1. Traveling salesperson problem (TSP)

There are n cities, and there is a positive (integral) cost c_{ij} of traveling from the i-th city to the j-th city. The costs need not be symmetric, that is, we may have $c_{ij} \neq c_{ji}$. Moreover, we may have $c_{ij} + c_{jk} \neq c_{ik}$. A salesperson starts at a given city, visits every city once and only once, and eventually comes back to the city from where the tour started. The objective of the salesperson is to minimize the total cost of traveling, that is, to find a minimum-cost Hamiltonian cycle in a complete weighted directed graph G on n vertices. This is an optimization problem. We consider the equivalent decision problem TSP that, given G and a positive integer k, decides whether G contains a (directed) Hamiltonian cycle of total cost $\leq k$. Prove that TSP is NP-complete.

Solution sketch:

A directed Hamiltonian cycle in G of cost $\leq k$ is a succinct certificate for G to be in Accept(TSP), so TSP is in NP.

In order to prove the NP-hardness of TSP, we make a reduction from the D-HAM-CYCLE problem. Let $G = (V, E)$ be an instance of D-HAM-CYCLE with $|V| = n$. Create a complete graph on the vertex set V, and assign the costs $c_{ij} = 1$ if (i, j) is in E, or $c_{ij} = n + 1$ if (i, j) is not in E. Take $k = n$.

2. Longest path problem

Let $G = (V, E)$ be a weighted graph (directed or undirected), s and t two vertices in V, and k a positive integer. Assume that the edge weights are positive (or non-negative). Prove that the problem of deciding whether G contains an s,t path of weight $\geq k$ is NP-complete.

Solution sketch:

Reduce from HAM-PATH (or D-HAM-PATH). Let G be an instance of this problem. Assign the cost of 1 to each edge of G. Take $k = n - 1$.

3. Shortest path problem

Let $G = (V, E)$ be a weighted graph (directed or undirected), s and t two vertices in V, and k a positive integer. Consider the problem of deciding whether G contains an s, t path of weight $\leq k$. Justify whether this problem is NP-complete if

(a) all edge weights are positive (or non-negative),
(b) the edge weights may be positive, negative, or zero.

Solution sketch

(a) In this case, the problem is in P (and not NP-complete unless P = NP). Recall the Dijkstra or the Floyd–Warshall algorithm.

(b) In this case, the problem is NP-complete. Reduce from LONGEST-PATH. If G is an instance of LONGEST-PATH (with non-negative weights), just negate the weights to create an instance of SHORTEST-PATH (with negative weights allowed).

4. Let G be an undirected graph on n vertices. Which of the following problems is/are NP-complete? Justify.

(a) Decide whether G contains a clique of size $\geq n - 5$.
(b) Decide whether G contains an independent set of size $\geq n - 5$.
(c) Decide whether G contains a vertex cover of size $\geq n - 5$.
(d) Decide whether G contains a cycle of length $\geq n - 5$.
(e) Decide whether G contains a path of length $\geq n - 5$.

Solution sketch:

(a) – (c) are in P. We can find a solution (if it exists) for (a) and (b) in polynomial time by exhaustive search. G contains a clique/IS of size $\geq n - 5$ if and only if G contains a clique/IS of size equal to $n - 5$. The number of subsets U of V of size equal to $n - 5$ is $C(n, n - 5) = \theta(n^5)$. Each such subset U can be checked in polynomial time. For (c), the answer is always *Yes*, because any subset of V of size $n - 1$ (or n) is a vertex cover of G.

(d) Reduce from HAM-CYCLE. Add five isolated vertices.

(e) Reduce from HAM-PATH. Let (G, s, t) be an instance of HAM-PATH. Add two new vertices s' and t'. Connect s' to s, and t' to t. Add four more new isolated vertices.

5. Partition problem

You are given n positive integers $a_1, a_2, \ldots, a_n$ such that

$$a_1 + a_2 + \cdots + a_n = A \text{ is even.}$$

Decide whether the given integers can be partitioned into two subcollections such that the sum of the integers in each subcollection is $A / 2$. Prove that PARTITION is NP-complete.

Solution sketch:

Reduce from SUBSET-SUM. Let (S, t) be an instance of SUBSET-SUM with T equal to the sum of all the elements of S. Add to S two new elements $2T - t$ and $T + t$.

