CMSC 641 Design & Analysis of Algorithms Spring 2023
Section 01 Homework 9

Faisal Rasheed Khan

VB02734

vb02734@umbc.edu

1 Finding a three coloring

This problem explores the difference between decision problems (where we only provide yes/no answers) and search problems (where we actually find a witness to some property).

In computational complexity theory, we often work with algorithms that have access to an oracle. The oracle is a set and the algorithm is allowed to make membership queries to the oracle. That is, the algorithm can ask an oracle A whether a string x a member of the set A. The algorithm receives the answer from the oracle in a single step (i.e., the complexity of determining whether x ∈ A is not included the running time of the algorithm). The oracle only provides a yes/no answer.

Consider the usual three coloring decision problem defined below.

Three Coloring (3COL)

INPUT: an undirected graph G = (V, E)

QUESTION: Does there exist an assignment of colors to each vertex v ∈ V such that no more than 3 colors are used and for every edge (u, v) ∈ E, u and v are assigned different colors.

Assignment:

1. Suppose we are given 3COL as an oracle. Devise and describe a polynomial-time algorithm that finds a three-coloring of an undirected graph G, or states that no three-coloring of G is possible. When a three-coloring is possible, your algorithm must assign a color to each vertex of G and guarantee that adjacent vertices are assigned different colors.
   Recall that the oracle will only return a yes/no answer and will not provide you any evidence as to why the graph is or is not three colorable.
   Hint: you will need to repeatedly use the oracle on graphs that you construct.

A. <u>Algorithm to find a three-coloring undirected graph G:</u>
   1. Given graph G(V,E) and 3 colors (let them be green, red, and blue).
   2. Choose any vertex v as a starting vertex.
   3. Assign a color to the starting vertex v and check with the oracle if the coloring satisfies the 3-coloring constraint.
   4. For each adjacent vertex u of v, check if u has the same color as v. If it does, assign a different color to u and check with the oracle if the coloring satisfies the 3-coloring constraint.
   5. We use Breadth First Search to keep track of adjacent vertices, i.e we use queue data structure and ensure for vertex u, for every edge (u, v) ∈ E, u and v are assigned different colors.
   6. If the oracle returns "no", then the graph cannot be colored with 3 colors. Exit the algorithm.
   7. Repeat step 4 for all adjacent vertices of v.

8. If all vertices are colored and the oracle returns "yes" for the final coloring, output the coloring as a solution to the 3-coloring problem.

2. Explain why your algorithm works correctly.

A. My Algorithm works correctly because each vertex is assigned a color and no two adjacent vertices have same color i.e. for every edge (u, v) ∈ E, u and v are assigned different colors. At each color assigned to the vertex, oracle checks if it is meeting the 3COL constraint and ensures that the Graph G(V,E) is 3-colored, oracle returns either yes/no. This process is repeated for all vertices until either the graph is colored with 3 colors and the oracle returns "yes", or it is determined that the graph cannot be colored with 3 colors and the algorithm exits. Therefore, the algorithm guarantees to either find a valid 3-coloring of the graph or determine that one does not exist.

3. Briefly state and justify the running time of your algorithm. (Calls to the oracle return in Θ(1) time.)

A. The Algorithm considers V (Vertices) iterations for coloring and E Edges to check the adjacent vertices such that for every edge (u, v) ∈ E, u and v are assigned different colors. The oracle returns in O(1) time. Running time cannot be O(V*E) because while considering the worst case scenario i.e. every vertex has v-1 edges then we will come to know that we cant assign 3-color to the graph and the oracle returns no and we do not proceed further with our algorithm i.e. it terminates.
Running Time = O(V+E) + O(1)
= O(V+E)

4. Argue that if 3COL can be decided in polynomial time, then your algorithm can be used to find a three-coloring in polynomial time.

