COMP 271 Design and Analysis of Algorithms
2003 Spring Semester
Question Bank 5 – Selected Solutions

There is some overlap between these question banks and the tutorials. Solving these questions (and those in the tutorials) will give you good practice for the midterm. Some of these questions (or similar ones) will definitely appear on your exam! Note that you do not have to submit answers to these questions for grading. Your TAs will discuss answers to selected questions in the tutorials.

1. Suppose $L \in$ P. Then, as seen in class, $\overline{L} \in$ P. Since P $\subseteq$ NP we have $\overline{L} \in$ NP. But, by definition, this means

$$L = \overline{\overline{L}} \in \text{Co-NP}.$$

   So, we have just shown that $L \in$ P implies $L \in$ Co-NP or P $\subseteq$ Co-NP.

2. We already saw in class that $L \in$ P if and only if $\overline{L} \in$ P, i.e., P = Co-P.

   But if NP $\neq$ Co-NP we would then have

$$\text{P} = \text{NP} \neq \text{Co-NP} = \text{Co-P}$$

   leading to a contradiction.

3. (i) **True**
   (ii) **False**
   (iii) **True**
   (iv) **False**
   (v) **True**

4. (i) No problems in NP can be solved in polynomial time. **Unknown**
   (ii) Every NP-complete problem requires at least exponential time to be solved. **Unknown**
   (iii) X is in NP and X $\leq_P$ SAT. Then X is NP-complete. **False**:

5. Suppose that G is a graph that has a feedback vertex set of size k. The certificate is the set V' of vertices that are in the feedback vertex set. To determine whether V' is indeed a feedback vertex set, we need to test whether every cycle in G passes through at least one of these vertices. Note that there are exponentially many cycles in a graph, so testing each cycle would not run in polynomial time. Instead we delete all the vertices of V' from G, along with any incident edges. Let G' denote the resulting graph. Then we test whether G' has a cycle, by running DFS and checking whether the resulting tree has at least one backedge. If so, V' is not a feedback edge set (since the cycle in G' does not pass through any vertex of V') and otherwise it is.

9. **Set Cover (SC) $\in$ NP**

Th certificate consists of the subset C of F. Given C, we first check that C contains k sets. This can be done in $O(k)$ time. Note that the input size is at least k since F contains at least k sets. Next, for each element $x \in X$, we check if x appears in some set in C. This takes $O(k|X|^2)$ time, since each set in C has at most $|X|$ elements. Since both checks can be done in time polynomial in the input size, we conclude that the set cover problem is in NP.

Vertex Cover $\leq_P$ Set Cover

Goal:

We want a polynomial time computable function, which given an instance of the VC problem (a graph G and integer k) produces an instance of the SC problem (a domain X, a family of sets F over X, and integer k') such that G has a vertex cover of size k if and only if F has a subfamily of k sets that covers X.

Transformation:

Let $G(V, E)$ and k be an instance of the vertex cover problem. Define $X = E$, the set of edges of G. For each vertex v of G we create one set S_v which contains all the edges that are incident to v. In particular, for each $v \in V$, define

$$S_v = \{e \in E \mid e \text{ is incident to } v \text{ in } G\}.$$

Now let $F = \bigcup_{v \in V}\{S_v\}$. Intuitively, each vertex is associated with the set of edges it covers. Finally let $k' = k$. Output (X, F, k'). Clearly this can be computed in polynomial time.

Correctness:

If G has a vertex cover V' of size k, then we claim that the corresponding sets S_v for each $v \in V'$ form a set cover for X. The reason is that by definition of a VC, every edge is incident to a vertex in V' implying that every $e \in X$ is a member of the set corresponding to a vertex in V'. This collection of sets forms a set cover of size $k = k'$. Conversely, if $C = \{S_{v_1}, S_{v_2}, \ldots, S_{v_k}\}$ is a set cover for X, then every $e \in X$ is in one of these sets, implying that the corresponding vertices $V' = \{v_1, v_2, \ldots, v_k\}$ form a vertex cover of size k for G.

10. **Set-Partition (PART) $\in$ NP**

First we show that partition is in NP. The certificate for the partition problem consists of the two sublists S_1 and S_2. Given the sublists, in polynomial time we can compute the sums of the elements in these lists and verify that they are equal.

Subset Sum $\leq_P$ PART

Goal:

Next we want to show that subset sum (SS) is polynomially reducible to the set-partition problem (PART). That is, we want a polynomial time computable function f, which given an instance of SS (a set of numbers $S = \{x_1, x_2, \ldots, x_n\}$ and a target value t) outputs an instance of PART (a set of numbers $S' = \{x'_1, x'_2, \ldots, x'_n\}$) such

that S has a subset summing to t if and only if S' can be partitioned into subsets S_1 and S_2 that sum to the same value.

