CS 255 – Homework 5

Team Members:
Ravali Koppaka (012445693)
Vincent Stowbunenko (005625269)

1. An **almost clique** is any graph that is the result of deleting one edge from a clique. Prove that the problem of whether a graph G has an almost clique of size t is *NP*-complete.

Solution:

A language L is called ***NP*-complete** if

1. $L \in NP$, *and*
2. $L' \leq_P L$ *for every* $L' \in NP$

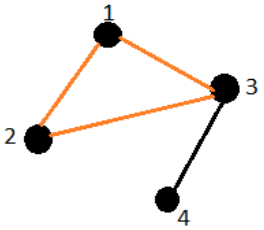
In NP

Given a graph G with 'n' vertices we can pick 't' vertices and check if they form an almost clique. We check the edges between every two vertices among the 't' chosen vertices and this verification algorithm runs in polynomial time $O(t^2)$

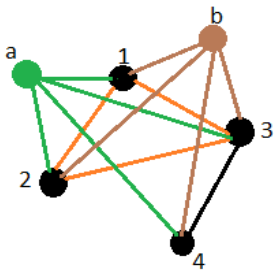
Reduction

'CLIQUE' to 'Almost CLIQUE'

From the given graph G construct, a graph G` by adding two vertices 'a' and 'b' and add edge from all the vertices in G to two vertices 'a' and 'b'. If graph G has a **clique** of size 'k' then graph G` will have **almost clique** of size 'k+2'. This reduction is in polynomial time O(n); n : number of vertices



Graph G



Graph G`

G has a clique of size 3: vertices 1,2,3

G' has almost clique of size 5: vertices 1,2,3, a, b

2. Show 0-1-2 integer programming is NP-complete where **0-1-2 integer programming** is the problem: Given a list of *m* linear inequalities with rational coefficients over *n* variables $u_1,\ldots u_n$ (i.e., *m* inequalities of the form $a_1u_1+a_2u_2+\cdots+a_nu_n \le b$ where the a_i and *b* are fraction *pq* for some integers *p* and *q*), decide if there is an assignment of the numbers 0, 1, or 2 to the variables that satisfies all the inequalities.

Solution:

In NP

Given a list of ***m*** linear inequalities over ***n*** variables, we can assign a value in {0,1,2} to each of the n variables and check if this assignment satisfies all the 'm' inequalities.

This verification algorithm runs in polynomial time O(m*n) – O(n) to find the sum of products of a_i and x_i and we have 'm' such inequalities

Reduction

This reduction is done in two steps

a) First, we reduce {0,1,2} integer programming to {0,1} integer programming problem

b) Now 3-SAT is reduced to {0,1} integer programming

   This way every {0,1,2} instance is represented as a {0,1} integer programming instance and to which a 3-SAT instance can be reduced to.

   Reduction from {0,1,2} to {0,1} integer programming
   Every variable x which takes values in {0,1,2} replace it with two variables x` and x`` such that x`, x`` take value in {0,1}

   Example:
   $x_1 + x_2 + x_3 >= 2$; where $x_1, x_2, x_3 \in \{0,1,2\}$

   $(x_1` + x_1``) + (x_2` + x_2``) + (x_3` + x_3``) >= 2$; $x_1`, x_1``, x_2`, x_2``, x_3`, x_3`` \in \{0,1\}$

   Possible values for $x_1`$; $x_1``$

$x_1`$	$x_1``$	$x1 = (x_1` + x_1``)$
0	0	0
0	1	1
1	0	1
1	1	2

   Reduction from 3-SAT to {0,1} integer programming
   Consider a 3-SAT instance with **m** clauses and **n** literals
   C1∧C2…. Cm each clause can be represented as an inequality

Example:
$C_1 = x_1 \text{ V } (\sim x_2) \text{ V } x_3$; '~' implies negation

$x_1 + (1 - x_2) + x_3 >= 1$; $x_1, x_2, x_3 \in \{0,1\}$

If x_i = 'T' then we assign a value $x_i = 1$
x_i = 'F' then we assign a value $x_i = 0$

If clause C1 is satisfied for some assignment of truth values of literals, then the corresponding inequality would also be satisfied.

So, if 3-SAT instance with '**m**' clauses and '**n**' literals is satisfiable than we have an assignment to '**n**' input variables that satisfies all the '**m**' inequalities. This reduction is in polynomial time O(m*n)

3. A neural net gate $NN(x_1,...,x_n,w_0,w_1,...,w_n)$ outputs 1 if $w_0+\sum_i w_i x_i>0$ and 0 otherwise. Here $x_i\rightarrow$ are viewed as the **inputs** and $w_i\!-\!\!\rightarrow$ are called **weights**. We imagine weights are fixed after some training process. A **neural network** is a directed acyclic graph where the nodes are labelled with *NN* gates. The output of such a network is computed in the natural way by evaluating gates which are immediately connected to the inputs, followed by gates all whose inputs now have values, and so on. Define the **neural network understanding (NNU) problem** to be given a neural network *N* and a setting for its weights $\vec{w}$, decide if there is a setting of its inputs $\vec{x}$ which makes it output 1. Show the *NNU* problem is *NP*-complete.

Solution:

In NP:
Given a neural network N, and a setting for its weights $\vec{w}$, we can assume some setting for $\vec{x}$ and check if neural network outputs '1'. This verification is done in polynomial time O(n*k) as it takes linear runtime to find the weighted sum; k implies the number of neural net gates(nodes) in the neural network

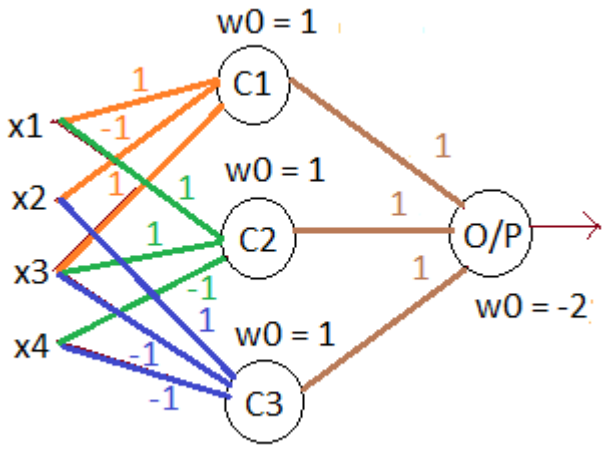
Reduction:
3-Sat to a neural network

3-Sat instance with 'm' clauses and 'n' literals
Each clause of the 3-Sat instance is represented as one neural net gate
C1∧C2∧C3
C1 = (x1) V (~x2) V (x3); ~ implies negation
C2 = (x1) V (x3) V (~x4)
C3 = (x2) V (~x3) V (~x4)

Neural Net gate for the clause is defined such that if the literal x_i in the clause is negated then we assign weight $w_i = -1$ else $w_i = 1$ and $w_0 = 1$

The output node will return 1 if all the clauses are satisfied.

Following is the neural network instance for the 3-Sat



NN gate (C1): w0 = 1
w1 = 1
w2 = -1; since x2 is negated
w3 = 1
$1 + x1 - x2 + x3 > 0$; if C1 is satisfied

Output Node(O/P): All the output from the clauses become input to this node;
$\forall i\ w_i = 1$; $w_0 = -(\text{number of input clauses} - 1)$
Weights are assigned such that the node will output '1' if all the clauses are satisfied.
The reduction is in polynomial time O(n*k) – k is the number of neural net gates(nodes) in the neural network