CIT 5960 - Spring 2023 Final Exam Practice Problems Solutions

1. You are given an undirected graph $G(V, E)$ in adjacency-list representation such that every vertex in G has degree at most k. Design a polynomial-time $(k+1)$-approximation algorithm for the maximum independent set problem on such graphs. Prove that your algorithm achieves the stated approximation ratio, and analyze its running time.

   Solution:

   The Algorithm: We will use a simple greedy process to design our approximation algorithm. Initialize S to be the empty set. Now select any vertex u in V, add it to the set S, and delete it and its neighbors from the set V of vertices. Repeat this process until the set V becomes empty. We output the set S as our approximate independent set.

   Justification: First observe that all vertices in S form an independent set since whenever we add a vertex to S, we discard all vertices adjacent to it from V. Next observe that since the degree of each vertex is bounded by k, whenever we add a vertex to S, we remove at most $(k+1)$ vertices from V. So upon termination, the set S will contain at least $|V|/(k+1)$ vertices. On the other hand, an optimal independent set can have size at most $|V|$, so this is a $(k+1)$-approximation algorithm.

   Runtime Analysis: The algorithm is easily implementable in $O(|V| + |E|)$ time as we can maintain a Boolean array $A[1 \ldots |V|]$ to record for each vertex whether or not it is deleted. Initially, all vertices are marked as undeleted. We process vertices in order $1, 2, 3, \ldots, |V|$ by traversing through A in this order, and whenever we encounter an undeleted vertex v, we include it in S, and scan its adjacency list to mark all its neighbors as deleted.

2. A *source set* of a graph $G(V, E)$ is a subset $X \subseteq V$ with the property that for each vertex $v \in V \setminus X$, there exists a vertex $u \in X$ such that there is a path from u to v in G. For both parts below, a clear description of your algorithm, an analysis of its time complexity, and a justification of the logic behind your approach is sufficient. No detailed proof of correctness is needed.

   (a) Design an $O(m+n)$ time algorithm to find a minimum size source set in an undirected graph (assume adjacency-list representation).

   (b) Design an $O(m+n)$ time algorithm to find a minimum size source set in a directed graph (assume adjacency-list representation).

   Solution:

   (a) The Algorithm: A minimum size source set can be found by finding the connected components of G, and picking exactly one arbitrary vertex from each connected component.

Justification: Any source set X of an undirected graph must include at least one vertex from every connected component of G; otherwise, if no vertex is included from some connected component C, there is no path from any vertex in $G - C$ to any vertex in C, and hence X cannot be a source set. Also, it is sufficient to include one vertex from each connected component of G because there is a path between any two vertices in a single connected component.

Runtime Analysis: Connected components in an undirected graph can be found in $O(m+n)$ time using DFS.

(b) The Algorithm: Compute the component graph G^{SCC} graph of G and let $C_1, C_2, \ldots, C_k$ be the SCCs of G. Let

$X^{SCC} = \{i \mid C_i \text{ has in-degree 0 in } G^{SCC}\}$

Denote the SCCs of zero in-degree as X^{SCC}. