a set of node pairs $(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)$.
- Map this directly to an instance of the Path Selection Problem, where each pair of nodes (s_i, t_i) corresponds to a requested path in G.
- If we can find a set of at least k node-disjoint paths in the Directed Disjoint Paths problem, we can select the same set of paths for the Path Selection problem.

If this reduction can be performed in polynomial time, then any algorithm that can solve the Path Selection Problem can solve the Directed Disjoint Paths problem, which we know is NP-Complete. This makes the Path Selection Problem NP-Hard.

Combining these two parts, we've shown that the Path Selection Problem is NP-Complete: it's in NP because we can verify a solution quickly, and it's NP-Hard because it's at least as hard as another NP-Complete problem, meaning no polynomial-time solution is likely to exist unless P = NP.