Question 1:

Algorithm 1 Minimal weight cycle in input graph, G, that contains an edge E_{uv}

a)
```
procedure SHORTEST_CYCLE :: INPUT → (G = (V,E), $E_{uv}$)
    let G' = $G - E_{uv}$                                ▷ Remove $E_{uv}$ from G
    let P = Dijkstra(G')                                 ▷ Shortest path edges from $V_u \to V_v$ in G'
    if dist[v] $\neq \infty$ then
        return $P \cup \{E_{uv}\}$
    else
        return no cycle exists that contains $E_{uv}$
procedure DIJKSTRA($G = (V, E)$)
    for $V_u \in$ V do
        dist[$V_u$] = $\infty$
    dist[s] = 0
    H = LinkedList(V)
    while H $\neq \emptyset$ do
        $V_t$ = deletemin(H)
        for $\{V_u, V_v\} \in E$ do
            if dist[$V_u$] + wt($V_u$, $V_v$) < dist[$V_v$] then
                dist[$V_v$] = dist[$V_u$] + wt($V_u$, $V_v$)
                decreasekey(H, $V_v$)
```

b) Remove edge E_{uv} from G, and let that graph be denoted G'. There are only two possible cases that can occur, a path, P, exists from $u \to v$ in G', or it doesn't.

In the case that P exists, that is to say that dist[v] $\neq \infty$, it would indicate that a cycle exists in G, as an alternate path from u to v exists besides E_{uv}. If DFS were to be performed on G' starting from u, v would be marked v as visited. Now if we were to add edge E_{uv} back in and rerun DFS, v would be both visited through P, and by E_{uv}; revisiting an already marked edge would indicate a cycle, as a back edge is present in G. $\therefore$ the algorithm will hold in the case that P exists.

If P does **not** exist, it would indicate that no cycles in G exists; also implying that G' is not connected. If G' were to be connected, performing Dijkstra's on G' (line 3 of the pseudo-code) will return the shortest path achievable from one vertex to another. Dijkstra's will indicate that no path exists, and the algorithm will correctly return false.

c) The running time of this algorithm is $O(|V|^2)$. The time complexity can be described in terms of the individual operations that occur in the algorithm.

Creating a graph G', where E_{uv} is removed, iterating through the edge set and finding and removing that edge could take at most as many operations as the size of the edge set; which at worst case is of size $|V|^2$.

Next Dijkstra's will be performed, implemented using a *Linked List* running operations **insert, decreasekey** will take constant time. As for finding and removing the minimum element using **deletemin**, this will take at most $|V|$ operations; seeing as you would have to potentially check all vertices before you find the suitable candidate. With a dense graph, the LinkedList implementation of Dijkstras will be the most efficient; as running time will be $O(|V|^2)$ as opposed to $O(|V|^2 log(|V|)$.

The rest of the operations in the algorhtm will happen in constant time: checking if P exists, then either returning the union of two sets P and $\{E_{uv}\}$, or returning that no cycle had been found. In the following algebraic representation I will disregard these operations, for the sake of simplicity; as it does not reflect the growth of the algorithm.

Let the size of the vertex set V be denoted as n.

$$T_n = O(n^2)$$
$$n^2 \leq cn^2$$

Let $c = 1$, and $n_0 = 1$; it holds that $T_n = O(n^2)$ for all values of $n \geq n_0$ when $c = 1$.

Question 2:

a) For any two arbitrary vertices, $u, v \in G$, stating that for the shortest path from $u \to v$, denoted as $S(u, v)$, is vertex disjoint to any other existing path from $V_u \to V_v$, denoted $P(u, v)$, is false.

To say that something holds for any two arbitrary points, would also claim that is should hold for any pair of points in G. Therefore, showing one instance of this problem as false, would make the claim altogether false.

To illustrate this, begin with the graph contained by K_4.

