CS 154 - Introduction to Automata and Complexity Theory

Spring Quarter, 2009

Assignment #7 - Sample Solutions

Problem 1 **(a)** We can non-deterministically guess four different truth assignments, and check if the formula is satisfied for all these assignments. This takes linear time on an NDTM.
(b) We just add two dummy variables to the 3-SAT formula, but do not use them in any clauses. This is a $O(1)$ time operation.
(c) Note that adding one dummy variable doubles the number of satisfying assignments (the dummy variable could be set to 0 or 1 in each satisfying assignment of the original formula). Therefore, the 3-SAT formula is satisfiable if and only if the new formula with the dummy variables has 4 truth assignments.

Problem 2. In order to prove that dominating set is NP-Complete, we need to show two things: DS is in NP, and DS is NP-Hard.

A nondeterministic machine can decide dominating set as follows:

- Nondeterministically choose a vertex from the graph. Repeat this process a total of k times. Note that since we can choose a single vertex more than once, we account for "smaller than k" dominating sets as well. This can be done in about $k \log n$ steps; in any case it will be polynomial time. This will be our dominating set.

- For each vertex in the graph, check whether that vertex is in the dominating set or adjacent to a vertex in the dominating set. This will take (as a loose upper bound) n^3k steps, if for each vertex we check every edge in the graph to see if that edge connects the vertex to each node in the dominating set. If we ever find a vertex which is not in the dominating set and is not adjacent to the dominating set, we reject.

- At this point we've verified our dominating set. If a dominating set exists, there exists an execution path by which we "guess" the correct dominating set and accept. The overall algorithm runs in polynomial time, so DS is in NP.

In order to prove dominating set is NP-Hard, we will give a reduction of 3-SAT to DS (i.e. 3-SAT $<_{poly}$ DS). We claim that the following reduction works:

We input a 3-SAT formula $F(X_1, X_2, ..., X_n)$ with clauses $C_1, C_2, ..., C_m$. We produce a graph with $3n + m$ vertices. Three vertices correspond to each variable; these are labeled $x_i, \bar{x}_i, y_i$. One vertex corresponds to each clause; these are labeled c_i. The graph has $3n + 3m$ edges. The three vertices corresponding to each variable are connected by an edge (i.e. $x_i, \bar{x}_i, y_i$ form a triangle). Each clause vertex is connected to it's component terms (i.e. the clause $x_i + \bar{x}_j + x_k$ is connected by edges to the vertices $x_i, \bar{x}_j, x_k$. We claim that this graph has a dominating set of size n or smaller if and only if 3-SAT should accept. We need to prove both directions. We also need to show that this reduction runs in polynomial time, but that should be straightforward from the construction.

Assume 3-SAT should accept. Then there exists an assignment of true-false values to the variables such that all clauses are true. We will form a set S which includes x_i for all X_i which are true in this assignment, and $\bar{x_i}$ for all X_i which are false. This set contains exactly n vertices. All the vertices which came from variables will be covered by this set, since one of $x_i, \bar{x_i}$ was selected for every i. Since the truth assignment makes every clause true, one of the component terms of each clause is true and therefore in S. It follows that all the clause vertices are adjacent to vertices in S, and the set is a dominating set. Thus 3-SAT accepts implies DS accepts, and it follows that if DS rejects, 3-SAT should reject.

Suppose DS accepts. Then there exists a dominating set of size at most n. Call this set S. Since y_i is either in S or adjacent to a vertex of S for all i, and y_i is connected by edges only to x_i and $\bar{x_i}$, it follows that for every i, either x_i, $\bar{x_i}$, or y_i is in S. This already specifies n vertices, so exactly one vertex for each variable is included in S! We create a truth assignment as follows: X_i will be assigned true if x_i is in S. Otherwise X_i will be assigned false. Consider clause C_j. The vertex c_j was not in the dominating set (which only uses variable correlated vertices). So c_j is adjacent to some x_h or $\bar{x_i}$ in the dominating set. If c_j is adjacent to x_h in S, then since X_h is set to true it follows that c_j will be true. If c_j is adjacent to $\bar{x_i}$ in S, then x_i is not in S (remember that S uses exactly one vertex for each variable) and X_i will be false, so C_j will be true. It follows that this assignment is a solution for 3-SAT and 3-SAT will accept. Thus DS accepts implies 3-SAT should accept, and the reduction is correct.

