CS 577: Introduction to Algorithms 12/7/2015

Homework 11 Solutions (Review Problems)

Instructor: Dieter van Melkebeek TA: Kevin Kowalski

Problem 1

To show that Subgraph Isomorphism is NP-complete, we first show that is in NP. Without loss of generality, assume that all graphs are given as adjacency matrices, and let each witness be an ordered subset of the vertices of G. In linear time we can generate the adjacency matrix corresponding to the subgraph of G induced by the witness (where the i-th row of the matrix corresponds to edges from the i-th vertex in the witness) and check whether this is equal to the adjacency matrix of H. Since an ordered subset that produces an adjacency matrix equal to that of H can exist if and only if G has a subgraph isomorphic to H, this defines an efficient verifier for Subgraph Isomorphism, as desired.

To show that Subgraph Isomorphism is NP-hard, note that intuitively, Subgraph Isomorphism is about determining whether a big graph has some part of it that looks like a particular small graph. In fact, we have already seen an NP-complete decision problem that can be cast in this framework: Independent Set is about determining whether a big graph has a large part that is completely disconnected. Since Subgraph Isomorphism is a natural generalization of this, we can expect there to be a straightforward reduction from Independent Set to Subgraph Isomorphism.

The reduction is as follows. Given an instance (G, k) of Independent Set (where the goal is to determine whether G has an independent set of size at least k), our reduction is as follows. Create the adjacency matrix for a graph H on k vertices with no edges, and ask the Subgraph Isomorphism oracle whether G has a subgraph isomorphic to H. If G does, then we know it must have a set of k vertices with no edges between them, which form an independent set of size at least k. If G does not, then every set of k vertices in G must have at least one edge between them, so the largest independent set in G must have size $< k$. Hence, this reduction can be framed as a mapping reduction, and is correct if we return the same result as the oracle. Since creating H can trivially be done in linear time, this defines a polynomial-time reduction from Independent Set to Subgraph Isomorphism, so Subgraph Isomorphism is NP-hard and thus NP-complete.

Problem 2

We show that Multiple Interval Scheduling is NP-hard. In this problem we are given n jobs and associated time intervals for each job, and we are asked to determine the maximum number of jobs we can schedule so that no two scheduled jobs have any overlap in time.

To show that the problem is NP-hard, we will demonstrate a reduction from Independent Set. The intuition here is that these two problems have a similar flavor–in both we are seeking to choose a maximum number of items that do not 'interfere' with each other. We'll construct our reduction so that adjacent vertices in an instance of Independent Set correspond to overlapping jobs in the instance of Multiple Interval Scheduling we construct, and vice-versa.

Suppose we are given a graph G, with n vertices and m edges, and asked to find the size of the largest independent set in G. We construct an instance of Multiple Interval Scheduling with n jobs (one for each vertex) and m time intervals. That is, we divide our 'day' up into m indivisible

'hours', $t_1, t_2, \cdots, t_m$. Now, we label the edges of G as $e_1, e_2, \cdots, e_m$, and for each edge $e_i = (u, v)$, we say that jobs u and v both need to use the processor during time interval t_i.

Thus, each edge of G prevents the jobs corresponding to its endpoints from being accepted simultaneously. Conversely, if two jobs in our schedule cannot be accepted simultaneously, then their corresponding vertices must be connected in G, since two jobs will have overlapping time intervals only when there is an edge between their corresponding vertices in G.

Hence, we can accept k jobs in the scheduling problem if and only if there are no edges between the k corresponding vertices in G, so the maximum number of jobs we can schedule is equal to the size of the largest independent set in G. This reduction can be accomplished in polynomial time since we need only check each pair of vertices in G to see if they are connected by an edge, so Multiple Interval Scheduling is NP-hard, as desired.