CS561 HW13

Ryan Scherbarth

November 2024

1. **Prove by induction that any graph with maximum degree 3 can be colored with at most 4 colors. Recall that a *coloring* of a graph G is an assignment of a color to each node in G s.t. the endpoints of each edge in G are assigned different colors. Don't forget to include BC, IH, and IS in your proof**
   Hint: Perform induction on n, the number of node sin G. In the IS, think about how to make G smaller, so that you can use the IH.

   Base case: $n = 1$
   An undirected graph with $n = 1$ nodes has 0 edges, so it is trivially true that we can properly color our graph in this case.

   Inductive Hypothesis:
   Assume that for any graph G' s.t. $j < n$ nodes with a maximum degree of at most 3 can be colored using at most 4 colors.

   Inductive Step:
   Start by selecting a node v in G, and removing it, and all connected edges, to create a new sub graph G'. Since G' now has $n - 1$ nodes, we can induce the IH and conclude that G' can be colored by at most 4 colors.

   Next, we re-add v to G'. When v was removed, being at most degree 3 we know it could have connected at most 3 neighbors. Therefore, when we re-add we will be re-introducing at most 3 edges. Since we can give each of these 3 edges a unique color given 4 possible colors, we know we can satisfy the color-ability requirements. Therefore, via proof by induction, we can color any graph of degree at most 3 with at most 4 colors.

2. **Exercise 34.5-2 (0-1 Integer Programming)**
Given an integer $m \times n$ matrix A and an integer m-vector b, the 0-1 integer programming problem asks whether there exists an integer n-vector x with elements in the set $\{0, 1\}$ such that $Ax \leq b$. Prove that 0-1 integer programming is NP-Complete. (Hint: Reduce from 3-CNF-SAT).

We can prove this problem is NP-Complete using reduction from the 3-SAT problem.

NP-Easy
Our first step is proving that if we have a solution, we can prove that it is a valid solution in NP time. The solution to the 0-1 integer problem we need to verify that $Ax \leq b$, which takes $O(mn) + O(m)$ time. Since both steps are polynomial, the problem is in NP.

NP-Hard
The 3-SAT problem takes a boolean formula F containing n variables, and m clauses, each with 3 literals. It then returns a Boolean value.

For each variable x_j of F, where x_j is one of the n variables, we will give a corresponding variable for the 0-1 Integer problem, $y_j \in \{0, 1\}$. For each clause c_i, we represent as a summation of each of the x values.

The constraints $Ax \leq b$ are s.t. A corresponds to $a_{i,j}$ and b corresponds to b_i, and each row A_i corresponds to clause c_i.

Therefore, since we are able to reduce 3-SAT to the o-1 integer programming problem by transforming F into A and b, we can conclude that the question is NP-Hard.

NP-Complete
Since we can show that the problem is both NP-Easy (we can prove the reduction moving rightward) and is NP-Hard (we can also prove the correlation moving backward), we can conclude that the problem is NP-Complete.

3. **In the IGOR problem, Dr. Frankenstein tasks Igor to collect various body parts. Each grave contains an assortment of items: body parts and rubbish. Each item has a corresponding weight. Due to the circumstances of the midnight task, Igor must choose a set of graves to dig up before dawn, and in the darkness, he will have no time to sort through the remains. He must collect ALL buried items from each grave he digs up and place them on his corpse wagon, which has a maximum weight capacity.**
The input for the IGOR problems is a list, R, (duplicates allowed) of required body parts; a list of graves $G = [g_1, g_2, \ldots, g_n]$, each of which is a list of buried items. Each item $i \in \cup_{j=1}^{n} g_j$ has a weight defined by $w(i) = w_i$, s.t. $w_i > 0$. Finally, the wagon capacity is denoted by K where $K > 0$. The output is either $TRUE$ or $FALSE$.
For Example;
R = **[skull, torso, brain*, brain*, brain*]**
G = [g_1 = **[skull, brain]**, g_2 = **[skull, torso, brain, brain]**, g_3 = **[torso, pocketwatch]**]
$w(skull) = 2$, $w(torso) = 5$, $w(brain) = 1$, $w(pocketwatch) = 0.2$
$K = 15$
The example input returns $TRUE$ since Igor may choose graves g_1 and g_2. Igor places 6 items totaling 12 weight $\leq K = 15$ into the wagon, fulfilling the requisition R. Prove that IGOR is NP-Hard.
***Dr. Frankenstein understand the importance of finding a good brain.**

The vertex cover problem takes a graph G and integer k as input, and determines if there exists a subset of vertices $V' \subseteq V$ s.t. $|V'| \leq k$ with every edge E incident to at least one vertex in V'.

