COMP531
Theory of Computation
Asignment 4

Ethan D.H. Kim (260163006)

April 10, 2006

KIM, Ethan Dong Hyun (ID: 260163006)

Problem 1. #SHORTEST-PATHS

(Here, we give an algorithm to compute the number of shortest paths in a digraph.) Given a directed graph $D = (V, A)$, and two vertices $s, t \in V$, compute $f(s, t)$, the number of shortest paths from s to t.

First, we define an auxiliary graph $D' = (V, A')$, where

$$A' = \{(u, v) \in A \mid u \text{ lies on a shortest path from } s \text{ to } v.\}$$

We can compute D' as follows. First, set $A' = \emptyset$. Then, for each arc $(u, v) \in A$, set $A' := A' \cup (u, v)$, if $d(s, v) = d(s, u) + w(u, v)$, where $d(s, v)$ is the shortest $s - v$ path distance and $w(u, v)$ is the weight on the arc. In other words, we are only inserting arcs that are on some shortest $s - v$ path, for all $v \in V$. Clearly, the graph D' is acyclic. Then, the total number of shortest $s - t$ paths can be computed via the following recurrence:

$$f(s, t) = \sum_{v \in \delta^-(t)} f(s, v)$$

where $\delta^-(t)$ denotes the set of all neighbouring vertices of t joined by incoming arcs. To see this, note that the graph D' is constructed in such a way that if there is an arc $(v, t) \in A'$, the set of all shortest $s - t$ paths contains all shortest $s - u$ paths. Thus, if we sum up the number of shortest $s - v$ paths for all parent v of t, we are counting the exact number of shortest $s - t$ paths.

To see the running time of this algorithm, note that the function $d : s \times V \to Z^+$ can be computed using dijkstra in $O(m \log n)$, and D' can be computed in $O(m + n)$ time. Finally, $f : s \times V \to Z^+$ takes $O(m + n)$ time to compute using BFS. Overall, the algorithm clearly takes poly time. Therefore, $\#SHORTEST - PATHS \in \mathsf{P}$.

Problem 2. #ISOMORPHISMS $\leq_p$ ISOMORPHISM

Proof. Let $\langle G_1, G_2 \rangle$ be an instance of $\#ISO$. First, simply run ISO on the input $\langle G_1, G_2 \rangle$. If they are *not* isomorphic, output 0 as your output for $\#ISO$. If they are isomorphic, however, we must do more work.

Suppose $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic. An isomorphism is a bijection $f : V_1 \to V_2$ such that $\forall (u, v) \in E_1$, $(f(u), f(v)) \in E_2$. In other words, an isomorphism is simply a permuation of V_1 that respects the above condition. Thus, a permutation f of the vertices V of $G = (V, E)$ is an automorphism, if $\forall (u, v) \in E$, $(f(u), f(v)) \in E$. Now, this implies that the number of isomorphisms from G_1 to G_2 equals the size of automorphism group $aut\ G_1$. (And, of course, $|aut\ G_1| = |aut\ G_2|$.) Hence, it suffices to show that we can compute $|aut\ G_1|$ by calling ISO polynomially many times.

As in one of the previous assignments, we use the labelling trick. Label each vertex with a number from 1 to n. Let G^k denote the graph constructed as follows. For each v_i, $1 \leq i \leq k$, attach a simple path of length i. Then, the automorphisms on G^k would have the first k vertices fixed, and only permuting the remaining $n - k$ vertices. Note, in general, an automorphism group forms an automorphism partition. For instance, if we consider all

automorphisms in G, a vertex v_i can be mapped to a subset $\delta(v_i) \subseteq V$, which forms an equivalence class.

