Introduction

The point of this assignment is to expose you to the vast number of NP complete problems—most of which are very practical problems and occur frequently in industry—as well as give you deep experience with one of them. In this assignment, you will be given an NP complete problem from the list below and your task is to research the problem, understand why it is NP complete, and create a video presentation giving the details of the problem, why it is important, and the details of a reduction that demonstrates that it is NP complete. Video presentations must be 5–10 minutes in length

This assignment is a **group assignment** in which you work in pairs and partners will be assigned based off of similar problem interest and availability. You must fill out the Assignment 9 Questionnaire by **8:00 AM CDT on Thursday, April 30th** listing the problems you are interested in and your availability. If you cannot access the form, be sure to log into Google using your Drake credentials first. Before class on Thursday, groups and problems will be announced to you via an email.

Integrated into Slack is a **video calling feature** using Microsoft Teams that allows you to make calls to your partner, share screens, and prepare the presentation together. I recommend you use this to coordinate work on the assignment. There are various methods for recording a video presentation together, including Microsoft Teams, Zoom, and Blackboard Collaborate. The video presentation submission deadline is **11:59 PM CDT on Thursday, May 7th** on the last day of class.

All videos will be made available to the entire class via Blackboard, and an extra credit opportunity will be available for watching the videos and filling out a short rubric evaluating the presentations.

NP Complete Problems

Traveling Salesman. This problem is about finding cheap paths for a traveling salesman to visit every city in a graph. It demonstrates how difficult optimal package shipping problems are.

More formally, given a fully connected directed graph $G = (V, E)$ and a vertex $s \in V$, an *s-tour* of G is a path that starts and ends at s and visits every vertex at least once. Given a *cost function* $c : E \to \mathbb{Q}^+$ that assigns a

positive rational[1] cost to each edge in the graph G, we define the *total c-cost* of an s-tour to be $\sum_{e \in T} c(e)$ where $T \subseteq E$ is the set of edges used in the s-tour. The *traveling salesman problem* is then the language:

$$\text{TSP} = \{\langle G, c, s, d\rangle \mid G \text{ has an } s\text{-tour with total } c\text{-cost at most d}\}.$$

Longest Path. This problem concerns finding *long* paths from a vertex x to a vertex y in a graph. It is the natural opposite to the *shortest* path problem which can be efficiently solved. It is definitely unintuitive why finding long paths is so challenging but finding short paths is so easy.

More formally, we define a *simple path from x to y* in an undirected graph $G = (V, E)$ is a sequence of adjacent vertices that begins with x, ends with y, and which no intermediate vertex appears twice. The *longest path problem* is then the language:

$$\text{PATH}_{\text{LONG}} = \{\langle G, x, y, k\rangle \mid G \text{ contains a simple path from } x \text{ to } y \text{ of length at least } k\}.$$

Set Cover. This problem encapsulates the problem of choosing a list of items that satisfy a set of requirements. It is similar to choosing a list of classes at Drake that satisfy all the *area of interest* requirements for graduation.

More formally, let U be a finite set called a *universe set*, and let $\mathcal{S} = \{S_1, \ldots, S_n\}$ be a finite collection of sets such that $S_i \subseteq U$ for each $1 \leq i \leq n$. We call $\mathcal{S}$ a *family* of subsets over universe U. We say that a sub-family $\mathcal{C} \subseteq \mathcal{S}$ *covers* the set U if

$$U = \bigcup_{S_i \in \mathcal{C}} S_i.$$

In other words, every element of U is contained in one of the sets S_i in the sub-family $\mathcal{C}$. For example, if $U = \{1, 2, 3, 4\}$ and $\mathcal{S} = \{\{1, 2\}, \{3\}, \{4\}, \{3, 4\}\}$, then the set $\mathcal{C} = \{\{1, 2\}, \{3, 4\}\}$ covers U since the union of its subsets is the set U. However, if $\mathcal{C} = \{\{3, 4\}, \{3\}, \{4\}\}$, it does *not* cover U since the elements 1 and 2 are not included in any subset of $\mathcal{C}$.

The *set cover problem* is then the language:

$$\text{SET-COVER} = \{\langle U, \mathcal{S}, k\rangle \mid \mathcal{S} \text{ has a sub-family that covers } U \text{ of size } k\}.$$

Hitting Set. This problem encapsulates the problem of choosing a list of items that each "hit" a category of some kind. For example, if your major

[1] We use $\mathbb{Q}^+$ here instead of $\mathbb{R}^+$ to avoid discussing finite string encodings of reals.

requires you to take a class from each of the categories A, B, and C, then you must choose a set of classes to take such that you take at least one class from each category. The interesting aspect of this problem, is that some classes might exist in multiple categories.