To map the Igor problem onto the Vertex cover problem. For each grave, a vertex $v_i \in V$ maps to a grave g_i. Items in a grave correspond to edges of a vertex, and weights of items correspond to weights of the edges on a weighted graph V. We can assume all these weight functions for V are all 0 if it is an undirected graph. The wagon's capacity corresponds to the integer k.

Therefore, since we can directly map each operation from the Igor problem onto the vertex cover problem, we can conclude that the problem is NP-Hard.

4. **Rock, Paper, Scissors is a simple 2 person game. In a given round, both players simultaneously choose either Rock, Paper, or Scissors. If they both choose the same object, it's a tie. Otherwise, Rock beats Scissors; Scissors beats Paper; and Paper beats Rock. Imagine you're playing the following betting variant of this game with a friend. When Scissors beats Paper, or Paper beats Rock, the loser gives the winner \$1. However, in the case when Rock beats Scissors, this is called a smash, and the loser must give the winner \$10.**

   (a) **Say you know that your friend will choose Rock, Scissors, or Paper, each with probability $\frac{1}{3}$. Write a linear program to calculate the probabilities you should use to maximize your expected winnings. Let p_1, p_2, p_3 be the variables associated with your optimal probabilities for choosing Rock, Scissors, and Paper respectively. Note: If you want to check your work, there are several free linear program solvers on the internet: check Wikipedia page on linear programming.**

   Let p_r, p_s, p_p represent the probabilities of choosing rock, scissors, and paper accordingly. We can create a table that shows each of the possible earnings for each case;

	Rock	Paper	Scissors
Rock	0	1	-10
Paper	-1	0	1
Scissors	10	-1	0

   From here, we can form our equation for expected values. The expected value formula is given by $\sum p(x_i) * p_i$. We can form each of our cases where the expected value for the case where I choose rock is the summation of the **row** labeled rock. Our final equation to maximize will be

$$w = \frac{(p_s - p_r) + (10p_r - p_p) + (p_p - 1 - p_s)}{3}$$
$$w = 3p_r - 3p_s$$

   We will also introduce our constraint functions. The only constraints we need to add are those to make sure we satisfy the properties of probabilities;

$$0 \leq p_r,\, p_s,\, p_p \leq 1$$
$$p_r + p_s + p_p = 1$$

   Plugging into a linear algebra solver, we find the optimal values to be

$$p_r = 1$$
$$p_s = 0$$
$$p_p = 0$$

   Or in other words, we maximize our values by playing only rock, and we still satisfy all of our constraint functions.

(b) **Now say that your friend is smart and, also, semi-clairvoyant: she magically knows the exact probabilities you are using and will respond optimally. Write another linear program to calculate the probabilities you should use in order to maximize your expected winnings. Hint 1: If your opponent knows your strategy, her strategy will be to choose one of the three objects with probability 1. Hint 2: Review the LP we wrote for the shortest path problem.**

Just as we found in the last part, we will have the weight function as

$$w = (p_s - p_r) + (10p_r - p_p) + (p_p - 1 - p_s)$$

Since we can't assume the probabilities are equally likely at $\frac{1}{3}$ as we did last time, we will now have to add extra constraints in order to solve for those probabilities. We can do so by determining the probabilities for each case as

$$\begin{aligned} w &\leq p_s - p_r \\ w &\leq 10p_r - p_p \\ w &\leq p_p - 10p_s \end{aligned}$$

We introduce these constraints in addition to our previous constraints of

$$\begin{aligned} 0 \leq p_r,\, p_s,\, p_p \leq 1 \\ p_r + p_s + p_p = 1 \end{aligned}$$

and we find our final probabilities are

$$\begin{aligned} p_r &\approx 0\% \\ p_s &\approx 17\% \\ p_p &\approx 83\% \end{aligned}$$

Assigning these probabilities to how often we choose each of the different plays will help us maximize the score we achieve regardless of our opponent knowing our strategy. This is because we are essentially finding that when op knows exactly what our strategy, we need to shift to minimizing our losses, which is what we get when we play paper and scissors.

