Design and Analysis of Algorithms **Chen-Han, Tsai**
Homework1 985049600
October 2018

Question 1

Suppose I have an algorithm that can find the minimum vertex cover in $T(n)$ time, I can find the max-clique of a given graph $G = (V, E)$ by using such an algorithm:

Compute Max-Clique: Input graph G
1. Compute complementary graph of $G \rightarrow G*$
2. Run the minimum vertex cover algorithm over ($G*$ and receive a set of vertices V' (from the algorithm)
3. Return $\tilde{V} \leftarrow V/V'$ as the max-clique

Proof: The meaning of a minimum vertex cover is a set of vertices V' such that all the edges $e \in E$ are adjacent to one or more $v \in V'$. On the other hand, a max-clique is set of vertices $\tilde{V}$ such that all the vertices have edges connected with all the other edges. Note, that the minimum vertex cover of a max-clique of n vertices is any combination of $n-1$ vertices.

So by taking taking the complementary graph $G*$, we have removed the vertices of the max-clique to have any connection among themselves, and have them connect to vertices they were not connected to previously. The minimum vertex cover under $G*$ would now be the vertices not in the max-clique of G (V') since they would have the most connections to vertices in the max-clique. So by removing V' from V, we get a set of vertices that are not connected in any way in G', but are interconnected in G.

Computation Time: Suppose I was using a matrix of size (n, n) to denote the connections in G, then taking the complement would take time of $\theta(n^2)$. The minimum vertex cover algorithm takes time ($T(n)$. Removing is constant time, so in total : $O(n^2 + T(n))$.

Question 2

Given algorithm **ALG** that takes in a CNF function ϕ and a vector of corresponding variables $(x_1, ..., x_i, ...x_n)$ (possibly without set values), it outputs **TRUE** if ϕ is satisfiable using the vector parameters and **FALSE** otherwise.

The proposed algorithm **ALG'** is as follows:

ALG'$(\phi, (x_1, ..., x_n))$
1. If **ALG**$(\phi, (x_1, ..., x_n))$ returns **FALSE**: $\rightarrow$ return **FALSE**
2. For x_i from $i : 1 \rightarrow n$:
   a. Set $x_i = 0$
   b. If **ALG**$(\phi, (..., x_i = 0, ...))$ returns **FALSE** $\rightarrow$ set $x_i = 1$
3. Return vector $\bar{x} = (x_1, ...x_n)$ with newly set values

Explanation: Since we are given **ALG** that tells us if ϕ is satisfiable, we first run it on ϕ without any set values. If no satisfying assignment of $\bar{x}$ is possible to make ϕ return

TRUE, then there is no point wasting computation time and we stop. However, if it possible, then we start by setting the first variable x_1 as '0', and if $x_1 = 0$ causes ϕ to be unsatsifable (step 2.b of the algorithm), then we know that x_1 must be '1' to make ϕ satisfiable. Then we move to x_2 and do a similar computation; this time with x_1 set as the value of the previous step.

Computation Time: Step 1 of **ALG'** takes $T_{ALG}(n, m)$. If there is step 2, it takes time $T_{ALG}(n-1, m)+C$. This goes on for n cycles (actually, the last step can be done without **ALG** but we ignore this for now). In total, we spend a computation time of : $O(n + n \cdot T_{ALG}(n, m))$

Question 3

The factor 2 approximation for incident list representation of a connected graph $G = (V, E)$ is as follows:

Assumptions:
1) List/arrays starts with index '1'.
2) Every $v_i \in \bar{V}$ has a fixed array of size n.
3) $v_i[j]$ indicates a pointer to the vertex v_j adjacent to vertex v_i. If it has value NONE, then such a connection between v_j and v_i don't exist. $v_i[:]$ indicates the entire array of adjacent vertices of v_i

Factor 2 Approximation($V = \{1, ..., n\}$):

1. initialize $\tilde{m} \leftarrow 0, n_a \leftarrow 1, C \leftarrow \emptyset$
2. while $\tilde{m} < m$:
   a. set $v_a = V[n_a]$
   b. set $n_b = n_a + 1, v_b = v_a[n_b]$
      (if $v_b =$ NONE then increment n_b of $v_a[n_b]$ until not a NONE.)
      (if all are NONE, skip to 'e')
   c. for $i : 1 \rightarrow n$:
      i. if $v_a[i] \neq$ NONE : (value in array is not NONE)
         $\tilde{m} \leftarrow \tilde{m} + 1$
         set $pointer(v_a[i])[n_a]$ to NONE (access the pointer $v_a[i]$)
         set $v_a[i]$ to NONE
      ii. if $v_b[i] \neq$ NONE :
         $\tilde{m} \leftarrow \tilde{m} + 1$
         set $pointer(v_b[i])[n_b]$ to NONE
         set $v_b[i]$ to NONE
   d. $C \leftarrow C \cup \{v_a, v_b\}$
   e. $n_a \leftarrow n_a + 1$
3. return C

From step 2, we are bounded by m. At step 2.b, we can have at most n iterations before continuing. At step 2.c, we loop for n time again.

Hence, we have $O(m \cdot (n + n)) \rightarrow O(2mn)$.

Question 4

A) In the set L, we have a vertex v^* with the maximum amount of edges connected to R, and we denote the number of edges as d_L. v^* covers $\frac{d_L}{|R|}$ of the vertices in R, and that is the most any vertex in L can cover. That is why we need at least $\frac{|R|}{d_L}$ vertices in L to cover all the vertices in R ($\frac{d_L}{|R|} \cdot \frac{|R|}{d_L} = 1$). Therefore, the minimum size of the dominating set for R is $\frac{|R|}{d_L}$.

B) From a given minimum vertex cover(MVC) problem, we can create a vertex for each edge in the graph of MVC, and denote that set as R. And now, we put every vertex in the MVC graph into a set L. A vertex in L will be connected to a vertex (resembling an edge) in R if that resembling edge is connected to the vertex in the MVC problem.
At the end, each vertex in R will have only two edges to vertices in L since they resemble edges. By using the given algorithm, we can compute the minimum vertex cover in time of n.

C)
Dominating Set: given a bipartite graph $G = (V = (L, R), E)$

1. Initialize $L' \leftarrow \emptyset$, $R' \leftarrow R$, $M \leftarrow \emptyset$
2. while $R' \neq \emptyset$:
   a. choose a random vertex r_i from R'
   b. add to L' all left vertices l_i that shares an edge with r_i
   c. add to M vertex r_i (for proving purpose)
   d. for each vertex l_i that shares an edge with r_i, remove the any vertex adjacent to l_i
3. return L'

Claim: The Dominating Set algorithm has an approximation factor B $= d_R$

Proof:
For every cycle of the algorithm, we add into set $M \leftarrow r_i$.
For all l_i that is connected to r_i, at least one of them is part of OPT (since that is the purpose of OPT).
In step (d), since we removed all the vertex that shared an edge with r_i, no other r_j in M can be connected to previous l_i's.
Hence, $|M| \leq |OPT|$ (since there are no overlaps with other vertices r_j).
For each cycle, there is at most d_R vertices from L selected, so $|L'| \leq d_R \cdot |M|$.
$\rightarrow \frac{|L'|}{d_R} \leq |OPT|$
$\rightarrow |L'| \leq |OPT| \cdot d_R$