More formally, let U be a finite universe set, and let $\mathcal{S}$ be a finite family of subsets over U. We say that a set $H \subseteq U$ is a *hitting set* if every set $S_i \in \mathcal{S}$ contains at least one element of H. For example, if $U = \{1, 2, 3, 4\}$ and $\mathcal{S} = \{\{1\}, \{1, 2\}, \{3\}, \{3, 4\}\}$, then the set $H = \{1, 3\}$ is a hitting set.

The *hitting set problem* is then the language:

$$\text{HITTING-SET} = \{\langle U, \mathcal{S}, k\rangle \mid U \text{ has a hitting set of size } k\}.$$

Set Packing. This problem is concerned with the optimal allocation of resources. For example, suppose you have a set of ingredients in your kitchen and a set of recipes that use these ingredients. If you want to cook as many recipe dishes as possible with what you have, you must decide how to allocate your ingredients to recipes in the most efficient way possible.

More formally, let U be a finite universe set, and let $\mathcal{S}$ be a finite family of subsets over U. We say that a sub-family $\mathcal{C} \subseteq \mathcal{S}$ is *pairwise-disjoint* if for each $X, Y \in \mathcal{C}$ such that $X \neq Y$, then $X \cap Y = \emptyset$.

The *set packing problem* is then the language:

$$\text{SET-PACKING} = \{\langle U, \mathcal{S}, k\rangle \mid \mathcal{S} \text{ has a pairwise-disjoint sub-family of size } k\}.$$

Partition. This problem is concerned with dividing up valuable items into two sets that have equal value.

More formally, let S be a finite sequence of numbers (note that numbers may appear multiple times). We say that such an S *can be equally partitioned* if there is a partition of S into two parts S_1 and S_2 such that $\sum_{x \in S_1} x = \sum_{y \in S_2} y$. For example, the sequence $S = (3, 1, 1, 2, 2, 1)$ can be partitioned into $S_1 = (1, 1, 1, 2)$ and $S_2 = (2, 3)$, so it can be equally partitioned.

The *partition problem* is then the language:

$$\text{PARTITION} = \{\langle S\rangle \mid S \text{ can be equally partitioned}\}.$$

Knapsack. Earlier this year, a clerk at Lucky Candy Deli in New York went viral for giving customers five seconds to grab anything they want in the store for free if they could answer a simple math question. (It went viral on Tik Tok.) The problem of packing valuable items in a bag with finite capacity is exactly the knapsack problem.

More formally, suppose that there is a set of items

$$I = \{(v_1, w_1), (v_2, w_2), \ldots, (v_n, w_n)\}$$

where each item has value $v_i \in \mathbb{N}$ and weight $w_i \in \mathbb{N}$. Given a subset of the items $C \subseteq I$, define the *total value* of C to be the sum $v_C = \sum_{(v_i,w_i)\in C} v_i$ and the *total weight* of C to be the sum $w_C = \sum_{(v_i,w_i)\in C} w_i$.

The *knapsack problem* is then the language:

$$\text{KS} = \{\langle I, V, W\rangle \mid \text{there is a subset } C \subseteq I \text{ with } w_C \leq W \text{ and } v_C \geq V\}.$$

In the above language, $W \in \mathbb{N}$ is called the *capacity* of the knapsack and $V \in \mathbb{N}$ is the total value desired.

Bagging. At retail stores, employees try to bag items using the fewest number of bags possible. Suppose that a customer purchases a list of items I and each item has positive integer weight $w_i \in \mathbb{N}$. For $n, c \in \mathbb{N}$, we say that I can be (c, n)-*bagged* if the items can be partitioned into at most n bags and the total weight of the items in each bag does not exceed c.

We can formulate the grocery bagging problem as a language.

$$\text{BAG} = \{\langle I, n, c\rangle \mid I = (w_1, w_2, \ldots, w_k) \text{ is a list of items that can be } (c, n)\text{-bagged}\}.$$

Graph Coloring. A *coloring* of an undirected graph is an assignment of colors to its nodes so that no two adjacent nodes are assigned the same color. The *three coloring problem* is then the language:

$$\text{3-COLOR} = \{\langle G\rangle \mid G \text{ is colorable with 3 colors}\}.$$

Dominating Set. A subset $D \subseteq V$ of the vertices of an undirected graph $G = (V, E)$ is called a *dominating set* if every other vertex in $V \setminus D$ is adjacent to some node in D. The *dominating set problem* is the language:

$$\text{DOMINATING-SET} = \{\langle G, k\rangle \mid G \text{ has a dominating set with } k \text{ nodes}\}.$$