# COMS 4236 Homework 4

Mengqi Zong $< mz2326@columbia.edu >$

April 5, 2012

## Problem 1

First, the Partition Problem is in NP.

For a given partition, We can easily verify that, whether the two sums are equal or not in polynomial time.

Second, Partition Problem is NP-hard.

We can reduce Subset Sum to Partition Problem. Let the sum of all integers in S be $s$. If $t \geq s$, then we already solved this problem. If $t < s$, then

- If $t = s/2$, then the Subset Sum is a Partition Problem.
- If $t < s/2$, then we can create a new set $P = S \cup \{s - 2t\}$. Then we try to see if P can be equally partitioned. Since the sum of all integers in P is $2s - 2t$, if P can be equally partitioned, then the sum of each subcollection is $s - t$. Since number $s - 2t$ is in one of the subcollections, then the set S must have a subcollection whose sum is $t$.
- If $t > s/2$, then we create a new set $P = S \cup \{2t - s\}$. And we try to see if P can be equally partitioned. Since the sum of the integers in P is $2t - 2s$, if P can be equally partitioned, then the sum of each subcollection is $t$. Since number $s - 2t$ can only be in one of the subcollections, then the set S must have a subcollection whose sum is $t$.

Note that this reduction takes polynomial time.

To sum up, Partition Problem is NP-complete.

## Problem 2

a. First, the kernel can't contain none of $u, v$.

If the kernel $K$ does not include any nodes in $u$ and $v$, then condition (2) of a kernel can't hold. Because $u, v \notin K$ and there are no other edges entering the nodes $u, v$. For nodes $u, v$, they are neither in $K$ nor have an incoming edge from some node in K.

Second, the kernel can't contain both $u$ and $v$.

If the kernel $K$ contains both $u$ and $v$, then condition (1) of a kernel can't hold: for nodes $u, v \in K$ and there are edges $(u, v), (v, u)$.

Third, the kernel can contain exactly one of the two nodes without violating the definition of a kernel.

To sum up, any kernel of G must include exactly one of the two nodes $u, v$.

b. Suppose there exists a kernel $K$ of G such that any node in $K$ does not contain some node (distinct from $u, v, w$) that has an edge to at least one of the three nodes $u, v, w$.

If there is no such edge that comes from some node in $K$ that is distinct from the three nodes, then some of the nodes of $u, v, w$ must be in the kernel. Because, if none of the three nodes is in the kernel, condition (2) can't hold.
Suppose if some node in $u, v, w$ is in the kernel. Without loss of generality, let $u$ be in the kernel. Then $v$ can't be in the kernel. Since $w$ has only one coming edge (v, w) and $v$ is not in the kernel, then $w$ must be in this kernel. However, since $u, w$ are in $K$ and there is an edge (w, u), condition (1) can't hold.
So, the assumption is not correct. And any kernel of G must contain some node x (distinct from $u, v, w$) that has an edge to at least one of the three nodes $u, v, w$.

c. First, this problem is in NP.

For $G = (V, E)$, we can easily verify if a subset K of V is the kernel of G in polynomial time.

Second, We can reduce 3SAT to this problem.

For a given 3SAT problem, without loss of generality, suppose there are n variables $x_1, x_2, ..., x_n$ and m disjunction terms $t_1, t_2, ..., t_m$.

- First, we create $2n$ nodes to represent all the literals: $n_{x_1}, n_{\bar{x_1}}, ..., n_{x_n}, n_{\bar{x_n}}$. And for all $i = 1, 2, ..., n$, we add edges $(n_{x_i}, n_{\bar{x_i}})$ and $(n_{\bar{x_i}}, n_{x_i})$ to the graph.
- Second, for each disjunction term, we create three new nodes $u, v, w$ to form a cycle $u \to v \to w \to u$. The three nodes represent the three literals forming this term. And then we create three edges from the respective literals to the nodes in this new circle. For example, $u, v, w$ represent literal $x_1, \bar{x_2}, x_3$, then we add edges $(n_{x_1}, u), (n_{\bar{x_2}}, v), (n_{x_3}, w)$ to the graph.

Note this reduction takes polynomial time.

