Note: Your TA may not get to all the problems. This is totally fine, the discussion worksheets are not designed to be finished in an hour. The discussion worksheet is also a resource you can use to practice, reinforce, and build upon concepts discussed in lecture, readings, and the homework.

1 Optimization versus Search

Recall the following definition of the Traveling Salesman Problem, which we will call TSP. We are given a complete graph G of whose edges are weighted and a budget b. We want to find a tour (i.e., path) which passes through all the nodes of G and has length $\leq b$, if such a tour exists.
The optimization version of this problem (which we call TSP-OPT) asks directly for the shortest tour.

(a) Show that if TSP can be solved in polynomial time, then so can TSP-OPT.

(b) Do the reverse of (a), namely, show that if TSP-OPT can be solved in polynomial time, then so can TSP.

Solution:

(a) Do a binary search over all possible lengths of the optimal tour, going from 0 to the sum of all distances. Note that binary search is necessary here and we can't just increment the value of b by 1 each time since the sum of all distance is exponential in the size of the input.

(b) Run tsp-opt and return the optimal value. If it is greater than b, then no solution exists, by the definition of optimality.

2 Reliable Network

Reliable Network is the following problem: We are given two $n \times n$ matrices (a cost matrix d_{ij} and a connectivity requirement matrix r_{ij}) and also a budget b. We want to find a graph $G = (\{1, ..., n\}, E)$ such that the total cost of all edges (i.e. $\sum_{(i,j)\in E} d_{ij}$) is at most b and there are exactly r_{ij} vertex-disjoint paths between any two distinct vertices i and j.
Show that Reliable Network is NP-Complete.
Solution: Reduction from Rudrata Cycle to Reliable Network. Given $G = (V, E)$, take $b = n$, $d_{ij} = 1\ \forall(i, j) \in E$ and $d_{ij} > 1$ otherwise; set $r_{ij} = 2\ \forall(i, j)$. Now we must find edges such that the sum of the weights of all the edges is n (so only n edges), and they are part of exactly one cycle (so there are two vertex-disjoint paths between each pair of vertices). This must contain all of the vertices since each pair satisfies this. This is exactly a Rudrata Cycle.
To prove Reliable Network is NP. Given a solution of Reliable Network (i.e. a graph G). We simply run max flow with $(s, t) = (i, j)$ for any i, j to verify the r_{ij} constraints.

3 (★★★) 2-SAT and Variants

In this problem we will explore the variant of 3-SAT called 2-SAT, where each clause contains at most 2 literals (hereby called a 2-clause).

(a) Show that 2-SAT is in P. (Hint: Note that the clause $x \vee y$ is equivalent to the implications $\neg x \Rightarrow y$ and $\neg y \Rightarrow x$. If a 2-SAT formula is unsatisfiable, what must be true about the set of all such implications?)

(b) The problem of Max-2-SAT is defined as follows. Let $C_1, \ldots, C_m$ be a collection of 2-clauses and k a non-negative integer. We want to determine if there is some assignment which satisfies at least k clauses.

The problem of Max-Cut is defined as follows. Let G be an undirected unweighted graph, and k a non-negative integer. We want to determine if there is some cut with at least k edges crossing it. Max-Cut is known to be NP-complete.

Show that Max-2-SAT is NP-complete by reducing from Max-Cut. Prove the correctness of your reduction.

Solution:

(a) Let φ be a formula acting on n literals $x_1, \ldots, x_n$. Construct a graph with $2n$ vertices representing the set of literals and their negations. For each clause $(a \vee b)$ of φ add the edges $\neg a \Rightarrow b$ and $\neg b \Rightarrow a$. Use the strongly connected components algorithm and for each i, check if there is a SCC containing both x_i and $\neg x_i$. If any such component is found, report unsatisfiable. Otherwise, report satisfiable.

In the case that the algorithm reports unsatisfiable, notice that the edges of the graph are necessary implications. Thus, if some x_i and $\neg x_i$ are in the same component, there is a chain of implications which is equivalent to $x_i \to \neg x_i$ and a different chain which is equivalent to $\neg x_i \to x_i$, i.e. there is a contradiction in the set of clauses.

In the case that the algorithm reports satisfiable, there is a simple satisfying assignment: Take any sink component, and assign variables so all the literals in this component are True. Because of how we define the graph, there is a corresponding source component which has the negations of all literals in this component. Remove this source/sink component pair, and repeat the process until the graph is empty. Since we set components to true in reverse topological order, there is no implication from a true literal to a false literal. Since no literal and its negation are in the same SCC, we never try to set a variable to be both true and false. So this produces an assignment satisfying all clauses.

(Note: A common mistake is to report unsatisfiable if there is a path from x_i to $\neg x_i$ in this graph, even if there is no path from $\neg x_i$ to x_i. Even if there is a series of implications which combined give $x_i \to \neg x_i$, unless we also know $\neg x_i \to x_i$ we could set x_i to False and still possibly satisfy the clauses. For example, consider the 2-SAT formula $(\neg a \vee b) \wedge (\neg a \vee \neg b)$. These clauses are equivalent to $a \to b, b \to \neg a$, which implies $a \to \neg a$, but this 2-SAT formula is still easily satisfiable.)

(b) As suggested, we reduce unweighted `Max-Cut` to `Max-2-SAT`. Let a graph $G = (V, E)$ and an integer k be given. We want to find whether there is a cut in the graph of size k or more. We construct a boolean formula based on the graph. Every assignment of this formula will correspond to a cut in the graph.

For each vertex $u \in V$, we add a variable x_u, representing whether u is in S (true) or in $V \setminus S$ (false). For each edge $(u, v) \in E$, we add *two* clauses: $(x_u \vee x_v)$ and $(\overline{x_u} \vee \overline{x_v})$. Then, if (u, v) crosses the cut, *both* of these clauses will be satisfied (if $u \in S$ and $v \notin S$, then x_u and $\overline{x_v}$ are both true; if $u \notin S$ and $v \in S$, then $\overline{x_u}$ and x_v are both true). If (u, v) does not cross the cut, then exactly one of the clauses will be satisfied (the first if both u and v are in S, the second if neither is). Thus, any cut of size q in the graph corresponds to an assignment of values to variables that satisfies exactly $|E| + q$ clauses (1 for each of the $|E| - q$ pairs representing non-cut edges, 2 for each of the q pairs representing cut edges).

Thus, to solve `Max-Cut`(G, k), we construct this formula Φ_G and return `Max-2-SAT`$(\Phi_G, |E| + k)$. Lastly, we know that `Max-2-SAT` is in NP because given an instance and an assignment of variables, we can just iterate through the clauses and count how many are satisfied in polynomial time.