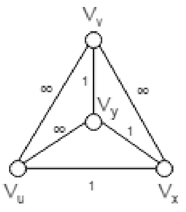

Figure 2.1 - Undirected graph with weighted edges.

It can be seen that upon removing the edge E_{uv}, the graph is still 2-connected, but no path $P(u, v)$ will exist that is vertex disjoint to the shortest path, $S(u, v)$.

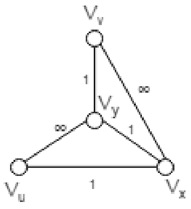

Figure 2.2 - Alteration of graph Figure 2.1 without E_{uv}.

After removing E_{uv} it becomes evident that the shortest path $S(u, v)$ will consume all vertices in G; as no edge with weight of infinity will be used. Since no vertices are left unmarked by S(u,v), $\therefore$ there **cannot** exist a path $P(u, v)$ vertex disjoint from path $S(u, v)$.

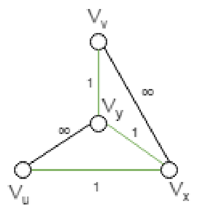

Figure 2.3 - Figure 2.2 modified to visualize path taken by $S(u, v)$

Question 3:

a) Every edge e, for where T (G's minimum spanning tree) intersects H (a connected subgraph of G) is guaranteed to be in the solution for an MST of H, T'.

Now suppose an arbitrary edge, e, exists in T that is **not** in T'. If e is not part of any minimum spanning tree, T, of H thismeans that another edge, v, must exist such that $wt(v) < wt(u)$. This would contradict the fact that T is the MST of G.

Lemma:
The MST of a graph, G, is the union of the MST's of its connected components.

Take T, and remove an arbitrary edge e. Every edge in an MST will exhibit the property that upon removal, two connected components will form. Remove edge e from T, and denote the two connected components as C_1, and C_2, each having it's own MST T_1, T_2. It be seen that performing a union on these two connected components, and taking it's MST will return T. Now take H into consideration. H is a connected-subgraph, aka connected component. Since G is connected, we know that H will span the rest of the elements in $G - H$.

Thus there must be an edge of minimal weight connecting H with $G - H$. Referring to the lemma we can see that performing a union of two connected components of G, and taking the MST of $H \cup (G - H)$ will result in T. Thus showing that T' of H will have all edges wherever $T \cap H$; in other words every edge in $T \cap H$ will be in T'.

Question 4:

a) This problem, let it be denoted as the **Weight Zero Cycle** problem, can be described as **at least** as hard as the **Subset Sum** problem.

For context, the **Subset Sum** problem is a decision problem that will take as input a set of integers, S, and will output yes or no depending on if an $S' \subseteq S$ exists that satisfies $\sum_{s \in S'} s = 0$. This is a well known NP-Complete problem, and can be used to verify that the **Weight Zero Cycle** problem is also NP-Complete using reduction.

First it must be shown that our problem is in NP; this can be done by showing that the time complexity of verifying that an already proposed solution takes at least polynomial time. The solution in this case being a cycle, which has an edge weight sum of 0. To verify this solution is correct would take at least **at least** n operations, where n is the size of the the clique. Therefore to verify the algorithm is correct will take at least polynomial time.

I will refer to the subset sum problem as Y, and weight zero cycle as X. If it can be shown that $Y \leq pX$, then it confirms that X is NP-Complete.

A cycle can be thought of a subset of edges in G, though it's not just an arbitrary subset of any size. Never the less, when put in these terms, this problem is essentially the subset sum problem. The polynomial factor here being performing DFS to identify the cliques. If a back edge is encountered, a clique is identified. The running time of DFS is $O(|V| + |E|)$. Then to identify if a given subset of edges in this identified clique adds up to 0, where the problem essentially becomes at least as hard as the subset sum problem.

To reiterate,

$$Y \leq pX$$

Y is the WeightZeroCycle problem.
X is the SubsetSum problem.
p is the time it takes to identify cliques.