We can prove that the kernel of the new graph contains exactly $n$ nodes from $n_{x_1}, n_{\bar{x_1}}, ..., n_{x_n}, n_{\bar{x_n}}$, and the $n$ literals is the solution of the given 3SAT problem (The kernel may also contain other nodes except from this n nodes).

- For every pair of nodes $n_{x_i}$ and $n_{\bar{x_i}}$, we have edges $(n_{x_i}, n_{\bar{x_i}})$, $(n_{\bar{x_i}}, n_{x_i})$ and there are no other edges entering the nodes $n_{x_i}$ and $n_{\bar{x_i}}$. From part a, we know that the kernel of the new graph must contain exactly one of the two nodes.
- As to the circles representing the disjunction terms, from the part b, we know that the kernel must contain some node x (distinct from the nodes forming that circle) that has an edge to at least one of the three nodes. As we can see, the "some" node x to this disjunction term is the value to satisfy this term.

To sum up, this problem is NP-complete.

## Problem 3

a. Suppose the optimal graph has cycles. Assume there's a circle $u \to v \to ... \to w \to u$ in the optimal graph, then we can remove one edge in the circle and still get all nodes connected. Since all distances are positive integers, the modified graph has less total distance than the previous optimal graph. In this case, the optimal graph with cycles is not optimal, which contradicts with the assumption. So, the optimal graph is a tree (every node has to be connected).

b. Given a number $a$, set $N = \{1, ..., n\}$ of n cities, a subset $M \subseteq N$ of mandatory cities, and the pairwise distances $d(i, j) > 0, 1 \leq i, j \leq n$ between the cities, which are assumed to be positive integers and symmetric. The problem is to find out if there is a connected graph H=(V,E) that includes all mandatory cities and which has total distance $d(H) = \Sigma\{d(i, j)|(i, j) \in E\}$ that is at most $a$?

c. Basically, we will use binary search to do solve this optimization problem.

Given input with size n, we can conclude that the maximum total distance will have no more than $n$ bits, that is, no more than $2^n$. Because the input contains all distances and bits represent edges, nodes. And the sum of the distances can't be a number that costs more than $n$ bits.

Then what we do is to first calculate the total distance $d$, then use the subroutine to check if $\frac{d}{2}$ satisfies. If $\frac{d}{2}$ returns 1, then check $frac\frac{d}{2} + d2$; if $\frac{d}{2}$ returns 0, then check $\frac{\frac{d}{2}+0}{2}$. And so on, until we find the optimal solution. This procedure takes $O(\log{(2^n)}) = O(n)$ times, and every time we call the subroutine which takes polynomial time. So in total, this algorithm takes polynomial time.

d. First, the decision version of the Steiner tree problem is in NP.

We can easily verify if a the spanning tree is at most $a$ in polynomial time.

Second, this problem is NP-hard. And we can reduce Node Cover to this problem.

Given Node Cover probelm with a graph G = (V,E), and number c. For every edge in E, we create a new node to represent this edge in G', and all the nodes form a set M. For every node in V, we create a new node to represent this node in G', and all the nodes form a set N. As a result, for the new graph G' = (V', E'), $V' = M \cup N$.

As to the edges of G',

- For $u, v \in M, d(u, v) = d(v, u) = \infty$.

- For $u, v \in N, d(u,v) = d(v,u) = \infty$.
- For $u \in M, v \in N$, if the edge represented by $u$ connects the node represented by $v$ in graph G, $d(u,v) = d(v,u) = 1$. If not, $d(u,v) = d(v,u) = \infty$.

Note that this reduction takes polynomial time.

In this case, graph G has a vertex cover that of size at most $c$ if and only if G' has a Steiner tree with total distance $d(H) = |M| + c - 1$.

To sum up, the decision version of the Steiner tree problem is NP-complete.

## Problem 4

a. In order to satisfy a DNF, we just need to satisfy at least one conjunction term. We can choose the first conjunction term of the DNF, if $x_i$ is in this term, set $x_i$ to be 1; if $\bar{x}_i$ is in this term, set $x_i$ to be 0. As to the rest, we can arbitrarily set the value, for example, all set to 0. Since the first conjunction term is satisfied, this DNF is satisfied.

