Advanced Analysis of Algorithms - Final (Solutions)

K. Subramani
LCSEE,
West Virginia University,
Morgantown, WV
{ksmani@csee.wvu.edu}

1 Problems

1. Consider the following greedy algorithm for finding the Minimum Vertex Cover in an undirected, labeled graph $\mathbf{G} =< \mathbf{V}, \mathbf{E} >$: Select the vertex with the maximum degree and add it to the cover. In the event that more than one vertex has the maximum degree, choose the vertex with the smallest label. Now remove this vertex and all its incident edges from $\mathbf{G}$ (these edges are covered by the chosen vertex). Repeat this step till all edges are deleted from the graph. It is clear that this algorithm produces a cover. Will this algorithm produce a minimum size cover? Justify your answer with a proof or a counterexample.

   Solution:

   Consider the labeled graph in Figure (1). If we use the above algorithm the vertex cover that is returned is $V' = \{v_3, v_4, v_2, v_6\}$, whereas the optimal vertex cover is $V' = \{v_4, v_2, v_6\}$.

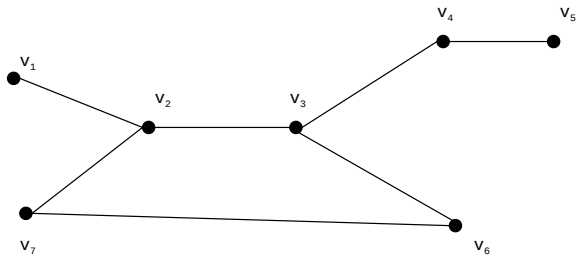

Figure 1: Counterexample to the described algorithm for Minimum Vertex Cover

   Note that labeling does not alter the `NP-completeness` of the vertex covering problem and it is unlikely that this heuristic or any other polynomial time heuristic for that matter, is going to produce the optimal cover. □

2. In the stagewise shortest path problem, you are given a staged graph $\mathbf{G} =< \mathbf{V}, \mathbf{E}, \mathbf{s}, \mathbf{t}, \mathbf{k}, \mathbf{c} >$, where, $\mathbf{V}$ is the set of vertices, $\mathbf{E}$ is the set of edges, s is the source vertex, t is the sink vertex, k is the number of stages and $\mathbf{c}$ is a weighting function that associates a positive weight to the edges in $\mathbf{E}$. In stage 1, s is the only vertex; likewise, in stage k, t is the only vertex. All the other vertices are partitioned among the other stages, with each stage containing at least one vertex. Edges in the the graph are strictly from stage i to stage $(i + 1)$, $1 \leq i \leq k - 1$. Describe a linear time algorithm to determine the shortest path from s to t. You may assume that $|\mathbf{V}| = n$ and $|\mathbf{E}| = m$.

   Solution: Since the edges are positively weighted, we may be tempted to use Dijkstra's algorithm; however, Dijkstra's algorithm is not known to run in linear time! Interestingly enough, a dynamic programming approach (as opposed to a greedy approach) actually works in time $O(m + n)$. We define $P(i, j)$ to be shortest path from vertex v_j in Stage i to vertex t and $d(i, j)$ to be the actual cost of this path.

We exploit the stagewise nature of the graph to derive:

$$\begin{aligned} d(i,j) &= \min_{v_l \in Stage\ (i+1),(v_j,v_l)\in E} c(v_j,v_l) + d(i+1,l) \\ \forall v_j \in Stage\ (k-1)\ d(k-1,j) &= c(v_j,t) \end{aligned}$$

Using an adjacency list structure to represent the staged graph and the associated data structures $d()$ and $P()$, we can implement the above recurrence relation in $O(m+n)$ time. Note that we are concerned with computing $d(1,1)$, with v_1 denoting the source vertex s. Further, although the recurrence is written in top-down fashion, the computation is done bottom-up. In other words, we first set $d(k,n) = 0$, where $v_n = t$. This is followed by processing the edges in Stage $(k-1)$, after which the edges in Stage $(k-2)$ are processed and so on. The computation of $d(1,1)$ then takes time proportional to the degree of s. Since each edge is processed exactly once and for constant time, the running time of the algorithm is $O(m+n)$. □

3. Consider Professor Boruvka's algorithm for determining the Minimum Spanning Tree in a weighted, undirected graph.

Function FIND-MST(**G=<V,E>**)
1: Let T be a subgraph containing only the vertices in $\mathbf{V}$.
2: **while** ($|T| \leq (n-1)$) **do**
3: **for** (each connected component C_i of T) **do**
4: Find the lightest edge $e = (u,v)$, with $u \in C_i$ and $v \notin C_i$.
5: Add e to T if it is not already in T.
6: **end for**
7: **end while**
8: **return**(T)

Algorithm 1.1: Boruvka's algorithm for Minimum Spanning Trees

Is the Professor's algorithm correct? Justify your answer.

