Note: Your TA probably will not cover all the problems. This is totally fine, the discussion worksheets are not designed to be finished in an hour. They are deliberately made long so they can serve as a resource you can use to practice, reinforce, and build upon concepts discussed in lecture, readings, and the homework.

1 Breaking Encryption

After years of research, Horizon Wireless released an encryption algorithm E that encrypts an n-bit message x in time $O(n^2)$. Show that if $\mathsf{P} = \mathsf{NP}$ then this encryption algorithm can be broken in polynomial time. More precisely, argue that if $\mathsf{P} = \mathsf{NP}$, then the following decryption problem can be solved in polynomial time.

DECRYPT :

Input: An encrypted message m_e (encrypted using the algorithm E on an unkown input)

Output: Decryption x of the message m_e, i.e an n bit string x such that encrypting x produces m_e.

Solution: First observe that DECRYPT $\in \mathsf{NP}$. This is because given a decryption x of an encrypted message m_e, we can run $E(x)$ and check if it outputs m_e. Since $E(x)$ has a $O(N^2)$ algorithm, we can do this check in polynomial time. Hence, DECRYPT is an NP problem.

Now if $\mathsf{P} = \mathsf{NP}$, DECRYPT will also be in P. This means DECRYPT can be solved in polynomial time.

2 Public Funds

You are looking to build a new fence for your mansion, to keep out pesky people protesting profligate purchases. You have m bank accounts at your disposal to use to pay for your fence; each account i has a balance of b_i. You must choose one of n options for your fence; each fence j costs c_j dollars. You would like to withdraw from at most k of the bank accounts to build the fence, and due to peculiar UC accounting rules, if you use a particular bank account, you must use the whole balance (all b_m dollars.)

Determine whether it is possible to exactly pay for some fence j; that is, whether there is a j between 1 and n such that you can withdraw exactly c_j dollars given the bank account balances $b_1, \ldots, b_m$, the fence costs $c_1, \ldots, c_n$, and k.

Your task is to prove that Public Funds is **NP**-complete.

(a) Prove that Public Funds is in **NP**.

(b) Prove that Public Funds is **NP**-hard by providing a reduction from Subset Sum.

Note: to rigorously prove the correctness of a reduction from A to B, you must show two things:

1. *If an instance of A has a solution, then the transformed instance of B has a solution.*
2. *If an instance of B in the format of the transformation has a solution, then the corresponding instance of A has a solution.*

Solution:

Proof that Public Funds is in NP.
We can verify a solution for Public Funds in polynomial time by verifying that the sum of the balances of the chosen bank accounts adds up exactly to the price of the fence.

Proof that Public Funds is in NP-Hard.
To show that it is in NP-Hard, we reduce from SUBSET SUM.

Let each of the m elements in the set of numbers correspond to a bank account's balance b_m. Let the desired sum s correspond to the price of the only fence c_1. Also, set the maximum number of bank accounts used, k, to the number of elements in the set of numbers m.

1. **If an instance of subset sum has a solution, then the transformed instance of Public Funds has a solution.**
If subset sum has a solution, then we can take the banks that correspond to the elements of the solution, and those banks should sum up to the only fence, thus Public Funds should have a solution.

2. **If an instance of Public Funds in the format of the transformation has a solution, then the corresponding instance of subset sum has a solution.**
If an algorithm for Public Funds finds a solution, we can interpret the bank accounts chosen as a solution for SUBSET SUM.

3 Approximating the Traveling Salesperson Problem

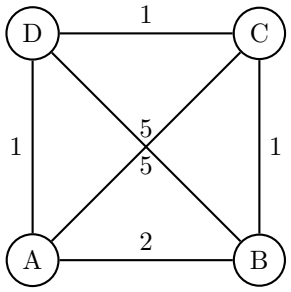
Recall in lecture, we learned the following approximation algorithm for TSP:

1. Given the complete graph $G = (V, E)$, compute its MST.
2. Run DFS on the MST computed in the previous step, and record down the the vertices visited in pre-order.

3. The output tour is the list of vertices computed in the previous step, with the first vertex appended to the end.

When G satisfies the triangle inequality, this algorithm achieves an approximation factor of 2. But what happens when triangle inequality does not hold?

Suppose we run this approximation algorithm on the following graph:



The algorithm will return different tours based on the choices it makes during its depth first traversal.

1. Which DFS traversal leads to the best possible output tour?

2. Which DFS traversal leads to the worst possible output tour?

3. What is the approximation ratio given by the algorithm in the worst case for the above instance? Why is it worse than 2?

Solution:

1. **A-D-C-B**, **D-C-B-A**, **B-C-D-A**, or **C-D-A-B**
   Explanation:
   The MST is {(A,D),(D,C),(C,B)}. The optimal tour uses all of these edges.
   For example, if we start traversing the tree at A, then the only traversal would be to follow the tree and go through the vertices in the order D, C, B
   Another potential traversal would be to start at D, move to C, then B, and backtrack, before going to A.
   Notice that if we started at D and moved to A first, then we would have to visit C next, and this would not lead to the best possible output tour (as per part (b)).
2. **C-B-D-A** or **D-A-C-B**
   See above explanation.
   Notice that the worst possible tour overall, **D-C-A-B**, is not possible to be output by this algorithm, regardless of the DFS traversal used.
3. $\frac{12}{5}$
   From previous parts, the optimal tour length is 5 while the worst possible is 12. This is worse than 2 because our graph does not satisfy the triangle inequality, specifically $\ell(A, C) > \ell(A, B) + \ell(B, C)$ and $\ell(B, D) > \ell(A, B) + \ell(D, A)$, which is the necessary condition for our algorithm to guarantee an approximation ratio of 2.

