Pset 7

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 ≥ 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 ≤ 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).

Solutions

1. Job scheduling >= Node Cover is not 100% unequivocally true, but is in most circumstances. Job scheduling can be solved in Polynomial time greedily, whereas Node cover is NP-Complete. This means we can solve job scheduling in the time it takes to check a solution for Node Cover. Despite this, node cover is NP complete; therefore it is possible someone has solved NP complete or can use NP complete to solve Job Scheduling, therefore we cannot be 100% sure, but in almost all reasonable cases, this is true.
2. Job Scheduling <= Node Cover is true. Node cover is NP, meaning we can check a solution to the problem in polynomial time, and job scheduling we can solve in polynomial time. We know Node cover will always take longer than job scheduling, because we can solve job scheduling in the same ammount of time it takes to check an answer for node cover (meaning we couldn't have solved node cover in this amount of time.)

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). Show that the problem of finding a Hamiltonian path ≥ p the problem of finding a Hamiltonian cycle. Show that the problem of finding a Hamiltonian cycle ≥ p the problem of finding a Hamiltonian path.

Solutions

1. Hamiltonian cycle visits every vertex exactly once, and the start and end vertex twice; Hamiltonian path is a path that passes through every vertex once. To show Hamiltonian path >= Hamiltonian cycle, you could solve hamiltonian path with hamiltonian cycle, finding a hamiltonian cycle and then removing the final node to give you the hamiltonian path.
2. To show Hamiltonian cycle >= Hamiltonian path, you could solve hamiltonian cycle with hamiltonian path, finding a hamiltonian path and then adding the start node as your final node to make a hamiltonian cycle. These problems are equal in hardness to each other for this reason.

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

A solution to the setcover problem looks like a set of the smallest number of subsets that fill all of the elements in any particular universe. You can verify a solution to setcover in polynomial time, but cannot solve it in polynomial time. You can use setcover to solve the Math camp problem, and we know setcover is an NP-complete problem, therefore, we know math camp problem is an NP-complete problem. Each of the subjects that need a couselor to teach them could be represented by an element in the universe, and each councilor covers a set of those nodes within their set. To find the optimal set of couselors, we could use setcover to find the optimal solution to this problem. Because we can solve this problem with setcover, and setcover is NP-Complete, then we know that Mathcamp is an NP-complete problem.

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

1. A solution to this problem highlights the processes which are isolated from all other processes. This can be verified in polynomial time, but likely not solved in polynomial time. To convert this problem, our gadget would take the processes and treat them as nodes in a graph, then each of the shared resources as edges, connecting all nodes that both use the same resource. You would then be able to run independent set on these to find nodes that are independent of each other (and therefore, do not share any resources.) To prove this, you start with Independent set, then for each vertex, create a process, and for each edge, create a resource. Then solving the kernels of hardness problem would result in the same answer as independent set, therefore, we used kernels of hardness to solve independent set.
2. If K = 2, you can try all of the pairs to find two that can be run at the same time. Because K is bounded to 2 in this form of the problem, we know that the possible times we may choose in the worst case is $\binom{n}{2}$, or ~O(n^2). This means that by simply trying every possible pair, we can get a polynomial time algorithm capable of solving.
3. If there are two types of resources CPU and memory, then an algorithm to solve this problem is go through each node would be to have a boolean representing cpu and a boolean representing memory. Then, for all processes: if the process uses both resources, do nothing; if the process uses only CPU, set the boolean for CPU to true; if the process only uses memory, set the boolean for memory to true. If both CPU and Memory are true, return Yes, Else, return false.
4. This problem is a special case of the first problem, which has a very similar solution. Using the same abstraction of edges to resources and nodes to processes, we find the problem can still be used to solve the independent set problem, with k = 2. Because of this, we know this subproblem is also NP-complete.