Complexity

I would recommend Chapter 8 of the Algorithms textbook as a reference:
https://people.eecs.berkeley.edu/ vazirani/algorithms/chap8.pdf.

Problem 0 – Complexity hierarchy (25%)

Consider the decision version of the job scheduling problem from the exam. We have a number of jobs to be run that all require exclusive use of a shared resource. The i^{th} job has a deadline d_i, a profit p, and takes 1 unit of time for which the job requires exclusive access of the shared resource. Only one job can use the resource at a time and you can schedule any jobs at any time before their deadline; assume that time starts at 0. Is there a scheduling of jobs with profit at least P?

1. Is it true that the decision version of Job Scheduling $\geq_p$ Node Cover? Either say it is unequivocally true (and why), it unequivocally is not true (and why), or that you cannot be 100% sure (and why).
2. Is it true that the decision version of Job Scheduling $\leq_p$ Node Cover? Either say it is unequivocally true (and why), it unequivocally is not true (and why), or that you cannot be 100% sure (and why).

Solution

Task 1

You cannot be 100% sure that the decision version of Job Scheduling $\geq_p$ Node Cover for all cases. The primary reason why we cannot be sure is due to the fact that Job Scheduling is inherently a greedy algorithm problem, whereas Node Cover is an NP-Complete problem. Semantically speaking, Node Cover will still be checking for solutions by the time Job Scheduling is finished being solved. It is possible to solve Job Scheduling in it's decision form using NP-complete, however, we cannot for certain say that it will solve it 100% of the time.

Task 2

Unlike Task 1, we can in fact be sure that the decision version of Job Scheduling $\leq_p$ Node Cover. As previously stated, Job Scheduling is inherently a greedy algorithm, and Node Cover is an NP-complete problem. Therefore, both can be ran in polynomial time, allowing for Job Scheduling to be solved, and Node Cover to check for solutions. Thus, we know that Node Cover will take longer to solve than Job Scheduling, yielding the conclusion that Job Scheduling $\leq_p$ Node Cover.

Problem 1 – Hamiltonian cycles and paths (25%)

Given a graph $G(V, E)$, a Hamiltonian path is a path in G that passes through every vertex exactly once. Given a graph $G(V, E)$, a Hamiltonian cycle is a path in G that starts and ends at the same vertex and visits every vertex exactly once (besides the start/end vertex which it visits twice).

1. Show that the problem of finding a Hamiltonian path $\geq_p$ the problem of finding a Hamiltonian cycle.
2. Show that the problem of finding a Hamiltonian cycle $\geq_p$ the problem of finding a Hamiltonian path.

Solution

Task 1

We can show that finding a Hamiltonian Path $\geq_p$ finding a Hamiltonian Cycle by definition. A Hamiltonian Cycle is a path through a graph in which every vertex is visited exactly once, and the path starts and ends at the same vertex. Similarly, a Hamiltonian Path a path through a graph in which every vertex is visited exactly once, minus the restriction of starting and ending at the same vertex. Therefore, a Hamiltonian Cycle is a Hamiltonian Path by definition. In this way, we can show that finding a Hamiltonian Path $\geq_p$ finding a Hamiltonian Cycle.

Since we know that a Hamiltonian Cycle is a Hamiltonian Path (but not necessarily vise-versa), we can compute a Hamiltonian Cycle in polynomial time, and then remove the starting/ending node in order to make it a Hamiltonian Path. Since both problems are solved in polynomial time, we can both say that they are the same hardness, and that the statement finding a Hamiltonian Path $\geq_p$ finding a Hamiltonian Cycle holds.

Task 2

We can show the reverse of Task 1, that finding a Hamiltonian Cycle $\geq_p$ finding a Hamiltonian Path, through the same procedure. By definition, a Hamiltonian Path passes through every vertex once, thus, we can add a starting/ending vertex to a Hamiltonian Path to make it a Hamiltonian Cycle. Since both problems are solved in polynomial time, we can both say that they are the same hardness, and that the statement finding a Hamiltonian Cycle $\geq_p$ finding a Hamiltonian Path holds.

