Written Assignment 4

CS 362

April 26, 2022

Damian Franco

1 Problem 8.1

***Q*uestion:**

For each of the two questions below, decide whether the answer is (i) "Yes," (ii) "No," or (iii) "Unknown, because it would resolve the question of whether $P = NP$." Give a brief explanation of your answer

(a) Let's define the decision version of the Interval Scheduling Problem from Chapter 4 as follows: Given a collection of intervals on a time-line, and a bound k, does the collection contain a subset of non-overlapping intervals of size at least k?

Question: Is it the case that Interval Scheduling $\leq_{\mathrm{P}}$ Vertex Cover?

(b) Question: Is it the case that Independent Set $\leq_{\mathrm{P}}$ Interval Scheduling?

***S*olution(s):**

The Interval Scheduling Problem defined here gives us a collection of intervals on a time-line and a bound k. The first question gives us the Interval Scheduling Problem. Given that, we are asked if this problem is at least as hard as Vertex Cover. Secondly, we are asked if Independent Set is at least as hard as Interval Scheduling. Let me set the stage and explain some of the main concepts and problems that I must know to solve these questions.

Interval Scheduling refers to a set of tasks that are represented by an interval describing the time in which it need to be processed by some machine. A subset of intervals is compatible if no two intervals overlap on the machine. This answers the first initial question in part (a) about if the collection contains a subset of non-overlapping intervals which is true. Compatibility is a very important part in Interval Scheduling problems and contributes to the many extensions of Interval Scheduling. Interval Scheduling can be described as a NP-hard problem if the "intersection graph" approach is used. This approach has vertices as the representation of the interval and the edge between two vertices appear if and only if their intervals overlap, but some say that it can be still be solved in polynomial time.

Vertex Cover is a graph problem, In Vertex Cover, we are given a graph $G = (V,E)$ and we say that a set of node S is a subset of V where S is known as the *vertex cover* if every edge e exists in E has at least one node in s. The vertices are the part that do the "covering" within the approach and finding large vertex covers of the graph is fairly easy compared to finding smaller vertex covers. This is how the Vertex Cover problem formulates when it asks, "Given a graph G and a number k, does G contain a vertex cover of size at most k?" The problem arises with new approaches but nobody knows how to be solved in polynomial time. Regardless, Vertex cover is a NP-complete problem as noted in principle (8.16) within the Algorithm Design book by Keinberg and Tardos.

Let's actually talk about Independent Set as well. Independent Set is also a graph problem. We are given a graph $G = (V,E)$ and we say a set of nodes

S is a subset of V is independent is no two nodes in S are joined by an edge. It is fairly easy to find small independent sets, but the difficult part is finding large independent sets (somewhat opposite of Vertex Cover). The problem asks, "Given a graph G and a number k, does G contain an independent set of size at least k?" This might seem polynomial-time solvable, but when looking at this as the optimization and decision version of the problem, there is more difficulty solving this problem. This problem can also be described as an NP-complete problem (8.16).

After establishing my footing with the three paragraphs before, let me finally answer the question.

We can make an inference about the relationship between Interval Scheduling and Vertex Cover from knowing that Independent Set $\leq_P$ Vertex Cover (8.4). As I stated earlier that Interval Scheduling can be solved in polynomial time. By principle (8.1) in Algorithm Design by Keinberg and Tardos, I can state that if with the use of a black box that Vertex Cover can also be solved in polynomial time since Interval Scheduling can be solved in polynomial time. This would indicate that Interval Scheduling $\leq_P$ Vertex Cover. The answer to part (a) will be yes based off these findings.

Moving onto the next part of the question, we are asked if Independent Set is at least as hard as Interval Scheduling. The Algorithm Design book states that Independent Set $\leq_P$ Vertex Cover (8.4) and vice versa, Vertex Cover $\leq_P$ Independent Set (8.5). Both proofs for each of the comparisons use a black box to solve the respective problem on the right side of each comparison. The black box helps us decide whether G has a vertex cover/independent set of size of at most k by asking if the black box if there is also a independent set/vertex cover of size of at least/most k nodes. We can conclude the Independent Set is at least as hard as Vertex Cover. In the previous paragraph, we found that Independent Set $\leq_P$ Vertex Cover, so by using the transitive relation (8.9) we can state that since Interval Scheduling $\leq_P$ Independent Set and Independent Set $\leq_P$ Vertex Cover, then Interval Scheduling $\leq_P$ Vertex Cover. The answer to part (b) of this question will also be yes based off this inference.

