CS 344: Homework #5

Due on April 26, 2017 at 11:59pm

Professor Bahman Kalantari Section #1

Douglas Rudolph, Mustufa Hussain

Problem 1: Use the fact that 3-CNF is NP-complete to prove NP-completeness of testing if a linear system of inequalities $Ax \leq b$ has a solution, where A is an $m \times n$ matrix of integer coefficients, x is a vector with n components consisting of either 0 or 1, and b is a vector with m integer components. Here $Ax \leq b$ means the inequality must be satisfied component-wise.

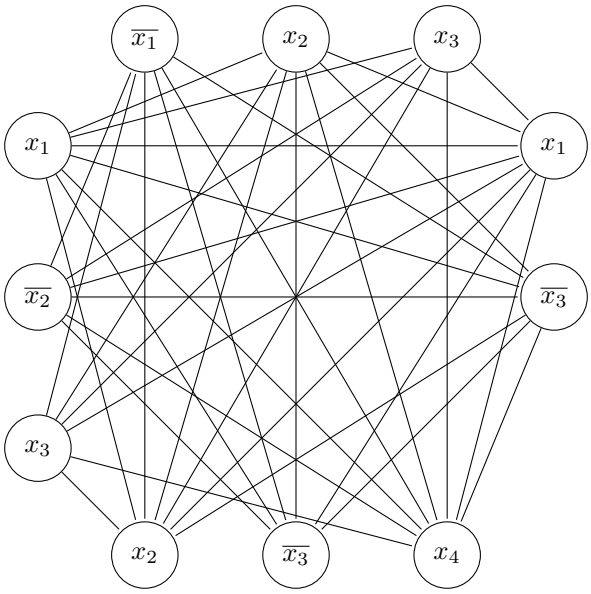
To show that this is the case, we have to show i) that the above problem is in NP ii) that 3-CNF can be reduced in polynomial time to the above problem.
Proof of i): To show that the above is NP, it suffices to provide a polynomial time algorithm that, given a solution with a certificate of verification, checks that the provided solution is valid. This can be accomplished in two steps:

1. Plug in the solution vector s into $Ax \leq b$
2. For each of the m constraints, verify that the inequality is satisfied

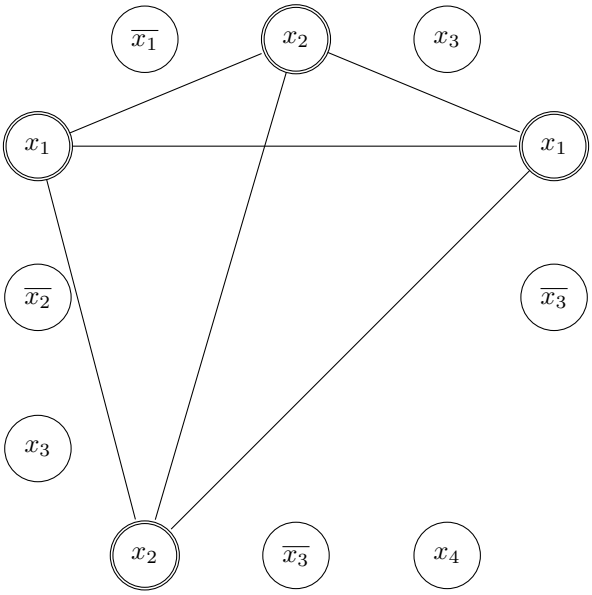
Clearly, this algorithm runs in polynomial time ($O(mn)$ to plug in the solution vector and $O(m)$ to check each constraint $\implies O(mn)$ algorithm) as desired.
Proof of ii): To prove this, we must describe how to convert a general 3-CNF problem into the form $Ax \leq b$. Given a 3-CNF of the form $(x \vee y \vee z) \wedge \cdots \wedge (a \vee b \vee c)$ where x,y,z,a,b, and c are boolean variables, we know that the expression will evaluate to true if each individual clause evaluates to true. In the case of our linear system, this is equivalent to having each constraint hold true. As such, we can think of each clause in 3-CNF as a linear constraint. Additionally, we know that a given 3-CNF clause will evaluate to true if at least one variable in the clause is true. If we represent true with 1 and 0 with false, this is equivalent to asking if the sum of the variables is greater than 1. This leads to a formula that models a clause a constraint: Asking if $(x \vee y \vee z)$ evaluates to true is equivalent to asking $x + y + z \geq 1$. It is also worth mentioning that if a given variable is a complement (i.e. of the form $\overline{x}$, we can represent it as $1 - x$ in our constraint (i.e. if it is false, it adds one to the boolean sum while it adds nothing if it is true). Finally, to make the inequalities of the form $Ax \leq b$, we can simply negate both sides of each constraint to convert each $\geq$ into a $\leq$. This gives us a linear system of the form $Ax \leq b$ where A is our $m \times n$ constraint matrix, where each row corresponds to a 3-CNF clause, x is a vector with n components, each of which is 0 or 1, and b is our vector of m constraints where each component is -1. Given n 3-CNF clauses, we can reduce it to the equivalent linear system with approximately 3*n operations, which corresponds to a polynomial time algorithm ($O(n)$) as desired.

