CPSC-320 Intermediate Algorithm Design and Analysis Winter 2000

Solutions to Homework # 5

1. (a) Independent Set (IS) decision problem: given a graph G and a number k, does G have an independent set of size at least k?

   To show that IS is NP-Complete, we need to show that (i) IS is in NP and (ii) some NP-Complete problem is polynomial-time reducible to IS; we will use CLIQUE for this purpose.

   (i) IS is in NP: a verification algorithm for IS takes as input an instance (G, k) of IS and also a set S of nodes of G. It checks that the set S has at least fk nodes and also that no pair of nodes in S are connected. If so, the algorithm outputs "yes" and otherwise the algorithm outputs "no." This can be done in $O(n^2)$ time.

   (ii) Reduction from CLIQUE to IS: Let (G, k) be an instance of CLIQUE. Construct an instance (G', k') of IS as follows: let G' be the graph with the same set of nodes as G; there is an edge $\{i, j\}$ in G' if and only if the edge $\{i, j\}$ is NOT in G. Also, let $k' = k$. Clearly G' can be computed efficiently, given G.

   Correctness: G has a clique of size k if and only if G' has an independent set of size k, because G' is the complement of G.

2. (b) Let Decision-IS be the subroutine that solves the independent set decision problem. We can use this subroutine to find an independent set of maximum size in G as follows. First, run Decision-IS on inputs $(G, n), (G, n-1)$ and so on until the Decision-IS first returns "yes," say on input (G, k). This tells us that k is the size of the maximum independent set.

   Next, let G' be the graph obtained by removing node 1 and all of its neighbors from G, and in addition, all edges from the neighbors of node 1 to other nodes that remain in G'. Run Decision-IS on input (G', k). If the output is "no," it must be that there is a maximum independent set, say I, containing node 1 in G. Set I cannot contain any of 1's neighbors. Therefore, the remaining nodes in I form an independent set the graph G'' obtained from G by removing node 1 and all of its neighbors from G, and in addition, all edges from the neighbors of node 1 to other nodes that remain in G''. To find the remaining nodes in I, recursively find a maximum independent set of size $k-1$ in G''.

   If the output is "yes," it must be the case that there is a maximum independent set of G that does not contain node 1. Hence, recursively solve the problem of finding an independent set of size k in G'.

   Since only ONE of these possibilites (yes or no) occurs, there is only one recusive call depending on whether node 1 is or is not in some maximum independent set. Hence the total time to find a maximum independent set is polynomial.

   (c) If each vertex of G has degree exactly 2, then each connected component of G is a "ring." A maximum independent set can be obtained as follows. For each ring, visit the nodes on that ring, starting at some arbitrary vertex and following the edges in the ring. Place the starting node in the independent set. When a new node i is visited, place it in the independent set if and only if neither of the nodes adjacent to it are in the independent set.

3. (a) No. Since Π_1 is reducible to Π_2, we know that the time needed to solve Π_1 is at most a polynomial time factor more than that needed to solve Π_2. But even if Π_1 is in P, it is possible that exponential time is needed to solve Π_2.

   (b) Yes. This follows from the definition of reduction: We obtain an efficient algorithm for Π_1 by first reducing Π_1 to Π_2 and then running an efficient algorithm for Π_2.

   (c) No; it is possible for example that both Π_1 and Π_2 are in P and that P $\neq$ NP.

   (d) Yes, since all problems in NP are reducible to any NP-complete.

4. (a) The transformation can be done as follows: Let x be an instance of TSP. If x satisfies the triangle inequality, x is simply mapped to x. Otherwise, let k be the max, over all triples a, b, c of $cost(a, c) - (cost(a, b) + cost(b, c))$. Now, simply produce an instance, say y, of TSP that satisfies the triangle inequality by keeping the set of cities the same and adding k to the cost of every edge in the instance x.

   Note that if $cost'$ is the cost function for the instance y, then $cost'(a, c) = cost(a, c) + k$ and $cost'(a, b) + cost'(b, c) = cost(a, b) + cost(b, c) + 2k$. Combining these identities with the fact that $cost(a, c) - (cost(a, b) + cost(b, c)) \leq k$, it follows that $cost'(a, c) \leq cost'(a, b) + cost'(b, c)$ and thus the triangle inequality holds for the instance y.

   Also, the cost of a tour of y is exactly the cost of the same tour in x plus nk. Therefore, the optimal tours of x are the same as the optimal tours of y.

   (b) The polynomial time transformation of part (a) does not "preserve approximability." For example, consider the simple instance x of TSP with just four cities, a, b, c, d. Suppose that all costs are 0, except that $cost(a, b) = T$ for any large value T. There are tours of total cost 0 in x, for example, a, c, d, b, a. Using the transformation of part (a), the instance x is mapped to an instance y in which the costs between all pairs of cities is T except the cost between a and b is now $2T$. In the instance y, the tour a, b, c, d, a has cost $6T$ which is just T greater than the cheapest tour, which has cost $5T$. But, mapped back to instance x, the tour a, b, c, d, a is clearly not within a constant factor of the optimal, which is 0. problem.