It can be seen that the weight zero cycle problem is NP-complete.

Question 5:

a) The problem of finding the smallest killer set K can be shown to be at least as hard as finding the minimal vertex cover a graph G.

To show a problem as NP Complete, it has to be shown as both NP Hard, and in NP.

This can be achieved by showing that the problem can be *verified* in polynomial time, and show that the problem can be reduced to some NP-Complete problem Y, s.t $Y \leq pX$.

Given an instance of proposed solution to the killer set problem, it can be shown that the it takes polynomial time to verify the correctness of this solution.

Proposed Solution

G = (V,E), C = $\{v_1, v_2 \dots v_n\}$, s.t $C \subseteq V$

Verification Algorithm

- Iterate through C, let current element be u - $O(|V|)$
- Check E it contains an undirected edge, e, s.t one of e's endpoints is u. - $O(|V|)$.
- Remove edge from e from E
- If the edge set is empty, after iteration, return true. If not, return false.

The verification algorithm at worst will take $O(|V|)$, and we can proceed to determine if the Killer Set problem is in NP-Complete.

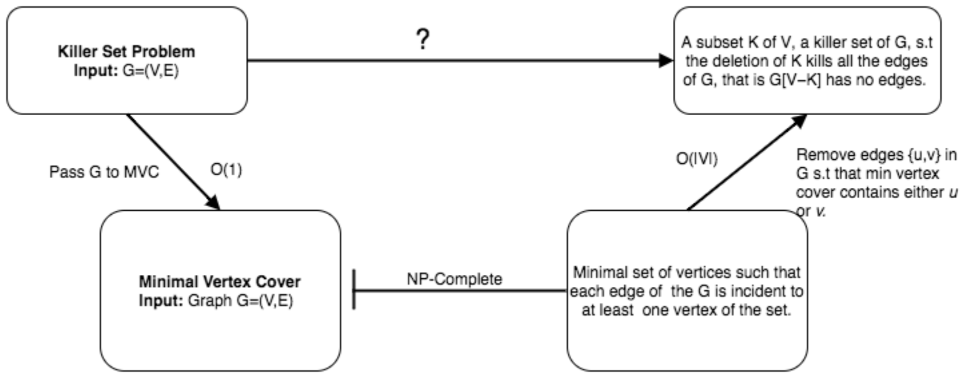
Killer Set Problem
Input: G=(V,E)
?
A subset K of V, a killer set of G, s.t the deletion of K kills all the edges of G, that is G[V−K] has no edges.
Pass G to MVC
O(1)
O(|V|)
Remove edges {u,v} in G s.t that min vertex cover contains either u or v.
Minimal Vertex Cover
Input: Graph G=(V,E)
NP-Complete
Minimal set of vertices such that each edge of the G is incident to at least one vertex of the set.

Figure 5.1 - Reduction Flow Chart

It is known that the the Minimum Vertex Cover problem is NP-Complete, showing the killer subset problem as at least as hard as Minimum Vertex Cover shows that it's also NP-Complete.

Passing the input of our problem to minimal vertex cover is constant time, as they take the same input. But the process of removing edges from the graph to determine a successful vertex cover is at most the size of the vertex set; thus linear time complexity.

Let Y = Minimal Vertex Cover
Let X = Killer Set Problem
Let p = Time it takes to remove edges to verify if vertex cover is killer set.

It holds that $Y \leq pX$.

References

[1] *http://seed.ucsd.edu/mediawiki/images/9/97/Lec6.pdf*

[2] *http://orac.amt.edu.au/notes/GraphTheory2-Dec2009.pdf*

[3] *http://www.cs.cmu.edu/afs/cs/academic/class/15750-s11/www/practice1s ln.pdf*

[4] *http://techieme.in/further-reading-for-minimum-spanning-trees/*

[5] *https://arxiv.org/pdf/1209.0700.pdf*