5. **X-STREAM Dance Dance Revolution (XDDR) is played on a Pogo Stick while blindfolded. At the beginning of each round, you can hop from your current square to any square. However, the target sequence, σ is not known in advance: the value $\sigma[i]$ is announced only at the end of round i, for each $i \in [1, n]$, where n is the length of σ.**
Your goal is to minimize cost: the number of rounds $i \in [1, n]$ in which you hope in a square different than $\sigma[i]$.
Below is an example game; your cost is 5 because there are 5 rounds where the square you hop in does not match the target square in σ.

σ	A	C	A	D	C	D	D
Pogo position	A	B	B	D	A	C	A
Cost?	0	1	1	0	1	1	1

In round i, you let $\overrightarrow{x_i}$ be a length 4 vector giving a probability distribution over the 4 possible squares on which you will hop, i.e. $\overrightarrow{x_i}[1]$, $\overrightarrow{x_i}[2]$, $\overrightarrow{x_i}[3]$, $\overrightarrow{x_i}[4]$ are probabilities of hopping into squares A, B, C, D respectively.
Then, for round i, you define $f_i(x_i)$ as your expected cost in round i. In round i, let c_i be a length 4 vector giving the cost outcome: $c_i[j] = 0$ when j matches the square given by $\sigma[i]$ and $c_i[j] = 1$ otherwise. For example, in round i if $x_i = [\frac{1}{8}, \frac{1}{2}, \frac{1}{8}, \frac{1}{4}]$ and $c_i = [1, 1, 1, 0]$ your expected cost for this round is $\frac{3}{4}$.

(a) **Give the mathematical expression for f_i as a function of $\overrightarrow{x_i}$ and $\overrightarrow{c_i}$.**

f_i will be given as

$$f_i(\overrightarrow{x_i}, \overrightarrow{c_i}) = \sum_{j=1}^{4} \overrightarrow{x_i}[j] \cdot \overrightarrow{c_i}$$

Where $\overrightarrow{x_i}$ gives the probability, on round i, that we jump to position j. $\overrightarrow{c_i}$ is the cost on this round that we jump to j.

(b) **Describe, algebraically, the convex search space k. What is the diameter, D, of k?**

The convex search space, k, is found by the set of all valid probability distributions,

$$k = \{\overrightarrow{x_i} \in R^4 | \sum_{j=1}^{4} \overrightarrow{x_i}[j] \geq 0 \forall j\}$$

So k is simply all values that are valid in the problem. We can find the diameter, D, as the max distance between any two points in k. The two extreme points we have will be the first case, $(1, 0, 0, 0)$ and the final case $(0, 0, 0, 1)$. Calculating the distance

$$\begin{aligned} D &= \sqrt{(1-0)^2 + 0^2 + 0^2 + (0-1)^2} \\ &= \sqrt{2} \end{aligned}$$

(c) **Zinkevich's theorem says that the cost of online gradient descent tracks the cost of the best offline solution, $x*$. In particular, if OPT is the cost of the best offline solution, then the cost of our algorithm is at most $OPT + \sqrt{n}DG$. Give a precise 1 line definition of OPT for this problem using the c_i values.**

We can find opt by just saying we define a function that chooses the optimal option on each iteration;

$$OPT = \min_{\overrightarrow{x_i} \in k} \sum_{j=1}^{4} \overrightarrow{x_i} \cdot \overrightarrow{c_i}$$

(d) **Now, you want to use some history. In particular, you notice that the current square in σ often depends on the last square. So you want to use the outcome of the last round to help set your probability distribution for the current round. To do this, how would you change the convex search space k? How many dimensions does it now have? Will OPT, G, and D likely increase or decrease? Will your algorithm's expected cost increase or decrease?**

To achieve this we would need to change x s.t. instead of just saving the iteration, i, it also saves all the previous iterations up to that point, represented together as $\overrightarrow{x_{i,j}}$. With the addition of j as the square from the previous round, each variable now holds 4 additional variables as j.

OPT will decrease, since we're not able to make better guesses each time based off of the previous guesses.

G increases because we now have larger changes that can be calculated taking into account both the current step and the previous. This is not necessarily a bad thing for G to increase or decrease in a vacuum.

D will increase substantially. For each item we store 4x items, so we're functionally increasing the search space k from 4 originally to 16 now since each of the previous 4 squares we save have 4 possibilities.