A. 3COL is the verification of a graph coloring problem(which can be of any color problem) that it has 3COL solution i.e. it verifies one of the solution for the graph. The verification takes polynomial time as this has been reduced from a hard problem of Graph Coloring.
Our Algorithm just verifies the 3-color can be assigned to the vertices such that for every edge (u, v) ∈ E, u and v are assigned different colors.
If the oracle returns "no" at any point, we know that the graph cannot be colored with 3 colors and we terminate the algorithm. Otherwise, we continue iterating over all vertices until all vertices have been assigned a color.
Our Algorithm runs in a polynomial time as it is a decision problem of having 3-color or not, and our algorithm runs in a polynomial time which is O(V+E).
Therefore, if 3COL can be decided in polynomial time, our algorithm can be used to find a three-coloring in polynomial time.

2 All But Five 3-Colorable

Consider the Three Coloring problem again:

Three Coloring (3COL)

INPUT: an undirected graph G = (V, E)

QUESTION: Does there exist an assignment of colors to each vertex v ∈ V such that no more than 3 colors are used and for every edge (u, v) ∈ E, u and v are assigned different colors.

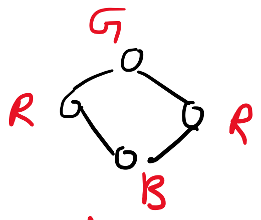
All But Five Three-Colorable (AB53C)

INPUT: a connected undirected graph G = (V, E).

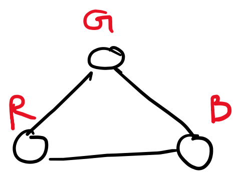
QUESTION: Does there exist $V^{|} \subseteq V$ and an assignment of one of three colors to each vertex in $V^{|}$ such that no two adjacent vertices are assigned the same color and $|V^{|}| \geq |V| - 5$? In other words, is G 3-colorable if we are allowed to not color up to 5 vertices? Show that AB53C is NP-complete by constructing a $\leq_m^P$-reduction f from 3COL to AB53C.

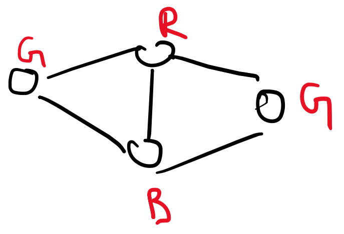
1. Describe a polynomial-time function f that given input G1 outputs G2 where G1 and G2 are undirected graphs. Note that G2 must be a connected graph.
   N.B.: the reduction function f has G1 as its ONLY input. It has no other information. In particular, it does not know whether G1 has a 3-coloring.

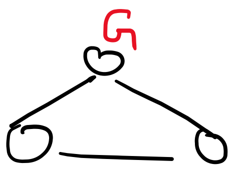
A. Given 3COL, Graph G(V,E)
   We know that 3COL is NP and it runs in a polynomial time.
   Consider the following gadgets:



red = true
blue = false



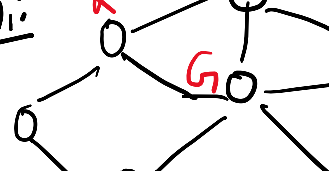




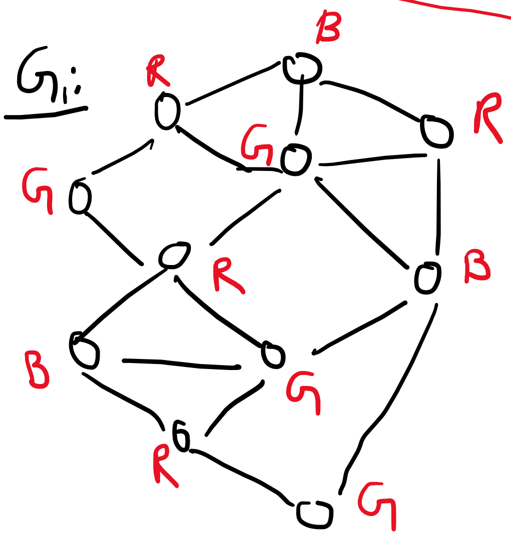
Negate

The above gadgets ensure that for a given any graph G(V,E) there is 3COL
We have to prove reduction from 3COL to All But 5 3 COL, we have to consider some other gadgets because in AB53C we do not consider 5 vertices but 3COL considers and 3COL might not be 3COL but AB53C will have 3COL after not coloring those vertices.

