Solutions to Homework Practice Problems

Practice problems:

1. [DPV] Problem 8.1 (TSP optimization versus search)

Optimization versus search. Recall the traveling salesman problem:

- `TSP`
- Input: A matrix of distances; a budget b
- Output: A tour which passes through all the cities and has length $\leq b$, if such a tour exists.

The optimization version of this problem asks directly for the shortest tour.

- `TSP-OPT`
- Input: A matrix of distances
- Output: The shortest tour which passes through all the cities.

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

Solution:

In order to show this, we will show that TSP-OPT reduces to TSP in polynomial time. The idea of this reduction is to use binary search with the b input for TSP to find the length of the shortest tour (then TSP will find the shortest tour itself). Let the distance matrix be denoted D. First, we need to find an upper bound B for the length of the shortest path. We can set B to be the sum of all distances in the distance matrix (so $B = \sum_{i,j} D_{i,j}$). Then, we run TSP with inputs D and $B/2$. If this finds a tour with length at most $B/2$, then we know the shortest tour has length in the range $[0, B/2]$. Otherwise, the shortest tour has length in the range $[B/2, B]$. Here is where we can apply binary search; for each successive iteration, run TSP on the midpoint of the remaining range for the length of the shortest path, then eliminate half of the range based on whether TSP finds a tour. Continue this recursive process until the range is narrowed to a single integer number, then return the path found by TSP on this integer. TSP will be run $O(\log B)$ times (since this is a binary search from 0 to B). The input size of TSP-OPT must be of length $O(\log B)$, since B was the sum of all the distances in D. Therefore, this reduction is polynomial, and if TSP can be solved in polynomial time, then so can TSP-OPT.

2. [DPV] Problem 8.3 (Stingy SAT)

`STINGY SAT` is the following problem: given a set of clauses (each a disjunction of literals) and an integer k, and a satisfying assignment in which at most k variables are true, if such an assignment exists. Prove that `STINGY SAT` is NP-complete.

Solution:

First, we show that Stingy SAT is in NP. Let the input formula be denoted by f, and let n be the number of variables and m be the number of clauses. Given a proposed satisfying truth assignment σ for f, we can just check clause by clause that σ satisfies at least one literal in each clause; this takes $O(n)$ time per clause and thus $O(nm)$ total time. Then in $O(n)$ time, we count how many variables are set to be TRUE in σ and check that the number is larger than k or not.

Now, we show that Stingy SAT is as hard as a problem known to be NP-Complete by reducing SAT $\rightarrow$ Stingy SAT. Suppose you have an input formula f for SAT with n variables and m clauses. Now, run Stingy SAT on f with $k = n$. Clearly, every assignment over n variables will have at most n variables set to true, so this extra restriction will not constrain the set of solutions. Then, if Stingy SAT returns a satisfying assignment, that assignment will also satisfy the original SAT problem. Likewise, if Stingy SAT returns that there is no such assignment, then there can be no solution to the SAT problem. The only transformation between SAT and Stingy-SAT was setting $k = n$; the input formula f and resulting truth assignment from Stingy SAT remain unchanged, so our transformations take $O(1)$ constant time (which is polynomial). Then, since SAT is NP-complete, Stingy SAT must at least as hard as SAT.

Given that Stingy SAT is in NP and at least as hard as SAT, we conclude that Stingy SAT is NP-Complete.

3. [DPV] Problem 8.4 (a),(b),(c) (Clique-3)

Consider the CLIQUE problem restricted to graphs in which every vertex has degree at most 3. Call this problem CLIQUE-3.

(a) Prove that CLIQUE-3 is in NP.

Solution:

Given an input graph $G = (V, E)$, a goal g, and a potential solution set $S \subseteq V$, it is easy to verify whether S is a clique by checking that all pairs of vertices in S are connected in $O(n^2)$ time, and to check that $|S| \geq g$ in $O(n)$ time. Since a solution can be verified in polynomial time, clique-3 is in NP.

(b) What is wrong with the following proof of NP-completeness for `CLIQUE-3`? We know that the `CLIQUE` problem in general graphs is NP-complete, so it is enough to present a reduction from `CLIQUE-3` to `CLIQUE`. Given a graph G with vertices of degree ≤ 3, and a

parameter g, the reduction leaves the graph and the parameter unchanged: clearly the output of the reduction is a possible input for the `CLIQUE` problem. Furthermore, the answer to both problems is identical. This proves the correctness of the reduction and, therefore, the NP-completeness of `CLIQUE-3`.

Solution:

The direction of this reduction is wrong. In order to show that CLIQUE-3 is NP-complete, we would instead need to show that some known NP-complete problem (such as CLIQUE) reduces to CLIQUE-3.

We want to prove that CLIQUE-3 is a computational difficult problem. To do that we want to show that if we somehow solve CLIQUE-3 efficiently then we can efficiently solve every problem in NP. We know that CLIQUE is NP-complete and hence if we can efficiently solve CLIQUE then we can efficiently solve every problem in NP. Therefore, we need to show that CLIQUE $\rightarrow$ CLIQUE-3, but the above proof does the reverse reduction.

(c) It is true that the `VERTEX COVER` problem remains NP-complete even when restricted to graphs in which every vertex has degree at most 3. Call this problem `VC-3`. What is wrong with the following proof of NP-completeness for `CLIQUE-3`?