6. Bin-packing problem

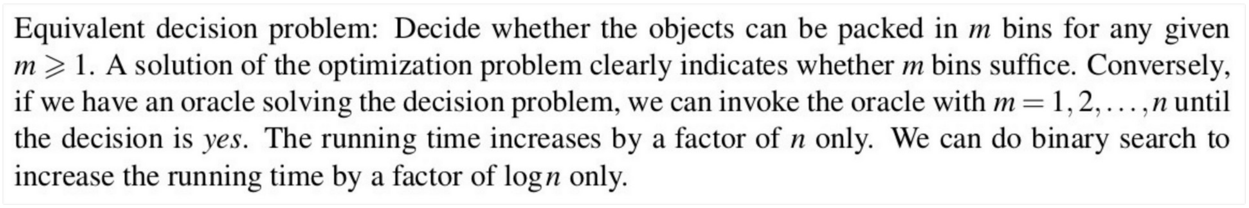
You are given n objects of weights w_1, w_2, …, w_n, and an infinite supply of bins each with weight capacity C. You want to pack all the objects in m bins such that m is as small as possible. Assume that each $w_i \leq C$.

(a) Frame an equivalent decision problem, and prove that the decision problem can be solved in polynomial time if and only if the optimization problem can be solved in polynomial time.

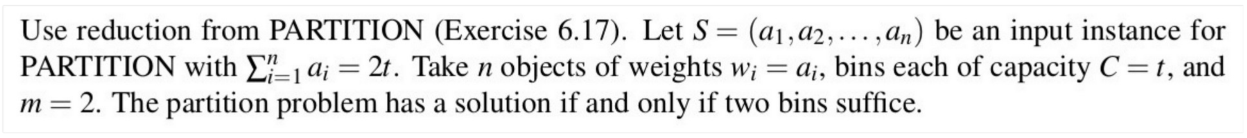
(b) Prove that the decision version is NP-complete.

Solution sketch:

(a)


Equivalent decision problem: Decide whether the objects can be packed in m bins for any given $m \geqslant 1$. A solution of the optimization problem clearly indicates whether m bins suffice. Conversely, if we have an oracle solving the decision problem, we can invoke the oracle with $m = 1, 2, \ldots, n$ until the decision is *yes*. The running time increases by a factor of n only. We can do binary search to increase the running time by a factor of $\log n$ only.

(b)


Use reduction from PARTITION (Exercise 6.17). Let $S = (a_1, a_2, \ldots, a_n)$ be an input instance for PARTITION with $\sum_{i=1}^{n} a_i = 2t$. Take n objects of weights $w_i = a_i$, bins each of capacity $C = t$, and $m = 2$. The partition problem has a solution if and only if two bins suffice.

7. Knapsack problem

A thief finds n objects of weights w_1, w_2, …, w_n and positive integer-valued profits p_1, p_2, …, p_n. The thief has a knapsack of capacity C. The goal of the thief is to pack objects in the knapsack without exceeding its capacity, so as to maximize the profit of packed objects.

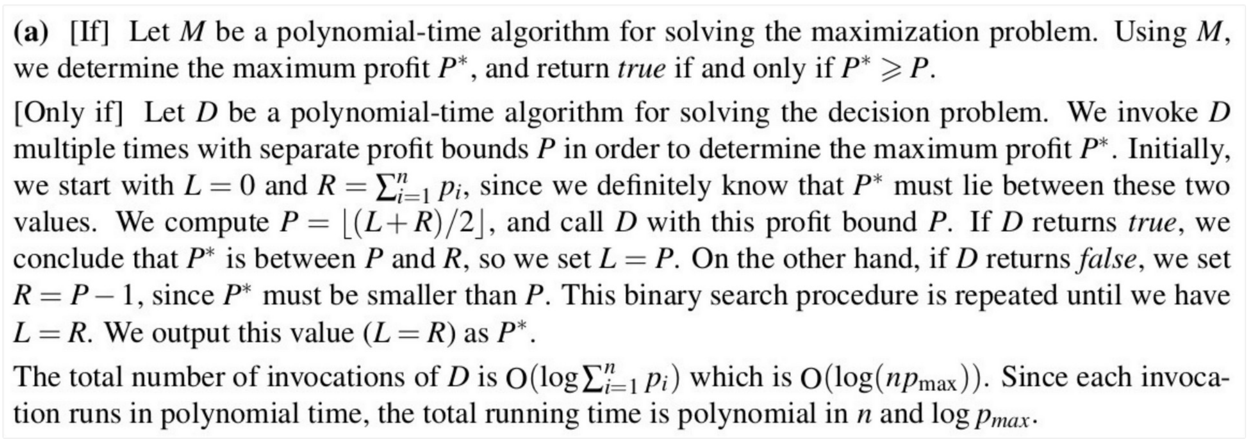
0,1 variant: Each object can be either packed or discarded.
Fractional variant: Any fraction of any object can be packed.

