Good problems

[DPV] 8.7. Consider a special case of 3SAT in which all clauses have exactly three literals, and each variable appears at most three times. Show that this problem can be solved in polynomial time. (*Hint:* create a bipartite graph with clauses on the left, variables on the right, and edges whenever a variable appears in a clause. Use Exercise 7.30 to show that this graph has a matching.)

[KT] **41.** Given a directed graph G, a *cycle cover* is a set of node-disjoint cycles so that each node of G belongs to a cycle. The *Cycle Cover Problem* asks whether a given directed graph has a cycle cover.

(a) Show that the Cycle Cover Problem can be solved in polynomial time. (*Hint:* Use Bipartite Matching.)

(b) Suppose we require each cycle to have at most three edges. Show that determining whether a graph G has such a cycle cover is NP-complete.

[CLRS] ***26.2-11***

The ***edge connectivity*** of an undirected graph is the minimum number k of edges that must be removed to disconnect the graph. For example, the edge connectivity of a tree is 1, and the edge connectivity of a cyclic chain of vertices is 2. Show how to determine the edge connectivity of an undirected graph $G = (V, E)$ by running a maximum-flow algorithm on at most $|V|$ flow networks, each having $O(V)$ vertices and $O(E)$ edges.

[CLRS] ***26-4 Updating maximum flow***

Let $G = (V, E)$ be a flow network with source s, sink t, and integer capacities. Suppose that we are given a maximum flow in G.

a. Suppose that we increase the capacity of a single edge $(u, v) \in E$ by 1. Give an $O(V + E)$-time algorithm to update the maximum flow.

b. Suppose that we decrease the capacity of a single edge $(u, v) \in E$ by 1. Give an $O(V + E)$-time algorithm to update the maximum flow.

[DPV] 7.19. Suppose someone presents you with a solution to a max-flow problem on some network. Give a *linear* time algorithm to determine whether the solution does indeed give a maximum flow.

[KT] **12.** Consider the following problem. You are given a flow network with unit-capacity edges: It consists of a directed graph $G = (V, E)$, a source $s \in V$, and a sink $t \in V$; and $c_e = 1$ for every $e \in E$. You are also given a parameter k.

The goal is to delete k edges so as to reduce the maximum s-t flow in G by as much as possible. In other words, you should find a set of edges $F \subseteq E$ so that $|F| = k$ and the maximum s-t flow in $G' = (V, E - F)$ is as small as possible subject to this.

Give a polynomial-time algorithm to solve this problem.

[DPV] 8.10. *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 $V(H)$.

(b) LONGEST PATH: Given a graph G and an integer g, find in G a simple path of length g.

(c) MAX SAT: Given a CNF formula and an integer g, find a truth assignment that satisfies at least g clauses.

(d) DENSE SUBGRAPH: Given a graph and two integers a and b, find a set of a vertices of G such that there are at least b edges between them.

(e) SPARSE SUBGRAPH: Given a graph and two integers a and b, find a set of a vertices of G such that there are at most b edges between them.

(f) SET COVER. (This problem generalizes *two* known **NP**-complete problems.)

(g) RELIABLE NETWORK: We are given two $n \times n$ matrices, a distance matrix d_{ij} and a *connectivity requirement* matrix r_{ij}, as well as a budget b; we must find a graph $G = (\{1, 2, \ldots, n\}, E)$ such that (1) the total cost of all edges is b or less and (2) between any two distinct vertices i and j there are r_{ij} vertex-disjoint paths. (*Hint:* Suppose that all d_{ij}'s are 1 or 2, $b = n$, and all r_{ij}'s are 2. Which well known **NP**-complete problem is this?)

[DPV] 8.20. In an undirected graph $G = (V, E)$, we say $D \subseteq V$ is a *dominating set* if every $v \in V$ is either in D or adjacent to at least one member of D. In the DOMINATING SET problem, the input is a graph and a budget b, and the aim is to find a dominating set in the graph of size at most b, if one exists. Prove that this problem is **NP**-complete.

[KT] **9.** Consider the following problem. You are managing a communication network, modeled by a directed graph $G = (V, E)$. There are c *users* who are interested in making use of this network. User i (for each $i = 1, 2, \ldots, c$) issues a *request* to reserve a specific path P_i in G on which to transmit data.

You are interested in accepting as many of these path requests as possible, subject to the following restriction: if you accept both P_i and P_j, then P_i and P_j cannot share any nodes.

Thus, the *Path Selection Problem* asks: Given a directed graph $G = (V, E)$, a set of requests $P_1, P_2, \ldots, P_c$—each of which must be a path in G—and a number k, is it possible to select at least k of the paths so that no two of the selected paths share any nodes?

Prove that Path Selection is NP-complete.

