COL351: Analysis and Design of Algorithms

Tutorial Sheet - 11

November 18, 2022

Question 1 You are given a graph $G = (V, E)$ and an integer k. A set $I \subseteq V$ is defined to be *Strongly Independent* if for any two nodes $v, u \in I$, the edge (v, u) does not belong to E, and there is also no path of 2 edges from u to v, i.e., there is no node w such that both $(u, w) \in E$ and $(w, v) \in E$. The Strongly Independent Set problem is to decide whether G has a strongly independent set of size k. Prove that the Strongly Independent Set problem is NP-complete.

Solution Checking membership in NP is straightforward. We reduce from independent set. Let G, k be an instance of the independent set problem. We map it to an instance of the Strongly Independent Set problem. For every edge e in G, we subdivide it by adding a new vertex x_e, i.e., if $e = (u, v)$ is an edge, then we replace it by two edges : $(u, x_e), (x_e, v)$. Further, we form a clique over all the new vertices x_e. Call this new graph G_0.

If S is an independent set in G, then the same set of vertices is a strongly independent set in G_0.

Conversely, let S be a strongly independent set in G_0. The non-trivial scenario is $|S| > 1$. Now S cannot contain any of the new vertices x_e (because any other vertex in G_0 is reachable from x_e by a path of length at most 2). Therefore, S contains vertices which correspond to those in G, i.e. $S \subseteq V(G)$. These vertices form an independent set in G.

Question 2 *Group Interval Scheduling (GIS) Problem* is defined as follows. You have a processor that is available to run jobs under the constraint that it can work on only one job at any single point of time. Jobs in this model, however, are more complicated - each job requires a set of intervals of time during which it needs to use the processor. For example, a single job could require the processor from 10 AM to 11 AM, and again from 2 PM to 3 PM. If you accept this job, it ties up with the processor during those two hours, but could still accept jobs that need any other time periods (including the hours from 11 to 2).

You are given a set of n jobs (J), each specified by a set of time intervals, and the question is: For a given number k, is it possible to accept at least k of the jobs so that no two of the accepted jobs have any overlap in time? Show that Group Interval Scheduling (GIS) is NP-complete.

Solution Checking membership in NP is straightforward. We reduce from Independent Set. Take an instance of independent set: graph $G = (V, E)$ and number k. We map it to an instance of the Group Interval Scheduling (GIS) as follows: the time interval stretches from 0 to $m = |E|$. Let the edges be numbered $e_1, ..., e_m$. Then the interval $[i, i+1]$ will correspond to the edge e_i. For a vertex v, we define a job J_v: this job J_v will consist of all intervals $[i, i + 1]$, for every edge e_i, which has v as an endpoint. It is easy to check that G has an independent set of size k iff the corresponding k jobs in the GIS instance do not overlap.

Question 3 Consider the Group Interval Scheduling (GIS) problem defined in the above question. For any given job $j \in J$, let Δ_j be the number of jobs in $J \setminus \{j\}$ that overlaps with j. Present a $(\Delta + 1)$-approximation algorithm for GIS problem, where $\Delta = \max_{j \in J} \Delta_j$.

Hint: Recursively pick a job that overlaps with minimum number of jobs.

Solution Intialize answer A to empty-set. Repeat the following two steps until no jobs are left:

1. Pick a job $j \in J$ that overlaps with minimum number of jobs in J, and add it to A
2. Remove job j and all jobs that overlap with j from set J.

The number of iterations is at least $n/(\Delta + 1)$ as in each iteration at least $(\Delta + 1)$ jobs are removed from J. So,

$$|A| \geq \frac{n}{(\Delta + 1)} \geq \frac{A_{opt}}{(\Delta + 1)}.$$

Question 4 Show that SAT $\leq_P$ 3-SAT.

Solution See http://www.cs.ecu.edu/karl/6420/spr16/Notes/NPcomplete/3sat.html

Question 5 Design an $O(2^k\ poly(n))$ time algorithm for k-vertex-cover problem: *Given a graph G on n vertices check if there exists a vertex cover of size at most k in G.*

Solution

Algorithm 1: *VertexCover(G, k)*

```
1 if $G$ has no edges then Return true;
2 if $k = 0$ then return false;
3 $(u, w) \leftarrow$ an arbitrary edge of $G$;
4 if VertexCover$(G - u, k - 1)$ then Return true;
5 if VertexCover$(G - w, k - 1)$ then Return true;
6 Return false;
```

Claim: If G has a vertex-cover C of size k, then this cover contains at least one of u or w. If C contains u, then the first recursive call will return true. If C contains w (but not u) then the second recursive call will return true. Otherwise, if G has no vertex cover of size k then the algorithm returns false.

The algorithm is clearly correct when $k = 0$. To prove that it's correct for $k > 0$ we can use induction and above claim.

Time complexity: The running time is $O(2^k\ poly(n))$ as k is the maximum depth of recursion.