Transformation:

Observe that the set-partition problem is a special case of the subset sum problem where we are trying to find a set of numbers S_1 that sum to half the total sum of the whole set. Let T be the sum of all the numbers in S.

$$T = \sum_{i=1}^{n} x_i.$$

If $t = T/2$ then the subset sum problem is an instance of the partition problem, and we are done. If not, then the reduction will create a new number, which if added to any subset that sums to t, will now cause that set to sum to half the elements of the total. The problem is that when we add this new element, we change the total as well, so this must be done carefully.

We may assume that $t \leq T/2$, since otherwise the subset sum problem is equivalent to searching for a subset of size $T - t$, and then taking the complement. Create a new element $x_0 = T - 2t$, and call partition on this modified set. Let S' be S together with this new element: $S' = S \cup \{x_0\}$. Clearly the transformation can be done in polynomial time.

Correctness:

To see why this works, observe that the sum of elements in S' is $T+T-2t = 2(T-t)$. If there is a solution to the subset sum problem, then by adding in the element x_0 we get a collection of elements that sums to $t + (T - 2t) = T - t$, but this is one half of the total $2(T - t)$, and hence is a solution to the set-partition problem. Conversely, if there is a solution to this set-partition problem, then one of the halves of the partition contains the element x_0, and the remaining elements in this half of the partition must sum to $(T - t) - (T - 2t) = t$. Thus these elements (without x_0) form a solution to the subset sum problem.

Here is an example. $S = \{5, 6, 7, 9, 11, 13\}$ and $t = 21$ is an instance of the subset sum problem. $T = 51$. We create the element $x_0 = T - 2t = 9$ and add it to form $S' = \{5, 6, 7, 9, 9, 11, 13\}$. Note that there is a solution to the subset sum since $5 + 7 + 9 = 21$. The total of elements in S' is 60, and so by including x_0 we get a solution to the partition problem $5 + 7 + 9 + 9 = 30 = 60/2$.

11. **Integer-Programming (IP) $\in$ NP**

First we show that integer-programming is in NP. The certificate is the n-vector x with elements in the set $\{0, 1\}$. Given the certificate, it can be verified in linear time (hence, polynomial time) that it satisfies $Ax \leq b$.

3-SAT $\leq_P$ IP

Goal:

Next we want to show that 3-SAT is polynomially reducible to the integer-programming problem (IP). That is, we want a polynomial time computable function, which given

an instance of 3-SAT (a set of variables Y and clauses C) outputs an instance of IP (an m-by-n matrix A and a m-vector b) such that the clauses C are satisfiable if and only if there is an integer n-vector x with elements in the set $\{0, 1\}$ which satisfies $Ax \leq b$.

Transformation:

For each clause C we write out an inequality as follows. If the literal is variable y_i, we replace it by the variable x_i; and if the literal is $\overline{y}_i$, we replace it by the quantity $1 - x_i$. In what follows, we will call this quantity that replaces the literal a *term*. Finally, the sum of these terms is set ≥ 1. For example, if the 3-SAT instance consists of two clauses $y_1 \vee \overline{y_2} \vee \overline{y_3}$ and $\overline{y_1} \vee y_3 \vee y_4$, then the inequalities are $x_1 + (1 - x_2) + (1 - x_3) \geq 1$ and $(1 - x_1) + x_3 + x_4 \geq 1$. One can obtain the matrix A and vector b from these inqualities quite easily. The entire transformation can be done in linear time (hence polynomial time).

Correctness:

If the clauses are satisfiable then consider the satisfying truth assignment. For each variable y_i, if y_i is true, then the corresponding variable x_i is assigned value 1, and if y_i is false, then the corresponding variable x_i is assigned value 0. Consider any clause $c \in C$. Clearly, the satisfying truth assignment makes at least one literal in the clause c true. If this literal is the variable y_i then x_i has value 1 and the inequality corresponding to clause c is satisfied; and if this literal is $\overline{y_i}$ then x_i has value 0, and so $1 - x_i$ has value 1 and again the inequality corresponding to clause c is satisfied. Since the argument applies to any clause c, it implies that all the inequalities are satisfied by the above assignment.

Conversely, if all the inequalities can be satisfied then consider the assignment of values to the variables x_i that satisfies the inequalities. For each variable x_i, if x_i is 1, then the corresponding variable y_i is set to true, and if x_i is 0, then the corresponding variable y_i is set to false. Consider any inequality and the corresponding clause $c \in C$. Clearly, if the inequality is satisfied then one of the three *terms* must be ≥ 1 (since all the terms are ≥ 0). If the term is of the form x_i then it implies that x_i is 1 and the corresponding literal y_i is true; and if the term is of the form $1 - x_i$ then it implies that x_i is 0 and the corresponding literal $\overline{y_i}$ is true. In either case, the clause c is satisfied. Our argument applies to any arbitrary clause c, hence the given clauses are satisfiable.