We present a reduction from `VC-3` to `CLIQUE-3`. Given a graph $G = (V, E)$ with node degrees bounded by 3, and a parameter b, we create an instance of `CLIQUE-3` by leaving the graph unchanged and switching the parameter to $|V| - b$. Now, a subset $C \subseteq V$ is a vertex cover in G if and only if the complementary set $V - C$ is a clique in G. Therefore G has a vertex cover of size $\leq b$ if and only if it has a clique of size $\geq |V| - b$. This proves the correctness of the reduction and, consequently, the NP-completeness of `CLIQUE-3`.

Solution:

The reduction is incorrect. Specifically, the statement "a subset $C \subseteq V$ is a vertex cover in G if and only if the complementary set $V - C$ is a clique in the same graph G" is incorrect. While it would be correct to say "a subset $C \subseteq V$ is a vertex cover in G if and only if the complementary set $V - C$ is a clique in the complementary graph $\bar{G} = (V, V^2 - E)$", the complement of G is not guaranteed to retain the property that every vertex has at degree at most 3, so this alternate statement is not useful for the desired reduction.

4. [DPV] Problem 8.10 (a) (Subgraph isomorphism)

Proving NP-completeness by generalization. For each of the problems below, prove that it is NP-complete by showing that it is a generalization of some NP-complete problem we have seen in this chapter.

(a) `SUBGRAPH ISOMORPHISM`: Given as input two undirected graphs G and H, determine whether G is a subgraph of H (that is, whether by deleting certain vertices and edges of H we obtain a graph that is, up to renaming of vertices, identical to G), and if so, return the corresponding mapping of V (G) into V (H).

Solution:

First, we show that Subgraph isomorphism is in NP. For a pair of graphs $G = (V, E)$ and $H = (V', E')$, given a mapping π from V to V', we can verify the solution in at most $O((|V| + |E|)|E'|)$ polynomial time by checking that each edge $(u, v) \in E$ from G maps to an edge $(\pi(u), \pi(v)) \in E'$ from H.

Now, we show: Clique $\rightarrow$ Subgraph isomorphism. Let the inputs to Clique be $H = (V', E')$ and an integer k. To construct the input to Subgraph isomorphism, let $G = (V, E)$ be a complete graph (every pair of distinct vertices is connected by a unique edge) where $|V| = k$. Run Subgraph isomorphism on G and H. H has an clique of size k if and only if there is a solution to Subgraph isomorphism, with the clique being the set of vertices that map to the subgraph G. The solution will also provide a mapping from V to V', allowing us to find the clique in the original graph H. Why do we know this is true? A clique is just a complete graph, so finding a clique in H is a special case of Subgraph isomorphism where G is a complete graph.

This input reduction takes polynomial time, since it creates a complete graph with k vertices in $O(k^2)$ time (k being bounded by $|V'|$, the number of vertices in H). Likewise, the output mapping is used to determine the vertices of the Clique in H in $O(|V|)$ time.

Then, since Clique is NP-complete, Subgraph isomorphism must be as well.

5. (Algorithm design problem)

This problem is designed to make you think about the following idea: sometimes a combination (or modification) of a hard problem is not necessarily more difficult. Problem 8.14 explores this idea further.

Design an efficient algorithm to solve the following problem:
Input: A directed graph $G = (V; E)$.
Output: A cycle if G has one, or a path that visits all vertices, if there is one.

Note that your algorithm will output NO only if there are no cycles in G and there is no path visiting all vertices.

Solution:

Let's break the output into three parts: (1) if a cycle exists, output it; (2) if no cycle exists but a path exists, output it; if neither a cycle nor a path exists, output NO. Tackling each in order provides the solution.

First, consider what it means for a directed graph to contain a cycle – this occurs when there exists a back edge, something easily detected via DFS. So, for Step 1 we run DFS on G. We then, for each edge $\overrightarrow{(v,u)} \in E$, look to find an edge such that $pre[u] < pre[v] < post[v] < post[u]$ – a back edge which indicates a cycle exists between u and v. We can then recover the cycle by examining the $prev[]$ array which defines the DFS tree. And we note that each step (DFS, checking for a back edge, recreating the cycle) takes $O(n+m)$ time.

Now, consider what it means if no cycle is found – our directed graph is acyclic, we have DAG. And in a DAG, there exists a path which visits all vertices only if there is a single unique topological sorting. How do we determine if that is the case? DFS output a topological sorting of the vertices: we use that sorting to confirm that for each vertex, a directed edge exists to the next vertex in topological order. That is, given $topo()$ which represents the vertices sorted in descending post-order, for each value $topo(i)$ where $1 \leq i < n$, there exists an edge$\overrightarrow{(topo[i], topo[i+1])}$. The sorting is the path which visits all vertices. Again, DFS takes $O(n+m)$ time, as does checking the topological sorting.

If neither case is true we output NO – there are no cycles present, and there is no path visiting all vertices.

The lesson? While the general search problem of finding a Hamiltonian/Rudrata path in a directed graph is NP-Hard, modifying the problem such that we constrain the graph by eliminating the presence of a cycle makes it a problem in P.