Part (a): The answer for this question is ***YES***.

Part (b): The answer for this question is ***YES***.

2 Problem 8.5

***Q*uestion:**

Consider a set A = *a1, . . . , an* and a collection *B1, B2, . . . , Bm* of subsets of A (i.e., *Bi* is a subset of A for each i).

We say that a set H is a subset of A is a hitting set for the collection *B1, B2, . . . , Bm* if H contains at least one element from each *Bi*—that is, if H intersection *Bi* is not empty for each i (so H “hits” all the sets *Bi*).

We now define the Hitting Set Problem as follows. We are given a set A = *a1, . . . , an*, a collection *B1, B2, . . . , Bm* of subsets of A, and a number k. We are asked: Is there a hitting set H is a subset of A for *B1, B2, . . . , Bm* so that the size of H is at most k?

Prove that Hitting Set is NP-complete.

***S*olution(s):**

We are given a Hitting Set problem. The problem asks us, ”Is there a hitting set H is a subset of A for $B_1, B_2, . . . , B_m$ so that the size of H is at most k?” In order to prove a new problem, I must use the basic and general strategy for proving new problems are NP-complete that was presented in Algorithm Design by Keinberg and Tardos. We can describe the Hitting Set problem as X. Given a new problem X, we were provide the steps that I will take to prove this problem:

(1) Prove that X is an element of *NP*

(2) Choose a problem Y that is known to be NP-complete

(3) Prove that $Y \leq_P X$

Step 1, I must prove that the Hitting Set is in NP. To do so I must understand the full implementation of Hitting Set. A hitting set H is a cover-like problem that tries to choose at most k elements to form a hitting set within a given set$A$. Let’s say that we are given the set H, then our goal now is to check if the size of H is at most k and if at least one member of the hitting set is intersecting with any other member in the collection that is given $B_1, B_2, . . . , B_m$ which can be verified in polynomial time. Considering that this interpretation of a problem, we can conclude that the Hitting Set problem is in NP.

Step 2, now let me choose a problem that is known to be NP-complete and similar to the Hitting Set problem. The problem that I am going to compare the Hitting Set problem to is the Vertex Cover problem. Both problems are ”covering” problems due to the fact that both are asking us to find ”at most k” elements within a set (Hitting Set) or a graph (Vertex Cover). As I stated in problem 1 (and within Keinberg and Tardos’ Algorithm Design), we know that Vertex Set is in NP. To prove that it is also NP-complete, Within Algorithm Design we see that 3-Satisfiability is NP-complete (8.15). I bring up this because we also know that 3-Satisfiability $\leq_P$ Vertex Cover which can conclude that Vertex Cover is also NP-Complete due to statement (8.15) within Algorithm Design by Klenberg. This statement states the following, "If Y is an NP-complete problem, and X is a problem in NP with the property that Y P X, then X is NP-complete." This is a very important principle that will help prove that the Hitting Set problem is also NP-complete. Statement (8.16) also states that that Vertex Cove is NP-complete as well. This makes Vertex Cover a great choose for this step and to use to complete the next step.

Step 3, this final step is prove that Vertex Cover $\leq_{\text{P}}$ Hitting Set. With the explanation in the previous paragraph, we know that Vertex Cover is a known NP-complete problem that is similar to the Hitting Set problem. The use of an arbitrary instance of the Vertex Cover problem $s_{Vertex\ Cover}$ and an instance of the Hitting Set problem $s_{Hitting\ Set}$ will be used to prove that Vertex Cover $\leq_{\text{P}}$ Hitting Set.

Let's start off with $s_{Vertex\ Cover}$ and an equivalent instance of Hitting Set problem. We are given a graph G = *(V,E)* and k within $s_{Vertex\ Cover}$ and an equivalent graph for $s_{Hitting\ Set}$. The goal is to choose at most k nodes within G to form a vertex cover of the graph and at most k nodes to form the hitting set within the graph G. We can claim that the sizes for both the vertex cover set and hitting set in both instances are at most k. In both instances, we can see that each set can be considered to have a similar construction on a graph. Every vertex cover set can be considered to have "hit" nodes within it and every hitting set can be considered to have "vertex cover" nodes in it if the approaches were used on the same graph. The instance $s_{Vertex\ Cover}$ is a "yes" instance of Vertex Cover and $s_{Hitting\ Set}$ is a "yes" instance of *Hitting Set* which indicates that both instances have the same answer. This can help use conclude that both $s_{Vertex\ Cover}$ and $s_{Hitting\ Set}$ can be constructed in polynomial time since we know that Vertex Set can be constructed in polynomial time. This also indicates that Vertex Cover $\leq_{\text{P}}$ Hitting Set.