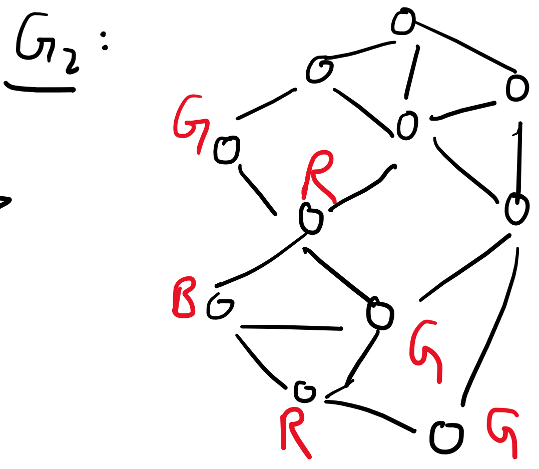
should not be fully connected



These gadgets have no 3CoL for given graph G(V,E) but it is 3CoL for AB53C

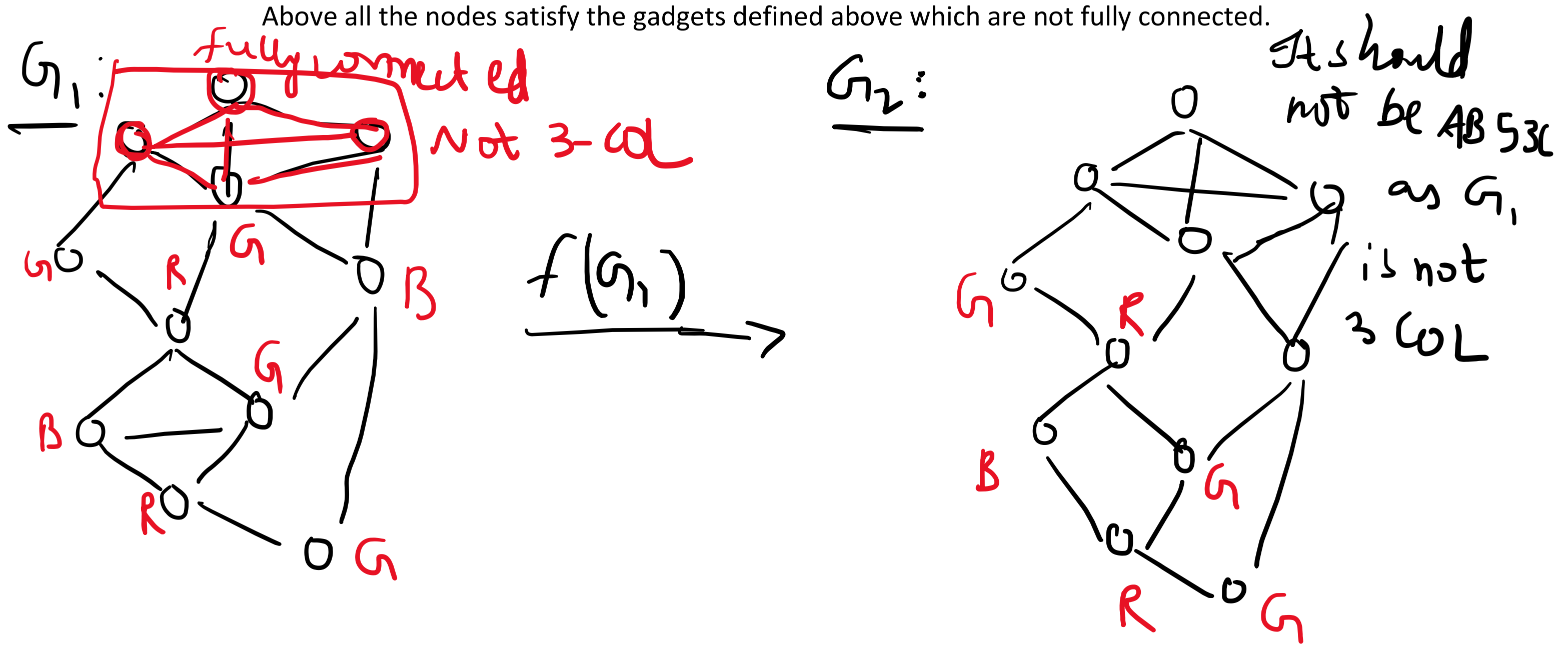


$f(G_1) \longrightarrow$



AB53C

Above all the nodes satisfy the gadgets defined above which are not fully connected.



