CS 6363.003 Spring 2021 — Author. Yaokun Wu (yxw200031)

Homework 5 Problem 1

(a) Let (S, T) and (S', T') be two minimum (s, t)-cuts in G. Prove that $(S \cap S', T \cup T')$ and $(S \cup S', T \cap T')$ are also minimum (s, t)-cuts in G. *[Hint: Let f^* be a maximum (s, t)-flow in G. What have we learned about f^* and the edges crossing any minimum (s, t)-cut?]*

Let f^* be a maximum (s,t)-flow in G.

So that $\begin{cases} \sum_{v \in S} \sum_{w \in \bar{T}} c(v \to w) = f^* \\ \sum_{v \in S'} \sum_{w \in \bar{T}'} c(v \to w) = f^* \end{cases}$

$$\Rightarrow \sum_{v \in S} \sum_{w \in \bar{T}} c(v \to w) + \sum_{v \in S'} \sum_{w \in \bar{T}'} c(v \to w) = 2f^*$$

$$\Rightarrow \sum_{v \in S \cup S'} \sum_{w \in \bar{T} \cap \bar{T}'} c(v \to w) + \sum_{v \in S} \sum_{w \in \bar{T} \setminus (\bar{T} \cap \bar{T}')} c(v \to w) + \sum_{v \in S'} \sum_{w \in \bar{T}' \setminus (\bar{T} \cap \bar{T}')} c(v \to w) \quad \cdots (1)$$

Due to the conservation constraints, for each $w \in [(s \cup S') \setminus (S \cap S')] \cap S$

$$\sum_{v \in (S \cap S')} c(v \to w) + \sum_{v \in [(s \cup S') \setminus (S \cap S')] \cap S'} c(v \to w) = \sum_{v \in \bar{T} \cap \bar{T}'} c(w \to v)$$

Substitute the constraints into (1).

We have $2\left(\sum_{v \in S \cup S'} \sum_{w \in \bar{T} \cap \bar{T}'} c(v \to w)\right) = 2f^*$

$$\Rightarrow \sum_{v \in S \cup S'} \sum_{w \in \bar{T} \cap \bar{T}'} c(v \to w) = f^* \quad \cdots (2)$$

Since $\sum_{v \in S} \sum_{w \in \bar{T}} c(v \to w) + \sum_{v \in S'} \sum_{w \in \bar{T}'} c(v \to w) = 2f^*$ can be rearranged

and rewritten as $\sum_{v \in S \cup S'} \sum_{w \in \bar{T} \cap \bar{T}'} c(v \to w) + \sum_{v \in S \cap S'} \sum_{w \in \bar{T} \cup \bar{T}'} c(v \to w) = 2f^* \quad \cdots (3)$

based on (2), (3) $\Rightarrow \sum_{v \in S \cap S'} \sum_{w \in \bar{T} \cup \bar{T}'} c(v \to w) = f^*$.

(b) Describe and analyze an efficient algorithm to determine whether G contains a unique minimum (s, t)-cut. *[Hint: Use the claim from part (a). Modify the process for finding a minimum (s, t)-cut given a maximum (s, t)-flow.]*

① Given a maximum (s,t)-flow and the corresponding residual graph G_f.

② Starting from s and use dfs on G_f to retrieve vertices that are reachable from s marked as set S_1. Mark set $T_1 = V - S_1$

③ Retrieve all vertices from which t is reachable as set T_2. Mark set $S_2 = V - T_2$

The minimum cut is unique iff $S_1 = S_2$ and $T_1 = T_2$.

Proof. part ② is already a minimum cut.

In part ③, since s and t are in different set, (S_2, T_2) is a cut. And we can easily observe that every vertex $v \in S_2$, $w \in T_2$, $f(v \to w)$ if exists, must equal to $C(v \to w)$ because otherwise v would be included in T_2, which contradicts $v \in S_2$. So that (S_2, T_2) must be a minimum cut as well.

Time complexity: $O(V+E)$ if maximum flow is given because we only need to traverse entire G_f twice.

CS 6363.003 Spring 2021 Author. Yaokun Wu (yxw200031)

Homework 5 Problem 2

Describe and analyze an algorithm to determine whether a given student can graduate. The input to your algorithm is the list of m requirements (each specifying a subset of the n courses and the number of courses that must be taken from that subset) and the list of courses the student has taken.

I'll construct a graph G with four types of vertices.

1. Source vertex s'
2. a vertex c_i for each courses i.
3. a vertex r_i for each requirements
4. a target vertex t'.

There are three types of edges.

1. an edge s' → c_i with capacity 1 for each course.
2. an edge c_i → r_j with capacity 1 for each course i and requirement j such that course i is in the subset of the requirement j.
3. an edge r_j → t' with capacity of the number of courses in order to satisfy the requirements.

I want to calculate the maximum flow f^* from s' to t

and compare it with the total number of courses required by all requirements, N.

If $f^* = N$, then the given student can graduate.
If $f^* < N$, then he/she can not graduate.