(a) Formulate an equivalent decision version of the 0,1 knapsack problem.
(b) Prove that the decision problem of Part (b) is NP-complete.
(c) Prove that the fractional knapsack problem is in P.

Solution sketch:

(c) Pack the objects in the decreasing order of p_i/w_i values. This is a greedy algorithm. Prove its correctness.

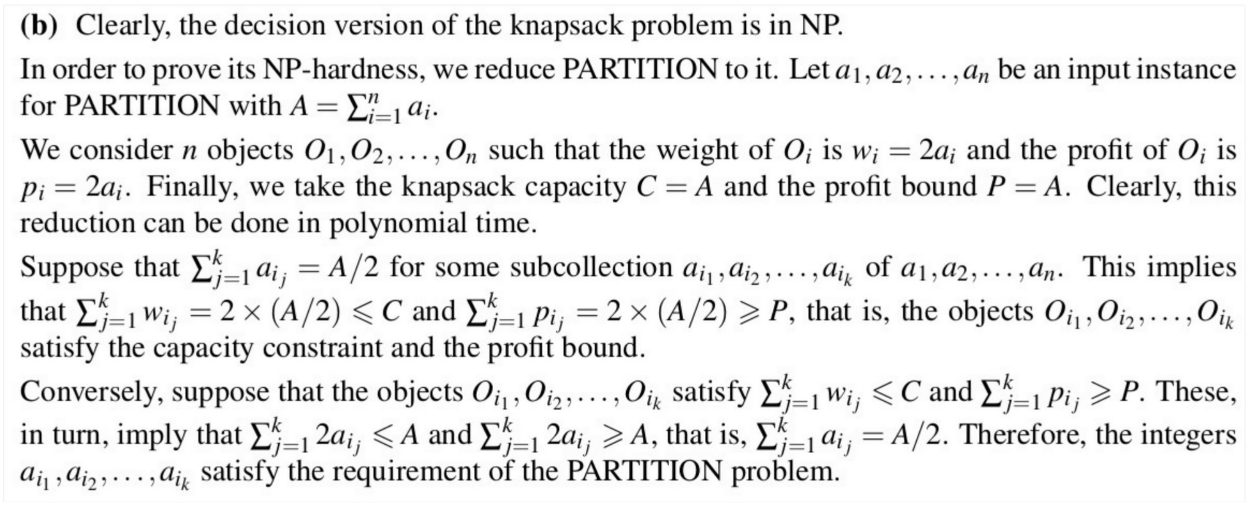
(a) Equivalent decision problem: Given w_1, w_2, …, w_n, p_1, p_2, …, p_n, C, and a (positive) integer P, decide whether a profit of $\geqslant P$ is achievable without exceeding the knapsack capacity C.



(a) [If] Let M be a polynomial-time algorithm for solving the maximization problem. Using M, we determine the maximum profit P^*, and return *true* if and only if $P^* \geqslant P$.

[Only if] Let D be a polynomial-time algorithm for solving the decision problem. We invoke D multiple times with separate profit bounds P in order to determine the maximum profit P^*. Initially, we start with $L = 0$ and $R = \sum_{i=1}^{n} p_i$, since we definitely know that P^* must lie between these two values. We compute $P = \lfloor (L+R)/2 \rfloor$, and call D with this profit bound P. If D returns *true*, we conclude that P^* is between P and R, so we set $L = P$. On the other hand, if D returns *false*, we set $R = P - 1$, since P^* must be smaller than P. This binary search procedure is repeated until we have $L = R$. We output this value ($L = R$) as P^*.

The total number of invocations of D is $O(\log \sum_{i=1}^{n} p_i)$ which is $O(\log(np_{\max}))$. Since each invocation runs in polynomial time, the total running time is polynomial in n and $\log p_{max}$.



(b) Clearly, the decision version of the knapsack problem is in NP.

In order to prove its NP-hardness, we reduce PARTITION to it. Let $a_1, a_2, \ldots, a_n$ be an input instance for PARTITION with $A = \sum_{i=1}^{n} a_i$.

We consider n objects $O_1, O_2, \ldots, O_n$ such that the weight of O_i is $w_i = 2a_i$ and the profit of O_i is $p_i = 2a_i$. Finally, we take the knapsack capacity $C = A$ and the profit bound $P = A$. Clearly, this reduction can be done in polynomial time.

