CSC363H5 Winter 2016 Assignment 4

Name: Akhil Gupta
SN: 1000357071

Question #	Score
1	
2	
3	
4	
Total	

Acknowledgements:

"I declare that I have not used any outside help (excluding the textbook, the notes on the course website, the teaching assistants, and the instructor) in completing this assignment."

Name: Akhil Gupta Date: March 27, 2016

Q1. Let k be any fixed positive integer, and consider the following restricition of the Clique problem.

A graph $G = (V, E)$.
Decide if G contains a clique with **exactly** k vertices.
For any positive integer k, give a polynomial-time algorithm for the k-Clique problem. Briefly discuss why this problem is in P, while the regular Clique problem is NP-Complete.

Answer: By giving a polynomial time algorithm, we are effectively showing that this problem is in P. The polynomial-time algorithm for the k-Clique problem is as follows:

1. We construct a Turing Machine M which takes as input an encoding of a graph $\langle G \rangle$, where $G = (V, E)$.
2. The machine then checks each subgraph of G with k-vertices (k is a fixed constant) to see if that subgraph forms a clique. In order to check that the subgraph forms a clique, we have to ensure that an edge exists between all pairs of vertices in that subgraph.
3. Once we find such a subgraph with k-vertices, we have found a clique with exactly k vertices.

Now to show that this algorithm takes polynomial time: Suppose our graph $G = (V, E)$ has n vertices, then according to our algorithm, we have to check $\binom{n}{k}$ k-vertex subgraphs. For example, suppose our graph has 6 vertices and we want to decide if it has a clique with exactly 4 vertices, then we have to look at all $\binom{6}{4} = 15$ 4-vertex subgraphs. Hence, our algorithm (Turing Machine M) takes $O(n^k k^2)$ (where k is a fixed positive integer independent of the length of the input); we have $\binom{n}{k} = \frac{n^k}{k!} = O(n^k)$ subgraphs to check and each subgraph has $O(k^2)$ edges. Hence, k-Clique is solvable in polynomial time.

k-Clique problem for a fixed k is in P and regular Clique problem is NP-complete because we can reduce a NP-Complete problem like 3SAT to regular Clique whereas we can give a polynomial-time algorithm for k-Clique. Additionally k-Clique is in P while regular Clique is NP-complete because the input to both the problems differ. For k-clique, k is a fixed positive integer and is independent of the length of the input, so we can design a polynomial time Turing Machine. But, the input to regular Clique is some arbitrary k, due to this we cannot design a polynomial time Turing Machine.

Q2. We have the natural three-colouring problem and its search variant:
Three Colouring: A graph $G = (V, E)$. Decide if G has a three colouring.
Three Colouring Search: A graph $G = (V, E)$. Output a three-colouring of G or reject if no such colouring exists.
Give a search-to-decision reduction from Three Colouring Search to Three Colouring. (**Hint:** Observe that a triangle must be coloured with three different colours — try adding a triangle connecting it to the vertices of G somehow!)

Answer: We can do a search-to-decision reduction from Three Colouring Search to Three Colouring. Following the definition of search-to-decision reductions, we need to give a polynomial-time algorithm computing Three Colouring Search using an oracle for Three Colouring.
Algorithm for Three Colouring Search:

1. On input graph $G = (V, E)$
2. Query oracle to see if G contains a Three Colouring. If not, reject.
3. Assume that G has a Three Colouring. Then construct a new graph $G' = (V', E')$ by adding new vertices r, b, g to G and connect the edges between them i.e. rb, bg, gr to form a triangle. We can see that G' is three colourable, and in any three colouring of G', the three new vertices must receive a colour $c(v) \in \{\text{red}, \text{blue}, \text{green}\}$
4. For each vertex $v \in V$
   (a) In order to colour v, we will add edges to G' between v to exactly 2 vertices of the triangle $\{r, b, g\}$.
   (b) Add edges vr and vg to G'. If G' is three colourable, then v must be blue since v is adjacent to r (red) and g (green)
   (c) Query the oracle to see if G' contains a Three Colouring. If so, set $G = G'$. Otherwise, delete the previously added edges (vr, vg) and add a different pair of edges: vg and vb (in this case, v must be red). Go to step (c). If so, set $G = G'$. Otherwise, delete the previously added edges (vg, vb) and add a different pair of edges: vb and vr. Therefore, v must be green. Go to step (c).
5. Output G'

The algorithm clearly runs in polynomial time since it does a constant number of loop throughs through all of the vertices of the graph of a sequence of polynomial time computations. That is, for each vertex, it checks that the edges from that vertex to 2 other vertices have different colours by trying 3 pairs of edges. Additionally, the calls to the oracle take $O(1)$ time. Therefore, it does a constant amount of work. Let's prove correctness: First, if G does not contain a Three Colouring then the algorithm rejects immediately, so assume that G contains a Three Colouring. We claim that the output graph G' by the algorithm has a Three Colouring in G. Clearly G' is a subgraph of G, and by definition the algorithm adds exactly those edges which would create a Three Colouring in G such that every pair of vertices connected by an edge has different colours. It follows that G' is a Three Colouring of G.

Q3. In this problem we give a search-to-decision reduction from L_{SEARCH} to L, using L_{EXT} as an intermediate step.

1. Show that L_{EXT} is in NP.
2. Give a polynomial-time algorithm computing L_{SEARCH} that uses an oracle for L_{EXT}.
3. Using (a), and the fact that L is NP-Complete, modify your algorithm from (b) to give a search-to-decision reduction from L_{SEARCH} to L.