In order to get the result, we at most need to scan entire the input (when there is only one conjunction term in the DNF). So, the Satisfiability problem for Boolean formulas in DNF is in P.

b. The complementary problem of DNF-TAUT is CP $= \{$DNF formula $F \mid \exists$ assignment $x, F(x) = 0\}$. We will prove that DNF-TAUT is coNP-complete by proving CP is NP-complete.

First, CP is in NP, because we can verify its result in polynomial time.

Second, We can reduce SAT to CP. For any given SAT problem, finding a satisfiable assignment is equivalent to finding an unsatisfiable assignment of its negation. And we can transform the negation of a CNF into a DNF in polynomial time. All we need to do is to change all $\wedge$ into $\vee$, change $\vee$ into $\wedge$ and replace literal $x$ with its counterpart $\bar{x}$.

To sum up, CP is NP-complete. Since DNF-TAUT is the complementary problem of CP, then DNF-TAUT is coNP-complete.

## Problem 5

a. Optimality Testing problem for $\Pi_1$ is in NP.

To solve this problem, all we need to do is to guess the a $y_{opt}$ that is the optimal solution and check if this $y_{opt}$ is the same as y. Since $|y| \leq p(|x|)$, we can this problem is in NP.

Optimality Testing problem for $\Pi_1$ is in coNP.

For $\Pi_1$, the complementary problem is y not an optimal solution for x. For that problem, we just guess the optimal solution $y_{opt}$, and check if $y_{opt}$ and $y$ are the same. Since $|y| \leq p(|x|)$, the complementary problem is in NP. And Optimality Testing problem for $\Pi_1$ is in coNP.

To sum up, Optimality Testing problem for $\Pi_1$ is in $NP \cap coNP$.

As to $\Pi_2$, since $\Pi_1$ and $\Pi_2$ are dual problems, they share the same optimal solution. So, Optimality Testing problem for $\Pi_2$ is also in $NP \cap coNP$.
b. For $\Pi_1$, the decision version is: "For a given x, k, if there exists a y that is a solution of instance x of $\Pi_1$, $f_1(x, y) <= k$."

First, we will prove $\Pi_1$ is in NP.

We can guess the a solution $y_{opt}$ that has the minimum value of $f_1(x, y)$ for all y that is the solution of instance x of $\Pi_1$. Then check if $f_1(x, y_{opt}) <= k$, if it is, return 1. Else return 0. As we can see, this problem is in NP.

Second, we will prove $\Pi_1$ is in coNP. We will prove this by showing its complementary problem is in NP.

The complementary problem is:" For a given x, k, if all y which is a solution of instance x of $\Pi_1$ that $f_1(x, y) > k$."

To solve this problem, We can guess the solution $y_{opt}$ that has the minimum value of $f_1(x, y)$ for all y that is the solution of instance x of $\Pi_1$. Then check if $f_1(x, y_{opt}) > k$, if it is, then return 1. Else return 0. As we can see, this complementary problem is in NP. So this problem is in coNP.

To sum up, $\Pi_1$ is in $NP \cap coNP$.

For $\Pi_2$, the decision version is: "For a given x, k, if there exists a y that is a solution of instance x of $\Pi_1$, $f_1(x, y) >= k$."

First, we will prove $\Pi_2$ is in NP.

We can guess the a solution $y_{opt}$ that has the minimum value of $f_2(x, y)$ for all y that is the solution of instance x of $\Pi_2$. Then check if $f_2(x, y_{opt}) >= k$, if it is, return 1. Else return 0. As we can see, this problem is in NP.

Second, we will prove $\Pi_2$ is in coNP. We will prove this by showing its complementary problem is in NP.

The complementary problem is:" For a given x, k, if all y which is a solution of instance x of $\Pi_2$ that $f_1(x, y) < k$."

To solve this problem, We can guess the solution $y_{opt}$ that has the maximum value of $f_2(x, y)$ for all y that is the solution of instance x of $\Pi_2$. Then check if $f_2(x, y_{opt}) > k$, if it is, then return 1. Else return 0. As we can see, this complementary problem is in NP. So this problem is in coNP.

To sum up, $\Pi_2$ is in $NP \cap coNP$.

So the decision version of both optimization problems is in $NP \cap coNP$.