Suppose that $\sum_{j=1}^{k} a_{i_j} = A/2$ for some subcollection $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ of $a_1, a_2, \ldots, a_n$. This implies that $\sum_{j=1}^{k} w_{i_j} = 2 \times (A/2) \leqslant C$ and $\sum_{j=1}^{k} p_{i_j} = 2 \times (A/2) \geqslant P$, that is, the objects $O_{i_1}, O_{i_2}, \ldots, O_{i_k}$ satisfy the capacity constraint and the profit bound.

Conversely, suppose that the objects $O_{i_1}, O_{i_2}, \ldots, O_{i_k}$ satisfy $\sum_{j=1}^{k} w_{i_j} \leqslant C$ and $\sum_{j=1}^{k} p_{i_j} \geqslant P$. These, in turn, imply that $\sum_{j=1}^{k} 2a_{i_j} \leqslant A$ and $\sum_{j=1}^{k} 2a_{i_j} \geqslant A$, that is, $\sum_{j=1}^{k} a_{i_j} = A/2$. Therefore, the integers $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ satisfy the requirement of the PARTITION problem.

8. MAX-CUT problem

Let $G = (V, E)$ be an undirected graph. We want to find a cut X, Y of V such that the number of edges connecting X and Y is as large as possible.

(a) Propose a decision version of the MAX-CUT problem. Prove that the maximization problem can be solved in polynomial time if and only if the decision problem can be solved in polynomial time.

(b) Prove that the decisional MAX-CUT problem is NP-complete.

Solution sketch:

(a) Decision version: Given G and a positive integer k, decide whether G has a cut of size $\geqslant k$.

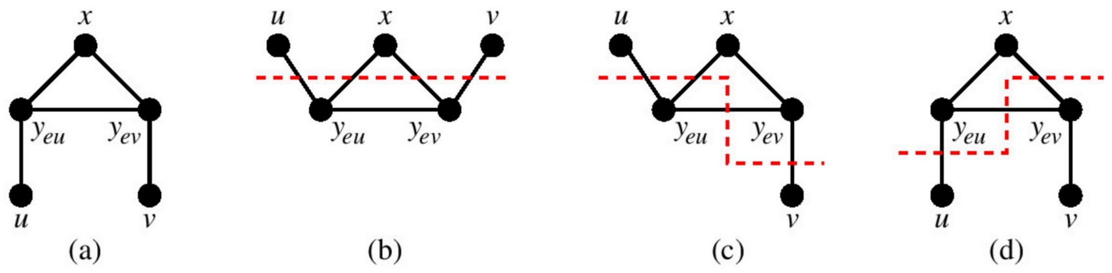
If the maximization problem can be solved in polynomial time, the decision version can evidently be solved in poynomial time.

Conversely, suppose that A is a polynomial-time algorithm for solving the decision version. We first compute the number m of edges in G. Then, we repeatedly call A with $k = m - 1$, $m - 2$, $m - 3$, … until A says *Yes*. The running time of this optimization algorithm is m times the running time of A. So if A runs in polynomial time, so too does this optimization algorithm. You may use binary search to limit the increase of the running to a factor of log m only, but this is not a critical issue for the proof.

(b)

Clearly, MAX-CUT $\in$ NP. To prove its NP-Hardness, we use reduction from INDEPENDENT-SET. Let $G = (V,E)$ and r constitute an instance of the INDEPENDENT-SET problem. We construct a graph $G' = (V',E')$ for MAX-CUT as follows. We include every vertex of V in V'. We add a new vertex x to V', and add x to all vertices in the copy of V in V'. For each edge $e = (u,v) \in E$, we introduce two new vertices y_{eu} and y_{ev} in V'. These new vertices and the vertices $x,u,v \in V'$ are connected as shown in Figure 137(a). Finally, take the cut size for G' as $k = r + 4|E|$.

Figure 137: An edge gadget for the reduction of INDEPENDENT-SET to MAXCUT



The property of the edge gagdet, that makes this construction work, is as follows. Consider a cut of G', and let $e = (u,v)$ be an edge of G. If one or both of u,v lie in the same part as x, then the gadget vertices y_{eu} and y_{ev} can be placed in appropriate parts of the cut such that the gadget for e contributes four edges to the cut (see Parts (b) and (c) of Figure 137). On the other hand, if both u and v lie on the other part of the cut as x, then no matter in which parts we put y_{eu} and y_{ev}, the edge gadget contributes at most three edges to the cut (Figure 137(d) shows one situation).