Answer 1. To show that $L_{\text{EXT}} \in NP$, we have to give a verifier that runs in polynomial time in the length of its input.
Verifier for L_{EXT}:

1. On input $\langle x, y \rangle$, and z (the certificate)
2. Check that z encodes an extension of y with $|z| \geq y$. If not, reject.
3. Check that $z = yw$ for all strings $w \in \{0,1\}^*$ where $|w| \leq |z| - |y|$. If not, reject.
4. Accept (x, z) if there exists some w such that $z = yw$

Clearly this verifier runs in polynomial time in the size of its input (x, y) - it takes polynomial time to check that z encodes a extension of y, and since the length of x, y, z, w are all finite, checking that $z = yw$ for some w takes polynomial time. Morover, if z is an extension of y, then there certainly exists a z that makes the verifier accept (just choose the z encoding the extension). Conversely, if the verifier accepts (x, z) then the string z is an extension of y by definition. It follows that $L_{\text{EXT}} \in NP$.

Answer 2. Polynomial-time algorithm computing L_{SEARCH} using an oracle for L_{EXT}.

1. On input $x \in \{0,1\}^*$
2. Query the oracle for L_{EXT} to see if there is a string y of length at most $c|x|^d$ where c, d are positive integers such that the verifier V accepts (x, y). If not, reject.
3. Otherwise, set $y = \epsilon$ where y is some string $\in \{0,1\}^*$.
4. While $(x, y) \notin V$ (let $\oplus$ represent string concatenation)
   (a) Query the oracle L_{EXT} on $(x, y \oplus 0)$
   (b) If the oracle returns yes then set $y = y \oplus 0$, otherwise set $y = y \oplus 1$.
5. Output y

The algorithm runs in polynomial time, since it consists of a single loop through the string y with multiple string concatenations of y with 0 or 1 of a sequence of polynomial time computations. Additionally, the calls to the oracle take $O(1)$ time. So the total computation time of the algorithm is polynomial. Let's prove correctness: First, if x does not contain a solution, then the algorithm rejects immediately, so assume that x does have a solution where there is a string y such that V accepts (x, y). We iterate over the length of y while maintaining the invariant that there is an extension y such that there is a valid solution to x. Since we proved in the earlier step that L_{EXT} is in NP, the solution to x must be polynomial in the length of x, and so the loop ends after a polynomial number of calls to the oracle, and hence returns a valid solution at the end.

Answer 3. In Answer 2. we gave a poly-time algorithm computing L_{SEARCH} that uses an oracle for L_{EXT}. Now, we have to give a search-to-decision reduction from L_{SEARCH} to L. So we have to construct an algorithm M computing L_{SEARCH} which has an access to oracle for L.

1. On input $x \in \{0,1\}^*$
2. Query the oracle for L to see if there is a string y of length at most $c|x|^d$ where c, d are positive integers such that the verifier V accepts (x, y). If not, reject.
3. Otherwise, set $y = \epsilon$ where y is some string $\in \{0,1\}^*$.
4. While $(x, y) \notin V$ (let $\oplus$ represent string concatenation)
   (a) Query the oracle L on $(x, y \oplus 0)$
   (b) If the oracle returns yes then set $y = y \oplus 0$, otherwise set $y = y \oplus 1$.
5. Output y

BONUS Let $G = (V, E)$ be a graph. Recall that a *dominating set* in G is a subset of vertices $V' \subseteq V$ such that for every vertex $v \in V$ either $v \in V'$ or some neighbour of v is in V'. In Tutorial 6 we proved that the deciding if a graph contains a small dominating set is NP-Complete.
Dominating Set:
A graph $G = (V, E)$, a positive integer k.
Decide if G contains a dominating set with at most k vertices.

Consider the following optimization variant of the Dominating Set problem.
Minimum Dominating Set:
A graph $G = (V, E)$ Output the smallest set of vertices $V' \subseteq V$ such that V' is a dominating set of G.
Give a search-to-decision reduction from Minimum Dominating Set to Dominating Set.

Answer: We can do a search-to-decision reduction from Minimum Dominating Set to Dominating Set. Following the definition of search-to-decision reductions, we give a polynomial-time algorithm computing Minimum Dominating Set using an oracle for Dominating Set.
Algorithm for Minimum Dominating Set:

1. On input G, where $G = (V, E)$ is an undirected graph
2. For $k = 0, 1, \ldots, n$
   (a) Query the Dominating Set oracle on $\langle G, k \rangle$. If the oracle returns no, go to (b)
   (b) Query the oracle until it returns yes for some k.
3. Using the obtained k value from step $2(b)$, we have to output the set of k vertices $V' \subseteq V$ such that V' is a dominating set of G.
4. Start with an empty set of vertices S. While $k - |S| > 1$:
   (a) Choose a vertex v that has not been marked and construct G'. G' will contain removing the vertex v from G, all of $v's$ neighbors, and all the edges incident to these removed vertices.
   (b) Query oracle on $\langle G', k - |S| - 1 \rangle$. If the oracle answers yes, add v to S and set $G = G'$. Otherwise, mark v.
   (c) Add the vertex with highest degree in G to S.
   (d) Output S.

We can see that this algorithm is polynomial since the operation (constructing the subgraph that is a result of removing nodes) can be done in polynomial time and is done at most once for every vertex. This algorithm is certainly polynomial since querying the oracle is bounded by $|V|$ since every graph has an dominating set $W \subseteq V$ and then the algorithm for the minimum k is also polynomial as described.