Solution: Professor Boruvka's algorithm is yet another implementation of the Cut Theorem for Spanning Trees, which states that the lightest edge crossing a cut can always be made part of a Minimum Spanning Tree. This theorem was proved in class during the discussion on Prim's algorithm. It follows that the Professor's algorithm is correct. □

4. In class, we showed that the HORNSAT and 2SAT problems were in P. Consider the following SAT variant called HORN⊕2SAT; a CNF formula is said to be HORN⊕2SAT, if every clause has exactly two literals or (inclusively) is HORN. Argue that checking the satisfiability of a HORN⊕2SAT formula is NP-complete.
Hint: Reduce 3SAT to HORN⊕2SAT.

Solution: Note that the input to a 3SAT instance is a collection of clauses with 3 literals in each clause. If a clause has 2 or more negative literals, the clause is already in HORN form, so nothing needs to be done. Consider a clause C_j with two positive literals, say $(x_1, x_3, \bar{x_4})$. C_j is replaced by the clause set $S_j = (\bar{z_j}, x_3, \bar{x_4})\ (z_j, x_1)$, where z_j is a new boolean variable. It is not hard to see that C_j is satisfied by an assignment if and only if S_j is. For instance, if $x_1 = \mathbf{true}$, S_j can be made **true**, by choosing z_j to be **false**. Likewise, if $x_1 = x_3 = \mathbf{false}$ and $x_4 = \mathbf{true}$, then regardless of how z_j is chosen, S_j will be **false**.

Now consider the case in which C_j is of the form (x_1, x_2, x_3) (three positive literals). We replace C_j by the set $S_j =$

$$\begin{aligned} &(\bar{z_{ij}}, x_1) \\ &(\bar{z_{3j}}, x_2, \bar{z_{2j}}) \\ &(z_{3j}, z_{1j}) \\ &(z_{2j}, x_3) \end{aligned}$$

where z_{ij}, z_{2j} and z_{3j} are new boolean variables. It is not hard to see that if $x_1 = x_2 = x_3 = \mathbf{false}$, S_j will be **false**, regardless of how the $z_{ij},\ 1 \leq i \leq 3$ are chosen. Likewise, if any of x_1, x_2 or x_3 is **true**, then S_j can be made **true**, by selecting the z_{ij} variables appropriately. □

5. In class, we showed that the Maximum Independent Set problem on an undirected graph $\mathbf{G} =< \mathbf{V}, \mathbf{E} >$ is `NP-complete`. What can you say about the complexity of the problem, if $\mathbf{G}$ does not have any cycles. Justify your answer with a polynomial time algorithm or a proof of `NP-completeness`.
Hint: Does Dynamic Programming work?

Solution: Without loss of generality, we can assume that $\mathbf{G}$ is a tree (say T), since if it were a forest, we can find the maximum independent set of each connected component separately and take their union. The next observation that we make is that T can be rooted at an arbitrary vertex, without affecting the Maximum Independent Set (Why?).

Lemma 1.1 *Let T be a rooted tree, with vertex r denoting the root. Let v denote an arbitrary vertex in T; let C_v denote the set of children of v and let G_v denote the set of grandchildren of v. Let $I(v)$ denote the size of the Maximum Independent Set (MIS) of the subtree rooted at vertex v. Then,*

$$I(v) \quad = \quad \max\left((1 + \sum_{u \in G_v} I(u)), \sum_{u \in C_v} I(u)\right)$$

Proof: Let S_v denote the MIS of the tree rooted at vertex v. Observe that either $v \in S_v$ or $v \notin S_v$ and that these cases are mutually exclusive and exhaustive. If $v \in S_v$, then clearly, none of the vertices in C_v can be part of S_v. Further, we can safely combine v with the union of the Maximum Independent Sets of the subtrees rooted at its grandchildren. Similarly, if $v \notin S_v$, we can simply set S_v to be union of the Maximum Independent Sets of the subtrees rooted at its children. □

Although $I(r)$ is expressed in top-down fashion, we compute it in bottom-up fashion, beginning at the leaves of the tree and working our way to the root. It is not hard to see that this algorithm runs in linear time.

□