CSE 3500 **Homework 7**

Complexity

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).

0.1 Part 1

It is not 100% true that the decision version of Job Scheduling $\geq_p$ Node Cover all the time. Since the decision version of Job Scheduling is a greedy algorithm and Node cover is a NP-complete problem, making it so by the time Job scheduling is solved, Node cover is checking for solutions. Though it is possible for someone to solve the decision version of Job scheduling using NP complete as it has become a decision problem. We cannot be 100% sure due to this but in most cases it should be possible therefore we cannot be 100% sure.

0.2 Part 2

It is true that the decision version of Job Scheduling $\leq_p$ Node Cover. Like we said in the previous part, the decision version of Job Scheduling is a greedy algorithm and Node cover is a NP-complete problem. This means we can solve job scheduling in polynomial time and check for a solution to Node cover in polynomial time. Due to this, we know that Node cover will take longer than Job scheduling and therefore Job scheduling is $\leq_p$ 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).

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

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

0.3 Part 1

We can show that the problem of finding a Hamiltonian path $\geq_p$ the problem of finding a Hamiltonian cycle by showing how both work. Since Hamiltonian cycle passes through every vertex once and returns back to the starting vertex, we can use this to solve Hamiltonian path. A Hamiltonian path works the same way where it just passes though every vertex once. To now show that Hamiltonian path $\geq_p$ the problem of finding a Hamiltonian cycle, we can use Hamiltonian cycle to pass through every vertex then when it returns back to the starting vertex, we just remove it which will give us the Hamiltonian Path. This makes both Hamiltonian path and Hamiltonian cycle equal in hardness.

0.4 Part 2

We can show that the problem of finding a Hamiltonian cycle $\geq_p$ the problem of finding a Hamiltonian path by doing the same thing as the previous part. We can use Hamiltonian path to pass through every vertex once and then add the starting vertex to create a Hamiltonian cycle. This makes both Hamiltonian path and Hamiltonian cycle equal in hardness.

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?

We can prove that the problem is NP-complete by finding a solution to the problem. The set cover problem is a problem that tries to find a subset of sets that covers all the elements which is like this problem. In a set cover problem, we have a set and we want to find a subset of that set that covers all the elements in that set. In this instance, the elements are the areas of mathematics and the sets are the counselors. Since we know that the set cover problem is NP-complete, we can use that to prove that this problem is NP-complete. The setcover problem can verify a solution in polynomial time. Since we can find a solution to the set cover problem, we now know that it is possible to hire at most k of the counselors and have at least one counselor qualified for each of the n areas of mathematics, therefore we now know that the problem is 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.

0.5 Part 1

We can use the independent set problem to solve this problem. The independent set problem is a problem that tries to find a subset of vertices that are not connected to each other. In this instance, the vertices are the processes and the edges are the resources. Since we know that the independent set problem is NP-complete, we can use that to prove that this problem is NP-complete. The independent set problem can verify a solution in polynomial time but not solve it in that time. By running a independent set, we can find vertices that do not share a resource and is able to run independently. By doing this we can solve the Kernels of Hardness problem. Since we can find a solution to the independent set problem, we can use that to solve the Kernels of Hardness problem. Therefore we know that the problem is NP-complete.

0.6 Part 2

When $k = 2$, that means that k would have a bound of 2. We can solve this in polynomial time by checking every pair in the graph that can run independently. If there are two pairs that can run independently, we can run them. If there are not two pairs that can run independently, we cannot run them. The worst case running time for this algorithm is $O(n^2)$. This is because we have to check every pair in the graph until we find two pairs that can run independently. This means that the problem can be solved in polynomial time.

0.7 Part 3

When there are two types of resources, CPU and memory access, we can solve this problem by sorting the processes by the amount of resources they need. We can then run the processes that need the least amount of resources first. We would then move onto the next process that needs the least amount of resources which would be processes that only require one. Those that only require one resource can run independently from a process that uses a diffrent single resource. This helps us maximize the amount of processes that can run. This would mean that we can solve the problem in polynomial time.

0.8 Part 4

When each resource is requested by 0, 1, or 2 processes, we can solve this problem like how we solved the first half of the problem. We can still use independent set to solve this problem with $k = 2$ as k represents how many can be active with a bound of 2. Since each resource is requested by 0, 1, or 2 processes we can use independent set to find a subset of vertices

that are not connected to each other. In this instance, the vertices are the processes and the edges are the resources. We can continue to run the processes that are not connected to each other unil all are finished. Since we know that the independent set problem is NP-complete, we can use that to prove that this problem is NP-complete.