CSOR W4231: Analysis of Algorithms (sec. 001) - Problem Set #6

Hongmin Zhu (hz2637) - hz2637@columbia.edu

December 4, 2019

Problem 1

Algorithm:

For this problem, we know the maximum flow f of the graph G. Assume the maximum flow is got by $Ford - Fulkerson$ method, we say the final graph is G' (flow of each edge is final, we cannot add more flow to this graph). Then we can use f and G' to recreate a series of augmenting paths with the following steps:

(1) since the maximum flow f is the minimum cut of the graph G, we can find the minimum cut of G and divide the vertices into two parts S and T

(2) the minimum cut line goes through all the edges P between S and T. We know that the sum of capacities of edges from S to T equals to the maximum flow f, so we can remove all the edges in P that are from T to S because using these edges will only cut down the overall flow of the graph which means the augmenting paths we want will not include these edge

(3) now that P only contains edges from S to T and each edge's flow equals to its capacity. We can be sure that the augmenting paths contain these edges. Let's say an edge in P is (u, v) and it has capacity of c. We know that augmenting paths containing (u, v) must contain edges from *source* to u that have total flow larger or equals to c, and edges from v to *sink* that have total flow equals to c. We say all the paths from *source* to u that satisfy this requirement is $left$, and all the paths from v to *sink* is $right$

(4) we enumerate all paths in $left$ and $right$, and connect them with (u, v). Then we get all the augmenting paths that contain edge (u, v) (in this way, we get not necessarily augmenting paths that are chosen in one run of $Ford - Fulkerson$ method, but all the possible augmenting paths in the graph G. If only the augmenting paths chosen in one run are needed, we could just pick one edge from $left$ and one from $right$ and

form one augmenting path with (u, v) together)

we loop over all the edges in P, and repeat steps (3)-(4). Then we have all the augmenting paths along which the flow would increase

Pseudo-code:

```
getSeries ($G'$, $f$)
    1  run DFS on $G'$ from $source$ and add all the reachable vertices to $S$, all the unreachable vertices to $T$ (find the minimum cut)
    2  $P \leftarrow$ edges from $S$ to $T$ with the total capacity equals to $f$
    3  for edge $(u, v)$ in $P$
    5      $left \leftarrow$ paths from $source$ to $u$ with total flow larger or equals to $c(u, v)$
    6      $right \leftarrow$ paths from $v$ to $sink$ with total flow larger or equals to $c(u, v)$
    7      $result \leftarrow$ pick edges from $left$ and $right$ to form an augmenting path together with $(u, v)$
    8  return $result$
```

Analysis:

If we just output a series of augmenting paths that are chosen in one run of $Ford - Fulkerson$ method, we can say that the size of this series is at most $|E|$ since each augmenting path will at least have one edge whose flow equals to capacity and the total number of edges is $|E|$.

Problem 2

(a)

For this problem, we can take some ideas from $Minimum-Cost-Flow$ problem. We define all the capacities of edges to 1 and the demand of vertex t to 1 so that we know the flow sum of this shortest path should be less or equal to k as the flow is either 0 or 1. We use $x_{(u,v)}, 0 \leq x_{(u,v)} \leq 1$ to indicate whether edge (u, v) is used to form the shortest path. So our goal is to minimize $\sum_{(u,v)\in E} x_{(u,v)} * w(u, v) * f(u, v)$, we can formulate the LP program as follows:
$min \sum_{(u,v)\in E} x_{(u,v)} * w(u, v) * f(u, v)$
s.t.
$\sum_{(u,v)\in E} f(u, v) - \sum_{(v,u)\in E} f(v, u) = 0, \forall u \in N - \{s, t\}$
$\sum_{(u,t)\in E} f(u, t) - \sum_{(t,u)\in E} f(t, u) = 1$
$f(u, v) \leq capacity(u, v)$
$capacity(u, v) = 1, \forall (u, v) \in E$
$\sum_{(u,v)\in E} \leq f(u.v)k$
$0 \leq x_{(u,v)} \leq 1$

(b)