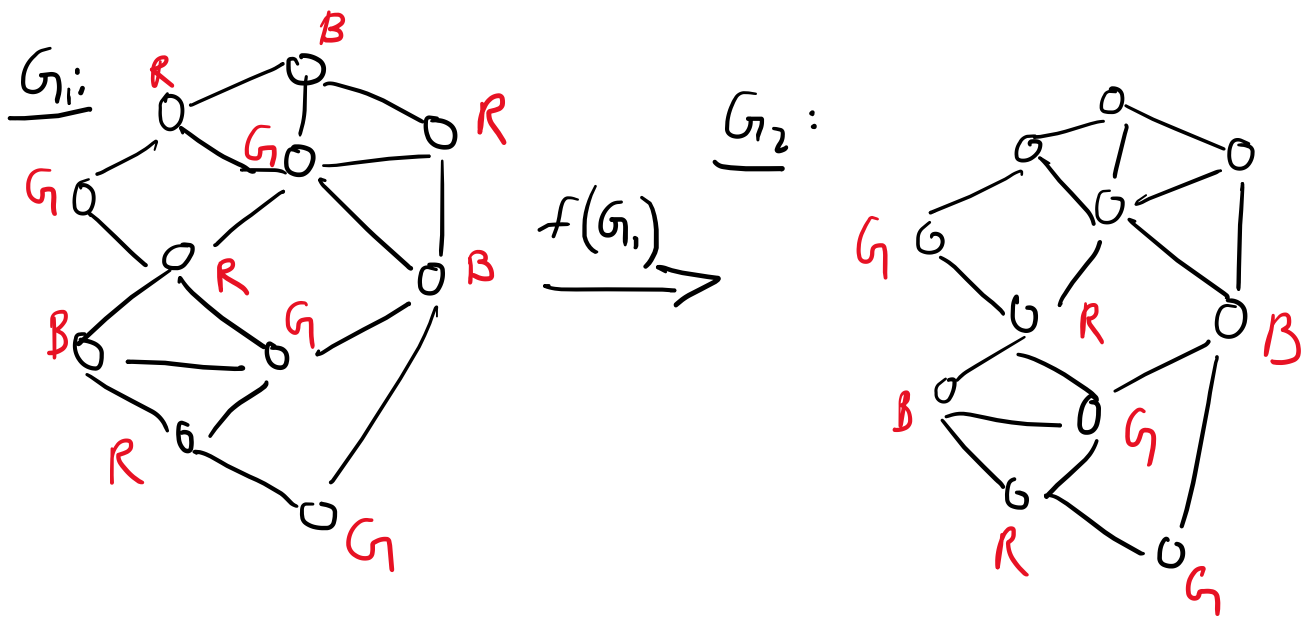
With the help of the above gadgets without fully connected components we define our polynomial time function we output G2 from G1. Our gadgets ensures that the graph will be 3COL.

2. Briefly argue that f runs in time polynomial in the size of G.

A. We know 3COL is NP and it runs in polynomial time which is O(V+E) i.e. every vertex is visited to mark the color and adjacent vertices are checked such that there are no two same adjacent colors assigned.
AB53C also considers every vertex except 5 vertices. Therefore, it also runs in a polynomial time which is O(V+E).