4 Boba Shops

A rectangular city is divided into a grid of $m \times n$ blocks. You would like to set up boba shops so that for every block in the city, either there is a boba shop within the block or there is one in a neighboring block (assume there are up to 4 neighboring blocks for every block). It costs r_{ij} to rent space for a boba shop in block ij.

Write an integer linear program to determine on which blocks to set up the boba shops, so as to minimize the total rental costs.

(a) What are your variables, and what do they mean?

(b) What is the objective function? Briefly justify.

(c) What are the constraints? Briefly justify.

(d) Solving the non-integer version of the linear program yields a real-valued solution. How would you round the LP solution to obtain an integer solution to the problem? Describe the algorithm in at most two sentences.

(e) What is the approximation ratio of your algorithm in part (d)? Briefly justify.

Solution:

(a) There is a variable for every block x_{ij}, i.e., $\{x_{ij} \mid i \in \{1, \ldots, m\}, j \in \{1, \ldots, n\}\}$. This variable corresponds to whether we put a boba shop at that block or not.

(b) $\min \sum_{i=1}^{m} \sum_{j=1}^{n} r_{ij} x_{ij}$. Alternatively, $\min t$ is correct as well as long as the correct constraint is added.

(c) (i) $x_{ij} \geq 0$: This constraint just corresponds to saying that there either is or isn't a boba shop at any block. $x_{ij} \in \{0, 1\}$ or $x_{ij} \in \mathbb{Z}_+$ is also correct.

(ii) For every $1 \leq i \leq m, 1 \leq j \leq n$:

$$x_{ij} + x_{(i+1),j} \mathbb{1}_{\{i+1 \leq m\}} + x_{(i-1),j} \mathbb{1}_{\{i-1 \geq 1\}} + x_{i,(j+1)} \mathbb{1}_{\{j+1 \leq n\}} + x_{i,(j-1)} \mathbb{1}_{\{j-1 \geq 1\}} \geq 1$$

This constraint corresponds to that for every block, there needs to be a boba shop at that block or a neighboring block.

$\mathbb{1}_{\{i+1 \leq m\}}$ means "1 if $\{i + 1 \leq m\}$, and 0 otherwise". It keeps track of the fact that we may not have all 4 neighbors on the edges, for instance.

(iii) If the objective was $\min t$, then the constraint $\sum_{i=1}^{m} \sum_{j=1}^{n} r_{ij} x_{ij} \leq t$ needs to be added.

(d) Round to 1 all variables which are greater than or equal to 1/5. Otherwise, round to 0.

In other words, put a boba shop on (i, j) iff $x_{i,j} \geq 0.2$.

(e) Using the rounding scheme in the previous part gives a 5-approximation.

Notice that every constraint has at most 5 variables. So for every constraint, there exists at least one variable in the constraint which has value $\geq 1/5$ (not everyone is below average). The total cost of the rounded solution is at most $5 \cdot$ LP-OPT, since $r_{ij} \leq 5 r_{ij} x_{ij}$ for any x_{ij} that gets rounded up, and the other i, j pairs contribute nothing to the cost of the rounded solution. Since Integral-OPT $\geq$ LP-OPT (the LP is more general than the ILP), our rounding gives value at most 5 LP-OPT $\leq$ 5 Integral-OPT. So we get a 5-approximation.

5 $\sqrt{n}$ coloring

(a) Let G be a graph of maximum degree δ. Show that G is $(\delta + 1)$-colorable.

(b) Suppose $G = (V, E)$ is a 3-colorable graph. Let v be any vertex in G. Show that the graph induced on the neighborhood of v is 2-colorable.

Note: the graph induced on the neighborhood of v refers to the following subgraph:

$$G' = (V' = \text{neighbors of } v, E' = \text{all edges in } E \text{ with both endpoints in } V').$$

(c) Give a polynomial time algorithm that takes in a 3-colorable n-vertex graph G as input and outputs a valid coloring of its vertices using $O(\sqrt{n})$ colors. Prove that your algorithm is correct.

Hint: think of an algorithm that first assigns colors to "high-degree" vertices and their neighborhoods, and then assigns colors to the rest of the graph. The previous two parts might be useful.

Solution:

(a) Let the color palette be $[\delta + 1]$. While there is a vertex with an unassigned color, give it a color that is distinct from the color assigned to any of its neighbors (such a color always exists since we have a palette of size $\delta + 1$ but only at most δ neighbors).

(b) In an induced graph, we only care about the direct vertices adjacent to v and v itself.
In any 3-coloring of G, v must have a different color from its neighborhood and therefore the neighborhood must be 2-colorable.

(c) While there is a vertex of degree $\geq \sqrt{n}$, choose any such vertex v, pick 3 colors, use one color to color v, and the remaining 2 colors to color the neighborhood of v (2-coloring is an easy problem). Never use these colors ever again and delete v and its neighborhood from the graph. Since each step in the while loop deletes at least $\sqrt{n}$ vertices, there can be at most $\sqrt{n}$ iterations. This uses only $3(\sqrt{n}+1)$ colors. After this while loop is done we will be left with a graph with max degree at most $\sqrt{n}$. We can $\sqrt{n}+1$ color with fresh colors this using the greedy strategy from the solution to part a. The total number of colors used is $O(\sqrt{n})$.