COMP 362 Honours Algorithm Design 2014
Assignment 3 Solutions

1) To show that Triangle Cover is in NP, we can use as a certificate that the input is a yes instance, an integer $j \leq k$ and a list W of j vertices of G, $w_1,\ldots,w_j$. To verify that we have a yes instance, we first verify that the input string does indeed correspond to an integer k and a graph with vertex set V and edge set E. We then verify that W is a list of vertices of this graph. Finally, we then check for each triple of vertices of V-W that there is some pair of the triple which are not joined by an edge.
   To show that Triangle Cover is NP-hard, we reduce Vertex Cover to it. Given an instance (G,k) of vertex cover, we obtain an instance (G',k) of Triangle cover by adding a vertex v_e corresponding to each edge e of G, and adjacent precisely to the endpoints of e. We need to show that G' has a triangle cover of size at most k precisely if G has an edge cover of size k.
   We let V' be the set of vertices of G' which are not vertices of G.
   Since the vertices of V' form a stable set in G', every triangle T of G' contains at most one such vertex. Hence T contains two vertices of G which are adjacent in G' and hence in G. Thus every triangle of G' contains an edge of G, and so any vertex cover of size at most k for G is a triangle cover of size at most k for G'.
   Now, suppose that G' has a triangle cover of size at most k, and choose such a cover S with as few vertices as possible and subject to this containing as few vertices of V' as possible. Any vertex v_e of $V' \cap S$ is in exactly one triangle whose other vertices are the endpoints of e. So, neither of these vertices is in S, as otherwise $S-v_e$ would still be a triangle cover for G', contradicting the fact that we chose S to be as small as possible. But now, letting x be an endpoint of e, we see that $S-v_e+x$ is a triangle cover of G', which is also a contradiction to the way in which we chose S. So S is a set of vertices of at most k vertices of G. Furthermore for any edge e of G, since S contains a vertex of the triangle formed by the endpoints of e and v_e, it must contain an endpoint of e. I.e. S is a vertex cover for G.

2) To show that this problem is NP complete, we can use as a certificate k vertex disjoint paths of G, $P_1,\ldots,P_k$ where P_i links s_i to t_i. More precisely, we give k ordered sets of vertices $X_1,\ldots,X_k$ where each X_i begins with s_i and ends with t_i. We again need to check that our input corresponds to an integer k, a graph G, and 2k vertices $s_1,\ldots,s_k,t_1,\ldots,t_k$. We then check that each X_i is an ordered set of vertices of G beginning with s_i and ending with t_i, and that each vertex of G appears at most once in the union of these sets. Finally, we check that every two vertices which are consecutive in some X_i are joined by an edge. This is all easy to do in polynomial time
   To show the problem is NP-hard, we reduce 3-SAT to it. We consider an instance of 3-SAT consisting of a set C of m clauses over a set $X=\{x_1,\ldots,x_n\}$ of n variables (we assume that for every variable of X one of x_i or not(x_i) appears in a clause, as otherwise we could delete it and reindex the other variables). We set k=m+n and construct a graph G with 5m+2n vertices. For each clause C_i, we have a vertex corresponding to each of the literal z_j^1, z_j^2, and z_j^3 in the clause. In addition we have the vertices $s_1,\ldots,s_k$ and $t_1,\ldots,t_k$.

For j between 1 and m, both s_j and t_j are adjacent to z_j^1, z_j^2, z_j^3 and to no other vertices. For i between 1 and n, we will add edges as follows, we let Pos(i) be the set of vertices corresponding to some z_j^k which is x_i and let Neg(i) be the set of vertices corresponding to some z_j^k which is not(x_i). We choose any ordering of $\{s_{i+m}, t_{i+m}\} \cup Pos(i)$ which begins with s_{i+m} and ends with t_{i+m} and add an edge between vertices which are consecutive in this order. We choose any ordering of $\{s_{i+m}, t_{i+m}\} \cup$ Neg(i) which begins with s_{i+m} and ends with t_{i+m} and add an edge between vertices which are consecutive in this order. (Note that if Neg(i) or Pos(i) is empty then s_{i+m} and t_{i+m} are joined by an edge.) The edges in the paths formed are the only edges of $G'=G-\{s_1,\ldots,s_m,t_1,\ldots,t_m\}$. Thus G' has n components $U_1,\ldots U_n$ where U_i is a cycle containing s_{i+m} and t_{i+m}. Hence, every path of G' from s_{i+m} to t_{i+m} either uses all the vertices of Neg(i) or all the vertices of Pos(i).

We need to show that there is a satisfying truth assignment for C precisely if there are k vertex disjoint paths $P_1,\ldots,P_k$ of G such that P_i links s_i to t_i.

Given a satisfying truth assignment we obtain the paths as follows; (I) for each i between 1 and n, if x_i is assigned to be false (respectively true) , we let P_{m+I} be the path of G' between s_{m+I} and t_{m+I} whose interior is the vertices of Pos(i) (resp. Neg(i)), and (II) for each i between 1 and m, we choose a vertex corresponding to a literal of C_i assigned to be true by the satisfying assignment to be the interior of P_i.

Given a set of paths, to obtain a satisfying truth assignment we note that for i between 1 and n, P_{i+m} is a path of G', and so uses all the vertices of Pos(i) or uses all the vertices of Neg(i). If it uses all of Pos(i) we set x_i to be false, otherwise we set x_i to be true. Now, for i between 1 and m, the neighbour of s_i on P_i is a literal of C_i which must be true under the assignment. So, this assignment satisfies all the clauses.

3) Kleinberg and Tarjan say that A reduces to B if we can solve A in polynomial time using a subroutine that solves B in polynomial time. We may call this subroutine many times. In contrast, for Sipser A reduces to B, if we can construct in polynomial time an instance of B with the same answer as the input instance of A. Thus, essentially we are only allowed to call the subroutine once and we must call it at the end of our algorithm for solving A.

Also,Kleinberg & Tardos discuss problems where Sipser discusses language To be Precise, Kleinberg and Tardos would need to specify the encoding that is used to input instances to the Turing Machine.

Finally, with their discussion of input strings for instances, Kleinberg and Tardos gloss over the fact that there are some strings which are not valid input, and that our algorithm should really be able to handle such strings efficiently. For example, consider the following problem: Hamilton Cycle in Hamiltonian graphs. The input is a graph with a Hamilton Cycle and we are asked to determine if the input graph has a Hamilton cycle. It is easy to determine which inputs corresponding to valid inputs are yes instances in polynomial, and in fact, constant time, we just answer yes. However, we do not know how to distinguish in polynomial time between inputs G which are valid because they have a Hamilton cycle and those which are invalid because they do not.