All three steps above prove that the Hitting Set problem ***is NP-complete.***

3 Problem 8.19

***Q*uestion:**

Suppose you're acting as a consultant for the port authority of a small Pacific Rim nation. They're currently doing a multi-billion-dollar business per year, and their revenue is constrained almost entirely by the rate at which they can unload ships that arrive in the port.

Handling hazardous materials adds additional complexity to what is, for them, an already complicated task. Suppose a convoy of ships arrives in the morning and delivers a total of n canisters, each containing a different kind of hazardous material. Standing on the dock is a set of m trucks, each of which can hold up to k containers.

Here are two related problems, which arise from different types of constraints that might be placed on the handling of hazardous materials. For each of the two problems, give one of the following two answers:

- A polynomial-time algorithm to solve it; or
- A proof that it is NP-complete.

(a) For each canister, there is a specified subset of the trucks in which it may be safely carried. Is there a way to load all n canisters into the m trucks so that no truck is overloaded, and each container goes in a truck that is allowed to carry it?

(b) In this different version of the problem, any canister can be placed in any truck; however, there are certain pairs of canisters that cannot be placed together in the same truck. (The chemicals they contain may react explosively if brought into contact.) Is there a way to load all n canisters into the m trucks so that no truck is overloaded, and no two canisters are placed in the same truck when they are not supposed to be?

***S*olution(s):**

Part (a):

This problem asks us if there is any way to load all n canisters into m trucks and check that no truck is overloaded and all containers goes in any truck that is allowed to carry it. There is no other nuances that do not allow certain packages to be stored together so they will all be treated equally. This sounds like a problem that could be solved using a Network Flow approach.

Lets represent all the factors in this problem within a graph $G = (V,E)$. Now we need to assign two subsets of nodes within the graph. The first subset C will contain all the canisters that need to be transported $(c_1 ... c_n)$. The second subset R will represent trucks $(r_1 ... r_m)$. Edges will be generated between nodes within subsets C and R which is represented by (c_n, r_m). First step of a network flow problem is to assign the "super-source" s and "super-sink" t. The source node s will be connected by an edge between each node in the C subset and the sink node t will have a edge connecting from each node withing R.

Now that we have the lets find out the capacity for each edge. Edges generated between the the super source s and the nodes in canister subset C will have a capacity of 1. Other edges that will also have a capacity of 1 will be the edges that connect nodes from the canister subset C and the truck subset R nodes. The only edges that will not have a capacity of 1 is the edges connecting the nodes form the truck subset R to the super sink t. Capacity of these edges will be k which is given to us as the total number of canisters each truck can hold, which is also why I decided to set the capacity of these edges here. The capacity of k here will ensure that each truck will not be overloaded with canisters. We can assume that if there is a plausible s-t flow in the graph, then this would solve the problem asked in this part of the question. An example graph will be shown in Figure 1 below. The algorithm will follow all the steps that are taken within a max-flow network flow approach and sending on unit of flow to place canisters in trucks.

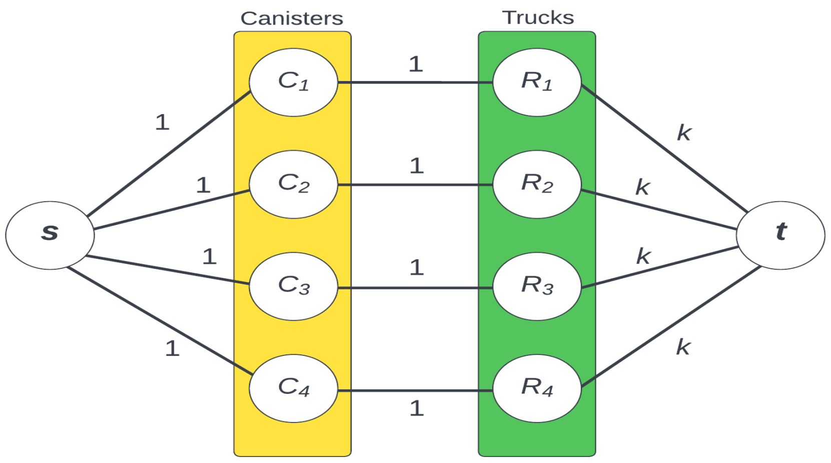

Figure 1: Shows an example of what a network within the canister-truck problem given to us. This example consists of only 4 nodes. This example is to mainly demonstrate how the overall graph should look similar to and what the capacities of each edge will be.