Now, suppose that $I \subseteq V$ is an idependent set of G. We produce a cut V_1', V_2' of G' of size exactly equal to $k = r + 4|E|$ as follows. We put x and all vertices of $V \setminus I$ in V_1'. We put all vertices of I in V_2'. Let $e = (u,v)$ be an edge of G. Since I is an independent set, both u and v do not belong to I and so not to V_2' as well. This implies that at least one of u and v, or possibly both, must be in the same part V_1' as x. Consequently, we can place y_{eu} and y_{ev} appropriately in the two parts so that the gadget corresponding to e contributes four edges to the cut. Finally, there are exactly r edges of the form (x,u), $u \in I$, in the cut. Thus, the cut is of the desired size.

Conversely, let V_1', V_2' be a cut of G' of size $k \geqslant r + 4|E|$. Suppose that $x \in V_1'$. Since an edge gadget can contribute at most four edges to a cut, there must exist $r' \geqslant r$ vertices $u \in V$ such that $u \in V_2'$ (consider the edges (x,u) in G'). If these r' vertices of V, that are in V_2', are already independent, we are done. So suppose that some edge $e = (u,v)$ between two of these r' vertices exists in E. The corresponding edge gadget contributes less than four edges to the cut. We can move one of the vertices and switch the side(s) of one or both of the gadget vertices y_{eu} and y_{ev} such that the cut size does not decrease. For example, in the situation of Figure 137(d), we simply transfer v from V_2' to V_1', so the edge gadget now contributes one more cut edge which compensates for the loss of the earlier cut edge (x,v). This process reduces r' by one. We continue doing these vertex switches in the cut until the vertices of V, that remain in V_2', become independent. Let ρ be this final value of r'. By the gadget property, we cannot have $\rho < r$.

9. MAX-3-CUT problem

Let $G = (V, E)$ be an undirected graph, and k a positive integer. Decide whether V can be partitioned into three parts X, Y, Z such that the number of edges in E connecting vertices from different parts is $\geqslant k$. Prove that MAX-3-CUT is NP-complete.

Solution sketch:

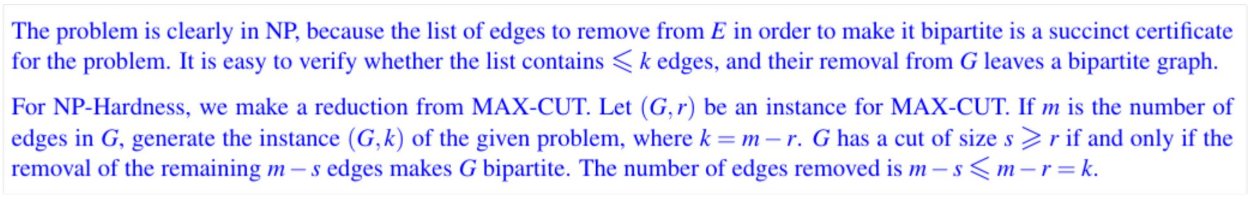
Clearly, MAX-3-CUT is in NP. To prove its NP-hard-ness, we use reduction from MAX-CUT. Let (G, r) be an instance of MAX-CUT. From this, we create an instance (G', k) for MAX-3-CUT as follows. First, make a copy of G to G' (both vertices and edges). Let $|V(G)| = n$. Add n new vertices $w_1, w_2, \ldots, w_n$ to G', and add an edge from each w_i to each old vertex v_j in the copy of G in G'. Finally, take $k = r + n^2$.

If G has a cut X, Y of size $\geqslant r$, then consider the 3-cut X, Y, Z of G', where $Z = \{w_1, w_2, \ldots, w_n\}$.

Conversely, let X, Y, Z be a 3-cut of G' of size $\geqslant k = r + n^2$. Let Z be the part containing the minimum number of old vertices v_j. But then, if some new vertex w_i is not in Z, then relocating w_i from X or Y to Z does not decrease the 3-cut size (w_i is connected to all old vertices). So we can assume that all w_i are in Z. Now, if Z contains one or more of the old vertices v_j, then relocating them from Z to X or Y (in any arbitrary manner) cannot again decrease the 3-cut size. So we have a 3-cut X, Y, Z of G' of size $\geqslant r + n^2$ with $Z = \{w_1, w_2, \ldots, w_n\}$. But then, X, Y corresponds to a 2-cut of G of size $\geqslant r$.