Now, consider the two graphs G^{k-1} and G^k for $1 \leq k \leq n$. Let γ be an automorphism in $aut\ G^{k-1}$. Then, γ can be uniquely identified by two automorphisms α and β, where $\alpha \in auto\ G^k$, and β is an automorphism that maps some vertex in $\delta(v_k)$ to v_k. In other words, a permutation γ of the remaining $n-k+1$ vertices in G^{k-1} is defined by some permutation β that maps a vertex from $\delta(v_k)$ to v_k, and another permutation α of the remaining $n-k$ vertices in G^k. This implies the following recurrence:

$$|aut\ G^{k-1}| = |aut\ G^k| \times |\delta(v_k)|$$

Notice that $|aut\ G^n|$ is clearly 1, and what we wish to compute is $|aut\ G^0| = |aut\ G|$. Then, what's remaining is how to compute $|\delta(v_k)|$ for $1 \leq k \leq n$. Let G^{k-1} be the graph described as above. Let $G_1 = G^k$(that is, attach a simple path of length k to v_k). Then, take another copy of G^{k-1} and attach a path of length k to another remaining vertex $\neq v_k$, and call it G_2. Now we can check whether G_1 is isomorphic to G_2 by invoking ISO once. Repeat this check for all other remaining vertices without a path attached. Count the number of times that ISO accepted. Essentially, what we are doing here is to take a graph $G_1 = G^k$, and G_2(a graph resulting from removing a path attached at v_k and attach it to some other vertex without a path.) and check if they are isomorphic. This way, we can compute $|\delta(v_k)|$ by invoking ISO exactly $n-k$ times. There are n steps to compute $|aut\ G^0|$ from $|aut\ G^n|$, and thus overall $O(n^2)$ invocation of ISO. We now have computed the size of $aut\ G$ by invoking ISO polynomially many times, and this completes the proof. □

Problem 3. Communication Complexity

Show that the communication complexity is exactly the minimum depth of any circuit that computes f.

Proof. Let $D(f)$ denote the depth of the circuit that computes f, and $CC(X, Y)$ denote the communication complexity, where X(*resp.* Y) is the set of all possible inputs from Player 1 (*resp.* Player 2). First, assume that all NOT gates are located at the variables. (We can achieve this by applying De Morgan's Law without changing the depth of the circuit.)

Lemma 1. *For any boolean function f there exists a communication protocol such that $CC(X, Y) \leq D(f)$*

We first show that $CC(X, Y) \leq D(f)$ by constructing/exhibiting a protocol that computes f with $\leq D(f)$ rounds. Suppose the last gate before the output of f is an AND gate. Then, f can be divided into two parts, ie) $f = f_1 \wedge f_2$. Since Player 1 observed $f(x) = 0$, it must be the case that $f_1(x) = 0$ or $f_2(x) = 0$. If $f_1(x) = 0$, send the bit 0, and if $f_2(x) = 0$, send the bit 1 to Player 2. If both $f_1(x) = f_2(x) = 0$, send whatever. Since Player 2 observed that$f(y) = 1$, it must be that the subfunction f_i, where Player 1 observed 0, evalutes to 1 for Player 2. (ie. $f_i(y) = 1$) We now have a smaller function. If the next gate is again an AND gate, repeat the same process.

If it is an OR gate, $f = f_1 \vee f_2$, then Player 2 sends a bit: if $f_1(y) = 1$, Player 2 sends 0. If $f_2(y) = 1$, send 1.(Again, if both evalutes to 1, send whatever.) Similar to as before, since $f(x) = 0$, whatever subfunction f_i that Player 2 directs, Player 1 would have $f_i(x) = 0$. Again, we then have a smaller function. Repeat this process until we reach the leaf of the circuit, which is a variable, possibly with a NOT gate. Now we found the coordinate i of the input, in which the $x_i \neq y_i$. This process took exactly the depth of the circuit as the number of communications between the players.

Lemma 2. *For any communication protocol from the set $X, Y \subset \{0,1\}^n$, there exists a function f such that $D(f) \leq CC(X, Y)$.*

Conversely, we show that $D(f) \leq CC(X, Y)$, by constructing a function f. In particular, we use induction on $CC(X, Y)$. When $CC(X, Y) = 0$, there must exist a coordinate i in the input, such that $f = x_i$(since we cannot use any gates, the function is monomial.). Thus, $D(f) = 0$ for the base case. Now, assume that the inequality above holds for $CC(X, Y) = n - 1$. Let's look at when $CC(X, Y) = n$. Suppose that Player 1 sends the first bit in the protocol. Then, we can divide the input X into X_0 and X_1, such that $\forall x \in X_0$, Player 1 sends 0, and $\forall x \in X_1$, Player 1 sends 1. (Notice that X_0 and X_1 are mutually exclusive.) Then, for the rest of communication between the players, we have $CC(X_0, Y) \leq n - 1$ and $CC(X_1, Y) \leq n - 1$. By the inductive assumption, we know that there exist two functions f_0 and f_1, such that $D(f_0) \leq CC(X_0, Y)$ and $D(f_1) \leq CC(X_1, Y)$. Since Player 1 observed that $f(X) = 0$, and Player 2 observed that $f(Y) = 1$, we set $f = f_0 \wedge f_1$. Now we have that $D(f) \leq max\{D(f_0), D(f_1)\} + 1 \leq n$. Similarly, when Player 2 sends a bit, we can construct the function $f = f_0 \vee f_1$ and achieve the same lower bound on n.

