CIS 675 (Fall 2018) Disclosure Sheet

Name: Wentan Bai
HW # 7

(**Yes**) **No** Did you consult with anyone on parts of this assignment, including other students, TAs, or the instructor?

Yes (**No**) Did you consult an outside source (such as an Internet forum or a book other than the course textbook) on parts of this assignment?

If you answered **Yes** to one or more questions, please give the details here:

I also consult all questions and extra question with another student, Wentian Bai. For all questions, we discussed our ideas and I finish my algorithms independently.

By submitting this sheet through my Blackboard account, I assert that the information on this sheet is true.

Question 1:

Let the algorithm that solves Hitting Set problem be called hitting set algorithm.

Reducing the Vertex Cover problem to Hitting Set problem

- f **:** For each $edge(u, v)$ in the graph of Vertex Cover problem, create a set which only has two elements $\{u, v\}$. Let the value b equal to given value c.
- h **:** Return the set that is given by hitting set algorithm.

4 Questions :

1. Is f in polynomial time?
   - Yes, all we doing is creating a set for each edge, whose time is $O(|E|)$.
2. Is h in polynomial time?
   - Yes, h only returns a set.
3. If hitting set algorithm finds a set, does this set satisfy the requirements of Vertex Cover problem?
   - Yes. First hitting set algorithm will give a set with size at most c. Second, the hitting set contains at least one element from every set S_i which represents the two endpoints of every edges. So, if there exists a hitting set, the elements in hitting set must connects with all edges and every edge must have at least one of its two endpoints included in the set of c nodes.
4. If hitting set algorithm cannot give a solution, does it mean there is actually no solution for Vertex Cover problem?
   - Yes. Since hitting set algorithm constrains the size of hitting set at most c and checks every edge of graph, there is no solution if hitting set algorithm cannot find one.

Show that Hitting Set is NP-Complete

- Vertex Cover is known to be NP-Complete.
- Vertex Cover problem is able to reduce to Hitting Set problem.
- The solution given by hitting set algorithm can be verified in polynomial time. If we get a hitting set, then we can iterate over all sets and check whether for each set there is at least one element in hitting set. This process is in polynomial time.

Therefore, Hitting Set is NP-Complete.

Question 2:

Let the algorithm that solves g-Clause SAT problem be called g-Clause SAT algorithm.

Reducing SAT problem to g-Clause SAT problem

- f **:** Set integer g to be the number of clauses in Set problem.
- h **:** Return the result that is given by g-Clause SAT algorithm.

4 Questions :

1. Is f in polynomial time?
   - Yes, all we doing is to assign g a value, whose time is $O(1)$.
2. Is h in polynomial time?
   - Yes, h only returns the assignment for each literal.
3. If g-Clause SAT algorithm finds an assignment, does this assignment satisfy the requirements of Sat problem?
   - Yes. We set g to be the number of clauses in SAT problem. If every clause is satisfied, then there will be g numbers of satisfied clauses. Then the whole Boolean formula is also satisfied so this assignment is the solution for SAT problem.
4. If g-Clause SAT algorithm cannot give a solution, does it mean there is actually no solution for SAT problem?
   - Yes. Since g is the number of clauses in SAT problem, if g-Clause SAT algorithm cannot give a solution, it means there is at least one clause in SAT problem that is not satisfied.

Show that g-Clause SAT is NP-Complete

- SAT problem is known to be NP-Complete.
- SAT problem is able to reduce to g-Clause SAT problem.
- The solution given by g-Clause SAT algorithm can be verified in polynomial time. If we get a truth assignment by algorithm, then we follow the assignment to assign truth values to pairwise literals. Then we check g numbers of clauses. This process is in polynomial time.

Therefore, g-Clause SAT is NP-Complete.

Question 3:

To prove HAMILTONIAN PATH is in NP-Complete, I will show HAMILTONIAN CYCLE problem is able to reduce to HAMILTONIAN PATH problem, and solution of HAMILTONIAN PATH problem can be verified in polynomail time.

Let the algorithm that solves HAMILTONIAN PATH problem be called HAMILTONIAN PATH algorithm.

Reducing HAMILTONIAN CYCLE problem to HAMILTONIAN PATH problem

- f **:** For every $edge(u, v)$ in the graph of HAMILTONIAN CYCLE problem, we temporarily remove $edge(u, v)$ and set s to u as well as t to v. We will run HAMILTONIAN PATH algorithm on the modified graph to try to find a path. If the HAMILTONIAN PATH algorithm does not find a suitable path, then we restore the $edge(u, v)$ and continue to check other edges.
- h **:** If HAMILTONIAN PATH algorithm find a path, then add $edge(s, t)$ to connect s and t and return the cycle.

4 Questions :

1. Is f in polynomial time?
   - Yes, all we doing is iterating $O(|E|)$ times over every edges and run HAMILTONIAN PATH algorithm.
2. Is h in polynomial time?
   - Yes, h add an edge and returns a cycle.
3. If HAMILTONIAN PATH algorithm finds a solution, does this solution satisfy the requirements of HAMILTONIAN CYCLE problem?
   - Yes. The HAMILTONIAN PATH algorithm give a path that touches every node exactly once. By add a $edge(s, t)$ to create a cycle, this cycle touches every node exactly once.
4. If HAMILTONIAN PATH algorithm cannot give a solution, does it mean there is actually no solution for HAMILTONIAN CYCLE problem?
   - Yes. If there is not a path that touches every node exactly once, then it means we must touch some nodes at least twice.

Show that HAMILTONIAN PATH is NP-Complete

- HAMILTONIAN CYCLE problem is known to be NP-Complete.
- HAMILTONIAN CYCLE problem is able to reduce to HAMILTONIAN PATH problem.
- We can follow the given path and check whether we touch every node exactly once. This process is in polynomial time.

Therefore, HAMILTONIAN PATH is NP-Complete.

Question 4:

1. Denoted coordinates of robot as (x, y). Using DFS on robot and try to make x and y closer to $(0, 0)$. When $x > 0$, then choose the way that makes x smaller. When $y > 0$, then choose the way that makes y smaller. When $x < 0$, then choose the way that makes x greater. When $y > 0$, then choose the way that makes y greater.

2. Each time, move PacMan to the token that is the nearest to PacMan. If there is a ghost on the nearest token or between PacMan and nearest token, then move to the second nearest token.

3. First choose the most frequent literal and set it to $True$ or $False$ to make the most clauses satisfy. Then choose another literal that can make the most remaining unsatisfied clauses satisfy by set this literal to $True$ or $False$. Repeat this step until no literal to choose.

4. Choose a node V_0 with the least number of edges but this node at least has one edge. Assign this node V_0 a color. Then assign another color to all nodes $V_1...V_n$ which are connected to V_0. Then check all nodes $V_p..V_q$ which are connected to $V_1...V_n$. For $V_p..V_q$, if some of them can be assign a color that we previously used, assign that color to them. Otherwise, assign a new color. Repeat this process until all nodes has a color.

Extra:

Part A

To prove that CLIQUE-3 is NP-Complete, we need to show CLIQUE is able to reduce to CLIQUE-3. Only if CLIQUE is able to reduce to CLIQUE-3, it means CLIQUE-3 is as hard as CLIQUE.

Part B

CLIQUE is NP-Complete and CLIQUE is not able to reduce to CLIQUE-3. In CLIQUE problem, we do not know how many degree each node has. If there are some 4-vertex cliques in graph, it is not able to reduce to CLIQUE-3 problem. It means CLIQUE-3 problem is easier than CLIQUE problem. Therefore CLIQUE-3 problem is not in NP-Complete.