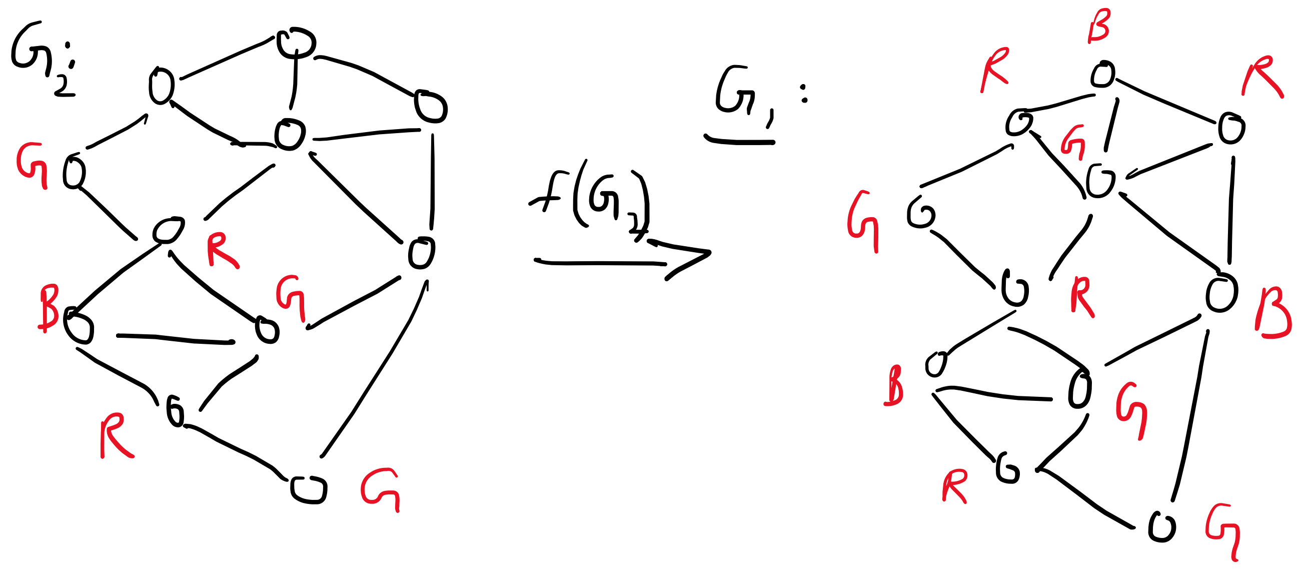
3. Prove that if G1 ∈ 3COL then G2 ∈ AB53C where G2 = f(G1) for the function f you described in part 1.

A. Claim: G1 ∈ 3COL then G2 ∈ AB53C where G2 = f(G1)
Proof:
All vertices has exactly one color
No two adjacent vertices have same color.
For the given Graph G(V,E) we consider the gadgets defined, the Graph should consider nodes in the form of defined gadgets which are not fully connected.
If the given graph is according to the gadgets which are not fully connected then we can ensure that each vertex gets one color, no two adjacent vertices has same color and the graph is 3COL
For the transformation of G1 i.e. G2=f(G1)
G2 has same 3COL as of G1 but only 5 vertices are not colored, still it is 3COL as G1 was 3COL.



4. Prove that if G2 ∈ AB53C then G1 ∈ 3COL where G2 = f(G1) for the function f you described in part 1.

A. <u>Claim:</u> G2 ∈ AB53C then G1 ∈ 3COL where G2 = f(G1)
<u>Proof:</u>
All vertices has exactly one color
No two adjacent vertices have same color.
If we perform coloring for the uncolored vertices in G2 should follow the rules of having exactly one color for vertex and no two adjacent vertices have same color
If the given graph is according to the gadgets which are not fully connected then we can ensure that each vertex gets one color, no two adjacent vertices has same color and the graph is 3COL.
For the transformation of G2 i.e. G1=f(G2)
G2 has same 3COL as of G1 but only 5 vertices are not colored, still it is 3COL.
We can say G1 is also 3COL if we consider the gadget defined such that there are no fully connected 4 vertex components in the graph G1 and G2.



3 Quarter Clique.

Consider the following two problems:

Clique

INPUT: an undirected graph G = (V, E) and a natural number k, where 1 ≤ k ≤ |V |.

QUESTION: Does G have a k-clique? I.e, does there exist V' ⊆ V such that |V' | = k and every pair of vertices in V' is connected by an edge in E?

QuarterClique (QC)

INPUT: an undirected graph G = (V, E). QUESTION: Does G have a clique with |V |/4 vertices? I.e, does there exist V' ⊆ V such that |V' | = |V |/4 and every pair of vertices in V' is connected by an edge in E?