□

Problem 4. 3-bit Negator

Here is the proposed solution for negating 3-bits. Let N_1, N_2 denote the two NOT gates used in the formula.

$$
\begin{aligned}
N_1 &= \neg((x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_2 \wedge x_3)) \\
N_2 &= \neg(((x_1 \vee x_2 \vee x_3) \wedge N_1) \vee (x_1 \wedge x_2 \wedge x_3)) \\
\neg x_1 &= ((x_2 \vee x_3 \vee N_2) \wedge N_1) \vee (x_2 \wedge x_3 \wedge N_2) \\
\neg x_2 &= ((x_1 \vee x_3 \vee N_2) \wedge N_1) \vee (x_1 \wedge x_3 \wedge N_2) \\
\neg x_2 &= ((x_1 \vee x_2 \vee N_2) \wedge N_1) \vee (x_1 \wedge x_2 \wedge N_2)
\end{aligned}
$$

To describe what the formula is doing, we look at what combinations of truth-values of x_1, x_2, x_3 induces true for $\neg x_1$. (the case for $\neg x_2, \neg x_3$ are symmetric.) Looking at the truth-table, there are three cases where $\neg x_1$ becomes true:

1. When all three $x_1 = x_2 = x_3 = 0$.
2. There are exactly one true-valued among $\{x_1, x_2, x_3\}$, and they are either $x_2 = 1$ or $x_3 = 1$.

3. There are exactly two true-valued among $\{x_1, x_2, x_3\}$, and they are $x_2 = 1$ and $x_3 = 1$.

With this insight, the formula for N_1 asserts "there are at most one true-valued among $\{x_1, x_2, x_3\}$". Then, using N_1 as part of the formula, N_2 asserts "there are no truth-valued input, or exactly two of $\{x_1, x_2, x_3\}$ are true". Then, let's look at the formula for $\neg x_1$. The first part $(x_2 \vee x_3 \vee N_2) \wedge N_1$ asserts the first two cases. Then, the part $x_2 \wedge x_3 \wedge N_2$ asserts the third case above. The truth-table for the formula is shown below.

x_1	x_2	x_3	N_1	N_2	$\neg x_1$	$\neg x_2$	$\neg x_3$
0	0	0	1	1	1	1	1
0	0	1	1	0	1	1	0
0	1	0	1	0	1	0	1
0	1	1	0	1	1	0	0
1	0	0	1	0	0	1	1
1	0	1	0	1	0	1	0
1	1	0	0	1	0	0	1
1	1	1	0	0	0	0	0

Problem 5. IP with perfect soundness $\subseteq$ NP

Proof. Let L be a language that has a single prover interactive proof system with perfect soundness. Note that, by definition of IP, the number of rounds and size of the messages between the prover and verifier is bounded above by a polynomial $p(n)$(since the running time of the verifier is polynomial.). Then we construct a nondeterministic polytime machine M as follows. First, M guesses the random string that the verifier would generate, and also the messages between the prover and verifier throughout the entire computation of the interactive proof system. The length of this guess, as stated above, is of polynomial size. Then, M simply goes through the guess (which is composed of the random string, and the sequence of communication between prover and verifier), and checks if they were valid communication. Suppose x was given as an input for M. If $x \in L$, there must exist a valid communication and random string that accepts x, by the definition of completeness. Thus, M can guess that communication and random string, and accept. If $x \notin L$, by the perfect soundness assumption, there does not exist a valid communication that will lead a proof system to accept. Thus, M cannot possibly guess such communication sequence, and so M rejects. □