10. Let $G = (V, E)$ be an undirected graph, and k a positive integer. You want to determine whether the removal of some k or fewer edges from E makes G bipartite. Prove that this problem is NP-complete.

Solution sketch



The problem is clearly in NP, because the list of edges to remove from E in order to make it bipartite is a succinct certificate for the problem. It is easy to verify whether the list contains $\leqslant k$ edges, and their removal from G leaves a bipartite graph.

For NP-Hardness, we make a reduction from MAX-CUT. Let (G, r) be an instance for MAX-CUT. If m is the number of edges in G, generate the instance (G, k) of the given problem, where $k = m - r$. G has a cut of size $s \geqslant r$ if and only if the removal of the remaining $m - s$ edges makes G bipartite. The number of edges removed is $m - s \leqslant m - r = k$.

11. Weighted MAX-CUT problem

Let $G = (V, E)$ be an undirected graph with each edge e carrying a positive weight w_e (assume to be a positive integer). The weight of a cut X, Y of V is the sum of the weights of the edges connecting X and Y. Let k be a positive integer. We want to decide whether G has a cut of weight $\geqslant k$. Prove that the weighted MAX-CUT problem is NP-complete.

Solution sketch:

Weighted MAX-CUT is a generalized version of the (unweighted) MAX-CUT problem. More specifically, reduce MAX-CUT to wt-MAX-CUT as follows. Let (G, k) be an instance of MAX-CUT. Take the weight of each edge of G as 1, and pass the same G and k along with these weights to the output.

12. Ferry-loading problem

Several vehicles wait in a long queue near a river bank. The lengths of the vehicles are l_1, l_2, l_3, …, l_n (positive integers) in that order from the beginning to the end of the queue. A big boat with two decks (left and right) comes to carry the vehicles across the river. Each deck of the boat has length L (again a positive integer). The vehicles must be loaded to the boat in the order they appear in the queue. For each vehicle, a decision is to be made about which deck it will join (provided that there is enough length remaining in that deck). The objective is to maximize the number of vehicles that can be loaded. As an example, take $L = 10$, and the first four vehicles having lengths 5, 5, 6, 4. Then all the four of them can be loaded ($5 + 5 = 6 + 4 = 10$). However, if the first two vehicles are loaded to different decks, then no other vehicle can be loaded. Let us consider the decision problem whether all of the vehicles can be loaded. Clearly, if

$$l_1 + l_2 + \cdots + l_n > 2L,$$

the answer is *No*. So assume that $l_1 + l_2 + \cdots + l_n \leq 2L$.

(a) Prove that the optimization problem is polynomial-time equivalent to the decision problem.

(b) Prove that the decision problem is NP-Complete.

Solution sketch:

(a) Let A be an algorithm for solving the decision problem. Run A on the first k vehicles $l_1, l_2, \ldots, l_k$ for $k = n, n-1, n-2, \ldots$ until A returns *Yes*.

(b) If all the vehicles can be loaded, then a specification of which vehicle will go to which deck is a succinct certificate for a feasible loading.

Next, we use reduction from PARTITION. Let $S = \{a_1, a_2, \ldots, a_n\}$ be an instance of PARTITION with $\sum_{i=1}^{n} = 2t$. Take n vehicles of lengths $l_i = a_i$, and the deck capacity $L = t$.

13. Let C be a set of classes. Each class is specified by its duration (a positive integer), can be scheduled to start at any working time of the day, and must finish uninterrupted before the working time ends. Given a positive integer k, we want to find out whether k classrooms suffice for scheduling all the classes in C on a single day. Assume that the working time of a day is from 8:00am to 8:00pm.

Prove/Disprove: This problem is NP-Complete.

Solution sketch:

This problem is NP-complete. Use reduction from BIN-PACKING.

14. Let C again be a set of classes. Each class is specified by a start time and an end time (both in the range from 8:00am to 8:00pm), and must be scheduled during this specified interval. Given a positive integer k, we want to find out whether k classrooms suffice for scheduling all the classes in C on a single day.

Prove/Disprove: This problem is NP-Complete.

Solution sketch:

This problem can be solved in polynomial time. Use the point-sweep algorithm of Exercise 8(a) in the Computational-Geometry set to compute the minimum number m of classrooms needed. Compare m with k.

15. DNFSAT

A Boolean formula φ is given in the disjunctive normal (sum-of-products) form. Prove/Disprove: The problem of deciding whether φ is satisfiable is NP-complete.

Solution sketch:

This problem is in P. A DNF formula is satisfiable if and only if it contains a product without complementary literals.