Prove that Quarter-Clique is NP-complete by constructing a $\le_m^P$-reduction from k-Clique:

1. Describe a $\le_m^P$ -reduction f from Clique to QC.
   You should describe how an instance of Clique is transformed into an instance of QC. Make sure that your reduction is in the correct direction for showing that QC is NP-complete. Pictures are great, but do include enough English to make your description unambiguous. N.B.: in the definition of Clique, the value k can range from 1 to |V |.

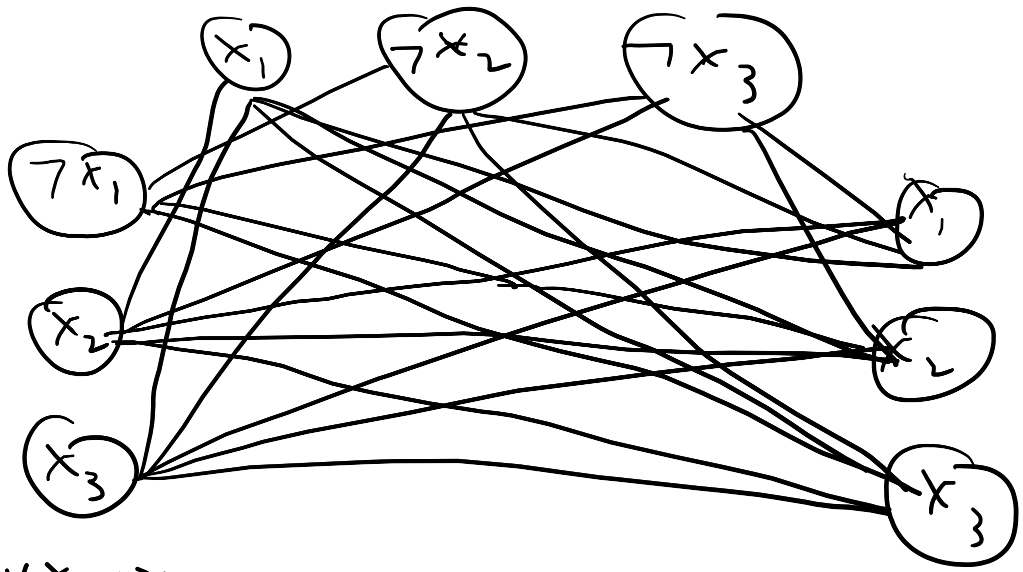
A. Given Graph G(V,E) with k-clique.
   We know that k-clique is NP and runs in a polynomial time

$\phi = C_1 \wedge C_2 \wedge C_3$

$C_1 = x_1 \vee \neg x_2 \neg x_3$

gadget

G:



$C_2 = \neg x_1 \vee x_2 \vee x_3$

$C_3 = x_1 \vee x_2 \vee x_3$

add 1 vertex for each literal: x_i, $\neg x_i$

add all edges except b/w:

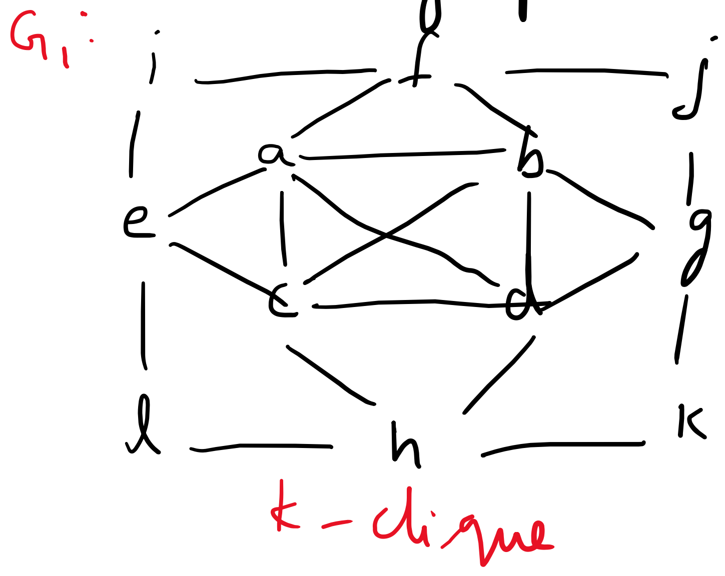
2 literals from same clause

x_i & $\neg x_i$

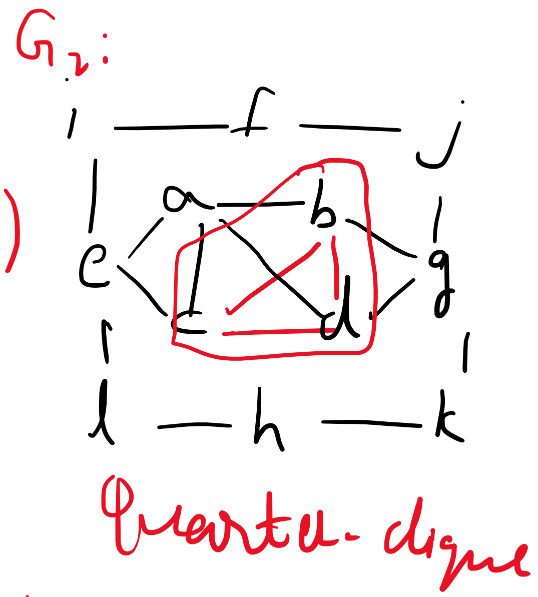
k-clique : $v' \subseteq v \ \ni |v'| = k$

Quarter clique : $\frac{|v|}{4}$; $v' \subseteq v$; $v' = \frac{|v|}{4}$

Consider a graph:



$\xrightarrow{f(G_1, 4)}$



$V = 12$

$k = 4$

$v' = k$

$v' = \frac{|v|}{4} = 3$

Verify from the gadget which 3 vertices has cliques. The gadget returns the vertices which has clique.

if $k = \frac{|V|}{4}$ then no transformation as $G_1 = G_2$

i.e $G_1 = f(G_1) = G_2$

if $k > \frac{|V|}{4}$ as above then with the help of gadget find the cliques for Quarter clique

if $k < \frac{|V|}{4}$ then there is no Quarter clique.

2. Prove that if (G1, k) ∈ Clique then G2 ∈ QC where G2 = f(G1, k) for the function f you described in part 1.

A. Claim: (G1, k) ∈ Clique then G2 ∈ QC where G2 = f(G1, k)
Proof:
k-clique vertices has edge with every k-1 vertex

As discussed above if $k = \frac{|V|}{4}$ then the no transformation i.e $G_1 = f(G_1) = G_2$
If $k < \frac{|V|}{4}$, then for every $\frac{|V|}{4}$ vertices find the vertices with the help of gadgets.
Give the satisfying assignment & see which vertices are in the Quarter clique.
There should be atleast one vertex from each clause.
Atleast one literal should be true.
if it follows the same then $G_2 = f(G_1)$

3. Prove that if G2 ∈ QC then (G1, k) ∈ Clique where G2 = f(G1, k) for the function f you described in part 1.

A. Claim: G2 ∈ QC then (G1, k) ∈ Clique where G2 = f(G1, k)

Proof:

Each clause has a true literal

$k = \frac{|V|}{4}$ (or) $k < \frac{|V|}{4}$ if it is satisfied

then it is a Quarter clique having k-cliques

vertices as its vertices.

With the help of the gadgets we can find

for $k < \frac{|V|}{4}$ cliques as some of the vertices

are in k i.e the satisfying assignment

tells which vertices would be in k in G_1

which are also in G_2 but very few vertices.

if $k = \frac{|V|}{4}$ then $G_1 = f(G_2) = G_2$

i.e no change in transformation.

References:

https://archive.org/details/introduction-to-algorithms-by-thomas-h.-cormen-charles-e.-leiserson-ronald.pdf/page/427/mode/2up

https://archive.org/details/AlgorithmDesign1stEditionByJonKleinbergAndEvaTardos2005PDF/page/n201/mode/2up

16-NP-Completeness-post.pdf - Google Drive

17.1-SAT-vs-Coloring-post.pdf - Google Drive

18.3-Hamiltonian-Cycle-post.pdf - Google Drive

19.1-Clique-post.pdf - Google Drive