Since we know that a network flow is how we solve this problem, we can say that this problem is solvable in polynomial time. In fact, for max network flow approaches, we can find the run time for the algorithm which is indicated by the number of edges and vertices. For this case we can say the v is the number of vertices and e are the number of edges in G with a run time of *O(v*e)*. From that, I conclude this problem is *not* NP-complete.

Part (b):

In this version of the problem, we add more nuance to the situations by making sure some canisters are not placed with other canisters due the hazardous nature of some of the canisters being set off when in the proximity of other canisters. Right away, I cannot think of any plausible way to find a polynomial-time algorithm that will account for the pairs of canisters so I am going to conclude that this is not done by network flow or any other polynomial time algorithm, but instead is an NP-complete problem. I will take the same steps listed in previous problem to prove so.

Step 1, here we need to set that this "Canister Problem" is in NP. We know that we need to sort these canisters in some way, so the sorting will part of this approach as well as the network flow part of this problem. Both sorting algorithms and network flow algorithms can be solved in polynomial time. Overall, the approach to this algorithm will include a graph $G = (V,E)$ and the given k which will be the maximum amount of canisters that can fit in a truck. Sorting must be done in this approach which we can say is similar to k-coloring. One can verify in polynomial time that at most k colors are used, and that no pair of nodes joined by an edge receive the same color. I now conclude that the "Canister Problem" is in NP.

Step 2, to pick a similar known problem I scoured through chapter 8 of Keinberg and Tardos' Algorithm Design and found a problem called the Graph Coloring problem. Graph coloring and k-Coloring problems are used to determine whether a graph G is bipartite. The 2-Coloring problem is denoted by the question, "Can one color the nodes of G red and blue so that every edge has one red end and one blue end?" These 2-Coloring problem are solvable in polynomial time. Extending this question to 3-Coloring proved to be a very complex and hard problem that is NP-complete (8.22). Since this problem is similar to what we are trying to achieve in the "Canister Problem", I will choose the 3-Coloring problem as a the known problem to prove that the "Canister Problem" is also NP-complete.

Step 3, this final step is prove that 3-Coloring $\leq_P$ "Canister Problem". The use of an arbitrary instance of the Vertex Cover problem $s_{\textit{3-Coloring}}$ and an instance of the Hitting Set problem $s_{\textit{Canister Problem}}$ will be used to prove that 3-Coloring $\leq_P$ "Canister Problem".

Let's start off with $s_{\textit{3-Coloring}}$ and an equivalent instance of "Canister Problem" $s_{\textit{"Canister Problem"}}$. We are given a graph $G = (V,E)$, a numerical value k. Let me now define all the nodes as v_i where i has the instance of a 1 of 3 colors in the arbitrary instance of $s_{\textit{3-Coloring}}$. Similarly, in the $s_{\textit{Canister Problem}}$ instance, we define each node as a canister and instead of colors, we can say that we have 1 of 3 trucks that each canister could be in. Since in the 3-Coloring problem it is stated that two neighbors are never the same color, we can now say the same for canisters within the trucks. Canisters will not be able to be place by a hazardous pair with the 3-Coloring problem approach. This now somewhat can "sort" the canisters if we take all the same colored/truck nodes within the instance of $s_{\textit{Canister Problem}}$. Similarly in an the $s_{\textit{3-Coloring}}$ instance we can see that it also will "sort" these colored nodes into 3 categories. The instance $s_{\textit{3-Coloring}}$ is a "yes" instance of 3-Coloring and $s_{\textit{Canister Problem}}$ is a "yes" instance of the "Canister Problem" which indicates that both instances have the same answer. This can help use conclude that both $s_{\textit{3-Coloring}}$ and $s_{\textit{Canister Problem}}$ can be constructed in polynomial time. This also indicates that 3-Coloring $\leq_{\mathrm{P}}$ "Canister Problem".

All three steps above prove that the "Canister problem" ***is NP-complete.***