Problem 3 (a) Given an instance of 2-SAT consisting of variables $X_1, \cdots, X_n$ and clauses $C_1, \cdots, C_m$, where $C_i = (X_{i_1} \cup X_{i_2})$ (variables may be negated). We propose the following reduction: construct an instance of 3-SAT consisting of variables $X_1, \cdots, X_n$, and Y and clauses $D_1, \cdots, D_m$, and E, where $D_i = (X_{i_1} \cup X_{i_2} \cup Y)$, and $E = (\bar{Y} \cup \bar{Y} \cup \bar{Y})$ (variables may be negated). This is polynomial time because the amount of work required beyond that of transcribing the input is: writing the new variable (constant); writing one more variable per clause (less than doubling the original time), and writing the new clause (constant time).

(b) Notice that to satisfy the last clause Y must be set to false. Therefore we may assume $Y = F$ in any satisfying assignment to the 3-SAT. Thus the Y can never satisfy any of the other clauses. A solution to 2-SAT is a solution to 3-SAT because the X_i that satisfies C_i will also satisfy D_i. Similarly, the reverse holds because every D_i must be satisfied by an X_i (since $Y = F$), and thus C_i will also be satisfied.

(c) No. We have shown that 3-SAT can be used to solve 2-SAT, and thus cannot be easier than 2-SAT. We already knew this, since 3-SAT is NP-complete and 2-SAT is in P.

Problem 4. The reduction works as follows, taking as input an instance of VC and producing as output an instance of ZIP. For each vertex $i \in V$, create a variable X_i in the ZIP instance in the output. For each edge $(i, j) \in E$, create the linear inequality

$$X_i + X_j \geq 1.$$

The value of t in the ZIP instance is the same as the value k in the VC instance.

It is clear that the reduction can be computed in polynomial time. To verify the validity of the reduction, we will show that the input graph $G(V, E)$ has a vertex cover of size k if and only if the ZIP instance in the output has a solution which satisfies all the inequalities and for which $\sum_{i=1}^{n} X_i = t$.

First, suppose that the graph $G(V, E)$ has a vertex cover $C \subseteq V$ of size k. For each X_i, set it to 1 if $i \in C$ and to 0 otherwise. Since each edge (i, j) in G has at least one of the two vertices i and

j in the vertex cover C, it follows that the corresponding inequality $X_i + X_j \geq 1$ will be satisfied, thereby ensuring that all inequalities are satisfied. Also, $\sum_{i=1}^{n} X_i = |C| \geq t = k$, thereby showing that the ZIP instance has a valid solution.

Conversely, suppose that the ZIP instance in the output has a solution which satisfies all the inequalities and for which $\sum_{i=1}^{n} X_i = t$. We will select a set $C \subseteq V$ as the set of all vertices $i \in V$ for which the corresponding variable X_i has the value 1 in the ZIP solution. It is clear that the size of C is exactly $t = k$. To verify that the set C is a vertex cover, observe that for each edge (i, j) the corresponding inequality $X_i + X_j \geq 1$ is satisfied, implying that at least one of i and j must be in C.

Problem 5 First we show that the problem is in NP. For this, we describe an NTM M that runs in polynomial-time. This machine M first guesses a $(1-1)$ mapping from the vertices of G_2 into the vertices of G_1, i.e., it finds a unqiue/distinct vertex in G_1 for each vertex in G_2. Then, deterministically and in linear-time, it verifies that an edge is present in G_2 if and only if the corresponding edge is present in G_1.

Now, to establish that SUBGRAPH ISOMORPHISM is NP-hard, we give a reduction from the INDEPENDENT SET problem already proved to be NP-hard in class. Given an instance, $G = (V, E)$ and k, of the INDEPENDENT SET problem, the reduction function f produces the following instance of SUBGRAPH ISOMORPHISM: Set $G_1 = G$, and G_2 as the empty graph on k vertices. It is clear that the reduction function f can be computed in linear-time by a DTM. Observe that if G contains an independent set of size k than that independent set will be isomorphic to G_2. Similarly, if G_2 is isomorphic to a subgraph of G, then G contains an independent set of size k. Thus, the reduction is correct, implying that SUBGRAPH ISOMORPHISM is NP-hard.