Problem 2: Consider the CNF expression $(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_3}) \wedge (x_2 \vee \overline{x_3} \vee x_4)$. Convert this CNF into a graph so that G has a clique of size 4 iff the CNF is satisfiable. Does G have a clique of size 4?

To convert the above CNF expression into a graph, we create a vertex for each literal and put an edge between vertices that don't belong to the same clause or contradict each other:

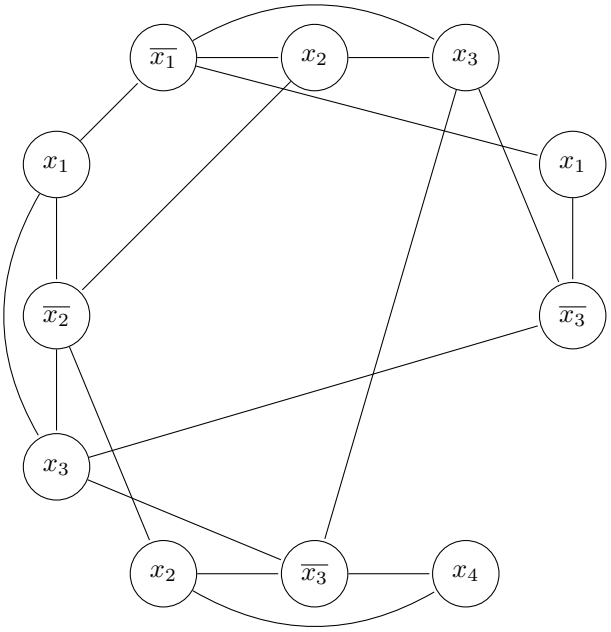


The graph does indeed have a clique of size 4 (given by setting x_1 and x_2 to true in the CNF problem); the nodes that form the clique are double circled and the edges which form the clique are included (all others are omitted for readability):



Problem 3: For the graph G = (V, E) above construct the complement graph $\overline{G} = (V, \overline{E})$. What is the largest clique in G? what is the largest vertex-cover in $\overline{G}$?

We construct the complement graph, $\overline{G}$:

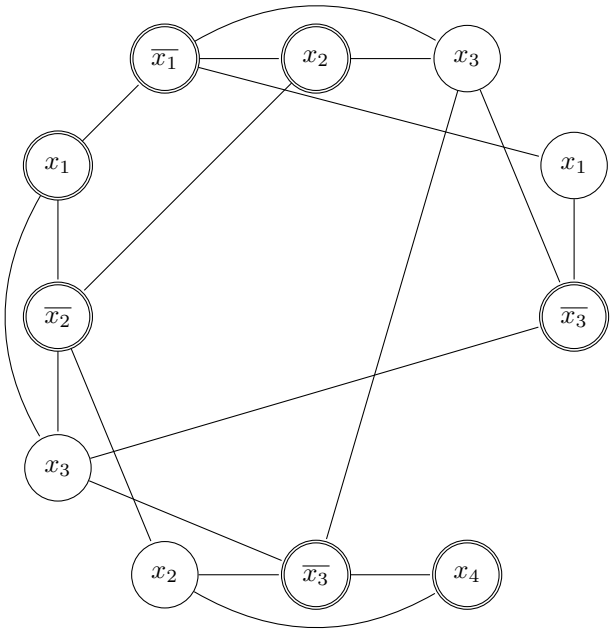


The largest clique in G is equivalent to the maximum size of sets of nonadjacent vertices in $\overline{G}$. These sets are as follows:

1. $\{x_1, x_4, x_2, x_3\}$
2. $\{\overline{x_3}, x_4, \overline{x_2}, \overline{x_1}\}$
3. $\{x_2, x_3, x_2, x_1\}$
4. $\{\overline{x_3}, \overline{x_2}, \overline{x_1}, \overline{x_3}\}$
5. $\{x_4, x_3, x_2, x_1\}$
6. $\{x_3, x_2, x_1, x_4\}$

As such, we conclude that the largest clique size is 4.

Because our largest clique is of size 4 and there are 11 vertices, we know that our vertex cover will be of size 7 (vertices in the vertex denoted by double circles):



We can find the above by starting with one of our sets (say $\{\overline{x_3}, \overline{x_2}, \overline{x_1}, \overline{x_3}\}$) and denoting adjacent edges. After this, we pick the vertices with leftover edges, namely $\{x_1, x_2, x_4\}$, giving us our vertex cover.

Problem 4: Suppose $G = (V, E)$ is a spanning tree. Describe an algorithm that would efficiently compute a minimum vertex-cover for G

Given an undirected graph G, we know that a spanning tree of the graph G is a subgraph that is made up of a collection of edges that connects all the verticies together. Because we know that this subgraph is made up of a collection of edges that manage to connect every vertex within the graph, then we know that the spanning tree is a filtered subset of the verticies that are apart a vertex cover.

In order to find the vertex cover, we can simply start by looking at the node with the highest degree within the spanning tree. Now that we have the node with the highest degree, we can mark that node as inside the vertex cover, and remove said node and all neighboring nodes from the set of verticies that are within the spanning tree. This process can then be repeated on the next node with the highest degree until there are no longer no more nodes within the set of the subgraph. The remainng marked nodes is the considered to be the vertex cover.