Problem 2 – Math Camp (25%)

You are put in charge of recruiting counselors for UConn's annual math camp. There are n areas of mathematics for which you need a counselor who is knowledgeable to help instruct the participants. You receive applications from m counselors; each counselor is qualified to assist with a subset of the n mathematics areas. Put in another way, each of the n areas of mathematics have a subset of counselors qualified to assist students.

Show that the following problem is NP-complete. For a given number $k < m$, is it possible to hire at most k of the counselors and have at least one counselor qualified for each of the n areas of mathematics?

Solution

We can show that the above problem is NP-complete by discovering a solution for said problem.

Theorem. *The Math Camp problem is NP-complete.*

Proof. We can prove that the Math Camp problem is NP-complete by showing that it has an NP-complete solution that models the problem. In this case, the Set Cover problem is the NP-complete solution that matches the Math Camp problem. The Set Cover problem deals with finding a subset that covers all elements of a set. In this case, the set is the set of all areas of mathematics, and the subset is the set of all counselors. The Set Cover problem is NP-complete, and therefore, the Math Camp problem is NP-complete.

Furthermore, since we know that the Set Cover problem is able to verify a solution in polynomial time, we know that the Math Camp problem is also able to verify a solution in polynomial time. With the information given from the problem statement, we know it is possible to hire at most k counselors, and have at least one counselor qualified for each of the n areas of mathematics.

Therefore, we can say that the Math Camp problem is NP-complete on the basis that the Set Cover problem is known to be NP-complete. □

Problem 3 – Kernels of hardness (25%)

There are n processes on a cluster that has m distinct resources (cpu1, cpu2, I/O, disk1, disk2, and so on). A process may request multiple resources, but a resource can only be allocated to a single process at a time. If a job is granted access to all of the resources it requests, it may run; otherwise, it is stuck in a pending state.

You are charged with writing an algorithm that allocates resources to processes.

Problem: Given a set of n processes, m resources, the set of requested resources for each process, and an integer k, is it possible to allocate the resources to processes so that at least k processes are active?

For the following list of problems, give a polynomial-time algorithm or prove that the problem is NP-Complete.

1. The general problem as defined above. If you need a hint, click here.
2. The special case of the problem where $k = 2$.
3. The special case where there are two types of resources: CPU and memory access. Each process requires 0 or 1 of each resource type.
4. The special case where each resource is requested by 0, 1, or 2 processes.

Solution

Task 1

The general solution for the Kernels of Hardness (KoH) problem requires us to interpret the processes as vertices on a graph, and the shared resources as edges. The connections indicate all processes that share a resource. Given this interpretation, we can use the Independent Set problem as a reduction to the given problem. The Independent Set problem is defined as finding a set of vertices in a graph such that no two vertices are connected. In the case of KoH, this would represent processes that do not share any resources.

We can show that this is the case by constructing such a graph. First, we start with an empty graph, and add a vertex for each process. Then, we add an edge between each pair of processes that share a resource. We can then show that the Independent Set problem is NP-complete, and therefore, the KoH problem is NP-complete.

Task 2

Given that $k = 2$, we can try running all possible pairs of processes, and see if they can run simultaneously. Since we know that $k = 2$, we know that the number of possible choices is $O(n^2)$ since we must check every pair. Therefore, by trying all possible pairs, we are able to find a solution in polynomial time.

Task 3

Given that there are two types of resources (CPU and memory), we can indicate whether or not the CPU/memory is in-use for a given node N_i, and the algorithm would iterate through each node $N_0...N_n$ and see if each respective resource is already in-use. If so, the node will be marked as unavailable until the process terminates. Otherwise, if the resource on N_i is available, it will be toggled unavailable and the process will be scheduled. This will be done for all processes that are queued, and for all available nodes. Therefore, we can find a solution in polynomial time.

Task 4

Given that each resource is requested by 0, 1, or 2 processes, we can see that this is a special case of the generalized KoH problem. We can use a similar approach to Task 1 in order to show that this case is also NP-complete. We can use the same steps to construct the graph, however, we must be mindful of the fact that each resource is requested by 0, 1, or 2 processes. In this way, the problem is still able to be solved using the Independent Set problem, with $k = 2$. Therefore, we know that this case of the KoH problem is also NP-complete.