[KT] 14 We've seen the Interval Scheduling Problem in Chapters 1 and 4. Here we consider a computationally much harder version of it that we'll call *Multiple Interval Scheduling*. As before, you have a processor that is available to run jobs over some period of time (e.g., 9 A.M. to 5 P.M).

People submit jobs to run on the processor; the processor can only work on one job at any single point in time. Jobs in this model, however, are more complicated than we've seen in the past: each job requires a *set* of intervals of time during which it needs to use the processor. Thus, for example, a single job could require the processor from 10 A.M. to 11 A.M., and again from 2 P.M. to 3 P.M.. If you accept this job, it ties up your processor during those two hours, but you could still accept jobs that need any other time periods (including the hours from 11 A.M. to 2 A.M.).

Now you're given a set of n jobs, each specified by a set of time intervals, and you want to answer the following question: For a given number k, is it possible to accept at least k of the jobs so that no two of the accepted jobs have any overlap in time?

Show that Multiple Interval Scheduling is NP-complete.

[KT] **28.** The following is a version of the Independent Set Problem. You are given a graph $G = (V, E)$ and an integer k. For this problem, we will call a set $I \subset V$ *strongly independent* if, for any two nodes $v, u \in I$, the edge (v, u) does not belong to E, and there is also no path of two edges from u to v, that is, there is no node w such that both $(u, w) \in E$ and $(w, v) \in E$. The Strongly Independent Set Problem is to decide whether G has a strongly independent set of size at least k.

Prove that the Strongly Independent Set Problem is NP-complete.

Interesting problems

[CLRS] ***34-4 Scheduling with profits and deadlines***

Suppose that we have one machine and a set of n tasks $a_1, a_2, \ldots, a_n$, each of which requires time on the machine. Each task a_j requires t_j time units on the machine (its processing time), yields a profit of p_j, and has a deadline d_j. The machine can process only one task at a time, and task a_j must run without interruption for t_j consecutive time units. If we complete task a_j by its deadline d_j, we receive a profit p_j, but if we complete it after its deadline, we receive no profit. As an optimization problem, we are given the processing times, profits, and deadlines for a set of n tasks, and we wish to find a schedule that completes all the tasks and returns the greatest amount of profit. The processing times, profits, and deadlines are all nonnegative numbers.

a. State this problem as a decision problem.

b. Show that the decision problem is NP-complete.

c. Give a polynomial-time algorithm for the decision problem, assuming that all processing times are integers from 1 to n. (*Hint:* Use dynamic programming.)

d. Give a polynomial-time algorithm for the optimization problem, assuming that all processing times are integers from 1 to n.

[CLRS]

26-2 Minimum path cover

A ***path cover*** of a directed graph $G = (V, E)$ is a set P of vertex-disjoint paths such that every vertex in V is included in exactly one path in P. Paths may start and end anywhere, and they may be of any length, including 0. A ***minimum path cover*** of G is a path cover containing the fewest possible paths.

a. Give an efficient algorithm to find a minimum path cover of a directed acyclic graph $G = (V, E)$. (*Hint:* Assuming that $V = \{1, 2, \ldots, n\}$, construct the graph $G' = (V', E')$, where

$$\begin{aligned} V' &= \{x_0, x_1, \ldots, x_n\} \cup \{y_0, y_1, \ldots, y_n\} \ , \\ E' &= \{(x_0, x_i) : i \in V\} \cup \{(y_i, y_0) : i \in V\} \cup \{(x_i, y_j) : (i, j) \in E\} \ , \end{aligned}$$

and run a maximum-flow algorithm.)

b. Does your algorithm work for directed graphs that contain cycles? Explain.

Challenging problems (optional)

[DPV] 8.23. In the NODE-DISJOINT PATHS problem, the input is an undirected graph in which some vertices have been specially marked: a certain number of "sources" $s_1, s_2, \ldots s_k$ and an equal number of "destinations" $t_1, t_2, \ldots t_k$. The goal is to find k node-disjoint paths (that is, paths which have no nodes in common) where the ith path goes from s_i to t_i. Show that this problem is **NP**-complete.

Here is a sequence of progressively stronger hints.

(a) Reduce from 3SAT.

(b) For a 3SAT formula with m clauses and n variables, use $k = m+n$ sources and destinations. Introduce one source/destination pair (s_x, t_x) for each variable x, and one source/destination pair (s_c, t_c) for each clause c.

(c) For each 3SAT clause, introduce 6 new intermediate vertices, one for each literal occurring in that clause and one for its complement.

(d) Notice that if the path from s_c to t_c goes through some intermediate vertex representing, say, an occurrence of variable x, then no other path can go through that vertex. What vertex would you like the other path to be forced to go through instead?