Atanas Dimitrov
Homework #6
CS 6470
Dr. Cai

Solution to problem 1:

If $G_1(V_1, E_1)$ is isomorphic to $G_2(V_2, E_2)$ then for every tuple vertices from G_1, (v_1, u_1) $v_1,u_1 \in V_1$, which comprise an edge $e_1 \in E_1$ there is a corresponding tuple (v_2, u_2) $v_2,u_2 \in V_2$, in G_2 which comprise an edge $e_2 \in E_2$ and vice versa. We can look at this relationship more formally as a one-to-one mapping ∂: $V_1 \rightarrow V_2$, where $(v_1, v_2) \in E_1 \Leftrightarrow (\partial(v_1), \partial(v_2)) \in E_2$. Assuming fixed ordering of the vertices in G_1, $V_1 = v_1, v_2, \ldots, v_n$ we can use $(u_1, u_2, \ldots, u_n)$ in G_2 as a certificate to check if the property of isomorphism holds for the graphs G_1 and G_2. Then a simple algorithm can check: if for any tuple (v_i, v_j) there is an edge in E_1 if and only if there is an edge between vertices (u_i, u_j) in E_2. This can be done with two nested for loops (one for i and one for j) which will obviously complete in O(n^2). This time is obviously polynomial. Prior to the algorithm we can make sure hey we have the same number of vertices to reduce the running time in some cases. This can be done in linear time and will not affect the above time complexity.

Solution to Problem 2:

Using the definition for polynomial time reducible languages we can observe the following:
1. Since $L_1 \leq_p L_2$, then $\exists$ $f:\{0,1\}^* \rightarrow \{0, 1\}^*$ such that $\forall x \in \{0, 1\}^*$, $x \in L_1 \Leftrightarrow f(x) \in L_2$.
2. Since $L_2 \leq_p L_3$, then $\exists$ $g:\{0,1\}^* \rightarrow \{0, 1\}^*$ such that $\forall y \in \{0, 1\}^*$, $y \in L_2 \Leftrightarrow g(y) \in L_3$.
We can now substitute f(x) for y in 2. since the values for f(x) are at least a subset of L_2. Then we get $x \in L_1 \Leftrightarrow f(x) \in L_2 \Leftrightarrow g(f(x)) \in L_3$. Following the transitivity of logical operator $\Leftrightarrow$ (easily proven with truth tables) we get that $x \in L_1 \Leftrightarrow g(f(x)) \in L_3$. Now we have to prove that g(f(x)) is computed in polynomial time which is pretty obvious from the fact that f takes polynomial time and g does also, thus $g \circ f$ takes at least polynomial time.

Solution to Problem 3:

The professor has misunderstood the dynamics of this proof. Although it is aimed at providing us with proof that circuit-satisfiability problem is NP-hard, it doesn't consider any circuit in particular for deducing the proof. The values which the professor has pointed out as being vital and essential for the correctness of the proof, are simply irrelevant to the proof as each of them will be determined whenever a specific instance of a circuit is described. We construct a specific algorithm A, we determine value for k and the size n whenever we are taking under consideration a particular circuit C=f(x) The proof thus is written abstract from the explicit values and structure of A, k, and the constant factor implicit in O(n^k). In addition those variable components do not relate to the running time of the function F. Actually the professor has attempted to undermine the best part of this proof: its flexibility.

Solution to Problem 4:

Polynomial time verification:

Given A, x , and b we can easy deduce if the inequality holds in polynomial time. This can be seen since to multiply A and x will take O(mxn). Comparison with b is O(m). Thus the verification can be done in polynomial time. Thus, the problem is in NP.

Now we have to show that the **0-1 integer programming problem can be reduced from 3-CNF-SAT problem**. Thus we will have to prove that 3-CNF-SAT <=p 0-1 integer-programming problem. Since the 3-CNF-SAT problem is NP-complete then this will subsequently imply that so is the 0-1 integer-programming problem. Let's assume we have k clauses involving n variables. Let each clause contain 3 variables. Then we can replace each variable with its integer value {0, 1}, the negation of each variable with 1-the value of the variable, and each V sign with a addition. We'd like to test whether the final result is ≥ 1. Choosing this setup requires the right side of every inequality to be grater than or equal to 1 because each inequality represents a clause in the 3-SAT. Thus having in mind the property of logical operator AND each clause must evaluate to 1 so that the final result is also 1(or TRUE). Whenever the final result is 1 then the 3-SAT is satisfiable. replacing the variable by 1-the variable will accomplish the following: if the variable is 1 then we will get 0 if it is 0 we will get 1 thus this setup accomplishes successfully the transformation. As an example consider the following:

(x1 v (not)x2 v (not)x3)∧((not)x1 v (not)x2 v x3)

Obviously a possible solution is: x1=1, x2=1, x3=1

We transform the 3-CNF-SAT into:

$x1 + 1 - x2 + 1 - x3 \geq 1$

$1 - x1 + 1 - x2 + x3 \geq 1$

Which is equivalent to:

$x1 - x2 - x3 \geq -1$

$- x1 - x2 + x3 \geq -1$

We multiply both equations by -1. We get:

$-x1 + x2 + x3 \leq 1$

$x1 + x2 - x3 \leq 1$

The original matrix A is:

-1	*1*	*1*
1	*1*	*-1*

x is the matrix:

x1
x2
x3

b is the matrix:

1
1

Thus Ax≤b becomes:

-1	*1*	*1*	***X***	*x1*	**≤**	*1*
1	*1*	*-1*		*x2*		*1*
				x3		

Solution to the system is x1=1, x2=1, x3=1.
Each inequality can then be considered as a row in the matrix equation. This transformation can certainly be done in polynomial time. Thus the 0,1 integer-programming problem is NP-complete.

Solution to problem 5:

Let ∂ be a boolean formula composed of n boolean variables x_1, x_2, x_3, ... , x_n. For each x_i, i=1..n, consider x_i=1 then check the formula for satisfiability if it is satisfiable then retain x_i to have the value of 1. If not satisfiable then assign x_i to be 0 and perform the check again. If satisfiable then retain x_i to be 0. If no value is found for x_i which is satisfiable then the formula is unsatisfiable. Since the algorithm is run for every x_i in ∂, the checks will be executed at most n times. Each step takes polynomial time thus the overall time complexity will be O(n*{polynomial time}) which is polynomial time.