Proof: Since each path from s' to r_i can only be exactly once, meaning that a course can only be used once to satisfy a requirement. So if all the requirments are met, the capacity from r_i to t are saturated.

Time complexity:

$O(VE)$ if we use Ford-Fulkerson.

$= O((n+m+2)\cdot(n+f(m)+m))$ where $f(m)$ is the sum of courses in each subset of requirement.

CS 6363.003 Spring 2021 Author. Yaokun Wu (yxw200031)

Homework 5 Problem 3

(a) Describe a polynomial-time algorithm to solve DNF-SAT.

Traverse the clause one by one. Once we found any clause that no both literal and its negation are in the clause. We found an solution to the DNF-SAT.

We can assign true to the original literal and false if its negation is in the clause.

(b) What is the error in the following argument that P = NP?

> Suppose we are given a boolean formula in conjunctive normal form with exactly three literals per clause, and we want to know if it is satisfiable. We can use the distributive law to construct an equivalent formula in disjunctive normal form. For example,
>
> $$(x \vee y \vee \bar{z}) \wedge (\bar{x} \vee \bar{y} \vee z) \Leftrightarrow (x \wedge \bar{y}) \vee (x \wedge z) \vee (y \wedge \bar{x}) \vee (y \wedge z) \vee (\bar{z} \wedge \bar{x}) \vee (\bar{z} \wedge \bar{y})$$
>
> Now we can use the algorithm from part (a) to determine, in polynomial time, whether the resulting DNF formula is satisfiable. We have just solved 3SAT in polynomial time. Since 3SAT is NP-hard, we must conclude that P = NP!

Because the input size has changed during the conversion. In the worse case from size n in CNF to 2^n in DNF. And the part(a) that run in polynomial time is with respect to input size in DNF, which is $O(2^n)$ so that solving DNF in polynomial-time still means exponential time to solve original CNF.

Or you might say that the reduction from original problem to DNF can not be done in polynomial time.

CS 6363.003 Spring 2021 Author. Yaokun Wu (yxw200031)
Homework 5 Problem 4

(a) ***Pebbling*** is a solitaire game played on an undirected graph G, where each vertex has zero or more ***pebbles***. A single ***pebbling move*** consists of removing two pebbles from any vertex v and adding one pebble to an arbitrary neighbor of v. (The vertex v must have at least two pebbles before the move.) The PEBBLEDESTRUCTION problem asks, given a graph $G = (V, E)$ and an initial pebble count $p(v)$ for each vertex v, whether there is a sequence of pebbling moves that removes all but one pebble from G.
Prove that PEBBLEDESTRUCTION is NP-hard.

We show that PEBBLEDESTRUCTION is NP-hard by a reduction from Hamiltonian path.

Given an arbitrary graph $G' = (V', E')$. Let pebble graph G be the exact copy of graph G'. Considering all possible settings such that one of the vertices v in G to have $p(v) = 2$ and the rest to have $p(v) = 1$.

We claim that at least one of the setting can lead to an solution iff G' has a Hamiltonian path.

The reduction requires only polynomial time because $G' = (V', E')$ is copied into $|V'|$ numbers of pebble graphs.

$\Rightarrow$ First, suppose G' has a Hamiltonian path with the path starting at vertex v'.

then, the setting of pebble graph with $p(v') = 2$ will have a solution because we can go long the Hamiltonian path and end up with one pebble in G.

$\Leftarrow$ On the other hand, if one of the setting has a solution. We can keep tracks of the solution path to find the hamiltonian path because for such setting, every vertex in G must be visited exactly once to have a solution.

(b) The StoneSolitaire problem asks, given an initial configuration of red and blue stones in an $n \times m$ grid, whether the puzzle can be solved. Prove that StoneSolitaire is NP-***complete***.

First, the problem is NP because given a solution, I can check whether the puzzle is solved in polynomial time.

Second, we show that the problem is NP-hard by reduction from SAT.

Let Φ be a boolean SAT formula with m variables and n clauses.

We can convert the formula into a puzzle in following ways.

① if the variable x_i is in jth clause, then place a blue stone at board[j][i]

② if the variable $\bar{x}_i$ is in jth clause, then place a red stone at board[j][i]

③ if the variable x_i nor $\bar{x}_i$ appears in jth clause, then leave board[j][i] blank.

We claim that the puzzle is solvable iff Φ is satisfiable.

The reduction requires polynomial time since we only need to go through the clauses exactly once.

$\Rightarrow$ First, if Φ is satisfiable, given a satisfying assignment to all variables. We will remove all the stones corresponding to any variable that are false in the assignment. By doing so, each column that representing a variable would contain only one color stone and each row will contain at least one variable. Therefore, the puzzle is solved.

$\Leftarrow$ On the other hand, if the puzzle is solvable, given a solved solution. We can correspondingly assign varible to true or false based on the corresponding column and the stone color on that column. And since each row has at least one stone, all the clauses will be evaluated to be true. Therefores the Φ is satisfied.

Since the problem is both NP and NP-hard, the problem is NP-Complete.