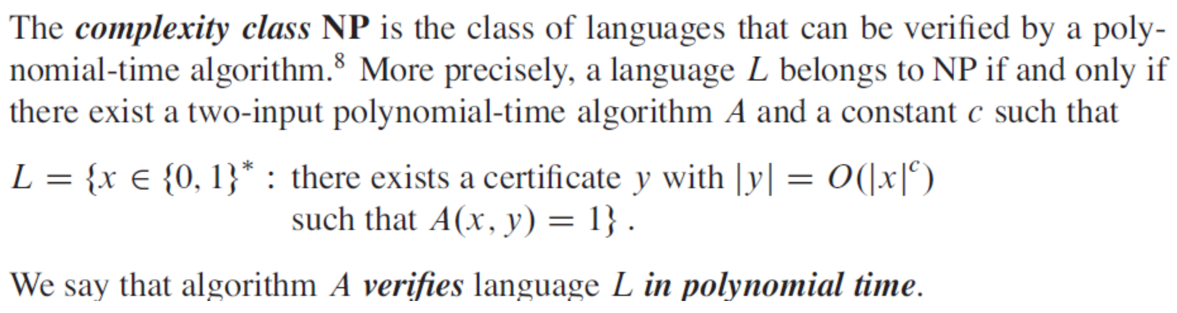
In problem (a), I say We use $x_{(u,v)}, 0 \leq x_{(u,v)} \leq 1$ to indicate whether edge (u, v) is used to form the shortest path. But $x_{(u,v)}$ is in fact from $\{0, 1\}$ which means this problem is actually an ILP problem, the shortest path is the optimal solution of this ILP problem, but we relax it to a LP problem. So now we have an optimal value of the obtained LP. Since the problem is minimization problem, we can know that the optimal value of LP is the minimum value of all the feasible solutions. And the optimal value of ILP problem is one of these feasible solutions, we can say that the optimal value of the obtained LP is upper bounded by the optimal value of ILP problem (the shortest path with at most k edges).

(c)

empty

Problem 3

According to the definition of NP class from the book:



The ***complexity class* NP** is the class of languages that can be verified by a polynomial-time algorithm.[8] More precisely, a language L belongs to NP if and only if there exist a two-input polynomial-time algorithm A and a constant c such that

$$L = \{x \in \{0,1\}^* : \text{there exists a certificate } y \text{ with } |y| = O(|x|^c) \text{ such that } A(x,y) = 1\} .$$

We say that algorithm A ***verifies*** language L ***in polynomial time***.

we know that for any decision problem in NP, we can design an algorithm that verifies the decision problem in polynomial time.
For example, for a decision problem L' in NP, we design an algorithm A':

```
ALGORITHM $A'$
1  for all possible certificate $y'$
2  if $A(x, y') == 1$
3      return 1
4  return 0
```

since the algorithm A' is a two-input polynomial-time algorithm, there are in total $2^{|y'|}$ numbers of possible certificates. And we know that $|y'| = O(|x|^c)$, c is some constant. So, the number is $2^{O(|x|^c)}$. For every possible y', the algorithm runs $A(x, y')$ in a polynomial time, let's say $O(|x|^{c'})$, c' is also some constant. Then we can get the total running time is $2^{O(|x|^c)} * O(|x|^{c'})$. In order to simplify the running time, we change it to $O(|x|^{c'})$ to $2^{log_2(|x|^{c'})}$, then the running time will be $2^{O(|x|^c) + log_2(|x|^{c'})} = 2^{O(|x|^c)}$, so we can say that the decision problem can be solved in time $2^{O(n^k)}$, where $k > 0$ is some constant.

Problem 4

problem $\in NP$. Given a solution x to the problem, we can verify $Ax \leq b$ in polynomial time (time to multiply A by x). Thus this problem is in NP.

problem $\in NPcomplete$. We know that for each row of the matrix A, we can formulate an inequality. So we can try to find a polynomial function to transform a CNF formula to this inequality. We can transform using the following rules:

(1) replace $\wedge$ to $+$
(2) replace $\neg x$ to $1 - x$
(3) for entries of A fill 1 for normal variable, -1 for negative variable and 0 for the rest

For example:
$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_3 \vee \neg x_4)$
$(x_1 \vee x_2 \vee x_3) \rightarrow (x_1 + x_2 + x_3) \geq 1 \rightarrow (-x_1 - x_2 - x_3) \leq -1$
$(x_1 \vee x_3 \vee \neg x_4) \rightarrow (x_1 + x_3 + 1 - x_4) \geq 1 \rightarrow (-x_1 - x_3 + x_4) \leq 0$
the matrix A and vector b are

$$\begin{bmatrix} -1 & -1 & -1 & 0 \\ -1 & 0 & -1 & 1 \end{bmatrix}$$

and

$$\begin{bmatrix} -1 \\ 0 \end{bmatrix}$$

if we want one $(-x_1 - x_2 - x_3) \leq -1$ to be true, at least one of x_1, x_2, x_3 should be 1 which also satisfies the clause $(x_1 \vee x_2 \vee x_3)$. So, we can say that if there is a solution to this 0-1 problem then there is a solution to the 3-SAT problem. We can then know 3-SAT $\leq$ 0-1 problem. So, 0-1 problem is proved to be in NP-complete.