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 Decision vs. Search vs. Optimization

Recall that a vertex cover is a set of vertices in a graph such that every edge is adjacent to at least one vertex in this set.

The following are three formulations of the VERTEX COVER problem:

- As a *decision problem*: Given a graph G, return TRUE if it has a vertex cover of size at most b, and FALSE otherwise.
- As a *search problem*: Given a graph G, find a vertex cover of size at most b (that is, return the actual vertices), or report that none exists.
- As an *optimization problem*: Given a graph G, find a minimum vertex cover.

At first glance, it may seem that search should be harder than decision, and that optimization should be even harder. We will show that if any one can be solved in polynomial time, so can the others.

(a) Suppose you are handed a black box that solves VERTEX COVER (DECISION) in polynomial time. Give an algorithm that solves VERTEX COVER (SEARCH) in polynomial time.

(b) Similarly, suppose we know how to solve VERTEX COVER (SEARCH) in polynomial time. Give an algorithm that solves VERTEX COVER (OPTIMIZATION) in polynomial time.

Solution:

(a) If given a graph G and budget b, we first run the DECISION algorithm on instance (G, b). If it returns "FALSE", then report "no solution".

If it comes up "TRUE", then there is a solution and we find it as follows:

- Pick any node $v \in G$ and remove it, along with any incident edges.
- Run DECISION on the instance $(G \setminus \{v\}, b-1)$; if it says "TRUE", add v to the vertex cover. Otherwise, put v and its edges back into G.
- Repeat until G is empty.

Correctness: If there is no solution, obviously we report as such. If there is, then our algorithm tests individual nodes to see if they are in any vertex cover of size b (there may be multiple). If and only if it is, the subgraph $G \setminus \{v\}$ must have a vertex cover no larger than $b-1$. Apply this argument inductively.

Running time: We may test each vertex once before finding a v that is part of the b-vertex cover and recursing. Thus we call the DECISION procedure $O(n^2)$ times. This can be tightened to $O(n)$ by not considering any vertex twice. Since a call to DECISION costs polynomial time, we have polynomial complexity overall.

Note: this reduction can be thought of as a greedy algorithm, in which we discover (or eliminate) one vertex at a time.

(b) Binary search on the size, b, of the vertex cover.

Correctness: This algorithm is correct for the same reason as binary search.

Running time: The minimum vertex cover is certainly of size at least 1 (for a nonempty graph) and at most $|V|$, so the SEARCH black box will be called $O(\log |V|)$ times, giving polynomial complexity overall.

Finally, since solving the optimization problem allows us to answer the decision problem (think about why), we see that all three reduce to one another!

2 Vertex Cover to Set Cover

In the minimum vertex cover problem, we are given an undirected unweighted graph $G = (V, E)$ and asked to find the smallest vertex cover (defined in the previous problem).

Recall the minimum set cover problem: Given a set U of elements and a collection $S_1, \ldots, S_m$ of subsets of U, find the smallest collection of these sets whose union equals U.

Give an efficient reduction from the minimum vertex cover problem to the minimum set cover problem. **Solution:** Let $G = (V, E)$ be an instance of the minimum vertex cover problem. Create an instance of the Minimum Set Cover Problem where $U = E$ and for each $u \in V$, the set S_u contains all edges adjacent to u. Let $C = \{S_{u_1}, S_{u_2}, \ldots, S_{u_k}\}$ be a set cover. Then our corresponding vertex cover will be $u_1, u_2, \ldots, u_k$. To see this is a vertex cover, take any $(u, v) \in E$. Since $(u, v) \in U$, there is some set S_{u_i} containing (u, v), so u_i equals u or v and (u, v) is covered in the vertex cover.

Now take any vertex cover $u_1, \ldots, u_k$. To see that $S_{u_1}, \ldots, S_{u_k}$ is a set cover, take any $(u, v) \in E$. By the definition of vertex cover, there is an i such that either $u = u_i$ or $v = u_i$. So $(u, v) \in S_{u_i}$, so $S_{u_1}, \ldots, S_{u_k}$ is a set cover.

Since every vertex cover has a corresponding set cover (and vice-versa) and minimizing set cover minimizes the corresponding vertex cover, the reduction holds.

3 Exact 3-SAT

In the 3-SAT problem, we have variables x_i and clauses, where each clause is the OR of **at most** three literals (a literal is a variable or its negation). Our goal is to find an assignment of variables that satisfies all the clauses.

The exact 3-SAT problem is just like the 3-SAT problem, except each clause has **exactly** three **distinct** literals.

Give a reduction from 3-SAT to exact 3-SAT. (Hint: Note that $(x \vee y) \wedge (x \vee \neg y)$ is logically equivalent to x).

Solution: Create two dummy variables z_1, z_2. For any clause with only one literal x, replace it with the clauses $x \vee z_1 \vee z_2, x \vee z_1 \vee \neg z_2, x \vee \neg z_1 \vee z_2, x \vee \neg z_1 \vee \neg z_2$. For any clause with two literals x, y replace it with the clauses $x \vee y \vee z_1, x \vee y \vee \neg z_1$.

Note that the clauses $x \vee z_1 \vee z_2, x \vee z_1 \vee \neg z_2, x \vee \neg z_1 \vee z_2, x \vee \neg z_1 \vee \neg z_2$ are logically equivalent to x, and the clauses $x \vee y \vee z_1, x \vee y \vee \neg z_1$ are logically equivalent to $x \vee y$. So the new set of clauses is logically equivalent to the old set of clauses. This means that:

- Given a satisfying 3-SAT assignment, we can construct a satisfying assignment for the exact 3-SAT instance, by just adding the dummy variables and assigning them arbitrarily value.
- Given a satisfying exact 3-SAT assignment, we can construct a satisfying assignment for the 3-SAT instance, by just deleting the dummy variables.

4 Cycle Cover

In the cycle cover problem, we have a directed graph G, and our goal is to find a set of directed cycles $C_1, C_2, \ldots C_k$ in G such that every vertex appears in exactly one cycle (a cycle cannot revisit vertices, e.g. $a \to b \to a \to c \to a$ is not a valid cycle, but $a \to b \to c \to a$ is), or declare none exists.

In the bipartite perfect matching problem, we have a undirected bipartite graph (a graph where the vertices can be split into L, R, and there are no edges between two vertices in L or two vertices in R), and our goal is to find a set of edges in this graph such that every vertex is adjacent to exactly one edge in the set, or declare none exists.

Give a reduction from cycle cover to bipartite perfect matching. (Hint: In a cycle cover, every vertex has one incoming and one outgoing edge.)

Solution: Given the cycle cover instance G, we create a bipartite graph G' where L has one vertex v_L for every vertex in G, and R has one vertex v_R for every vertex in G. For an edge (u, v) in G, we add an edge (u_L, v_R) in the bipartite graph. We claim that G has a cycle cover if and only if G' has a perfect matching.

If G has a cycle cover, then the corresponding edges in the bipartite graph are a bipartite perfect matching: The cycle cover has exactly one edge entering each vertex so each v_R has exactly one edge adjacent to it, and the cycle cover has exactly one edge leaving each vertex, so each v_L has exactly one edge adjacent to it.

If G' has a perfect matching, then G has a cycle cover, which is formed by taking the edges in G corresponding to edges in G': If we have e.g. the edges $(a_L, b_R), (b_L, c_R), \dots, (z_L, a_R)$ in the perfect matching, we include the cycle $a \to b \to c \to \dots, z \to a$ in G in the cycle cover. Since v_L and v_R are both adjacent to some edge, every vertex will be included in the corresponding cycle cover.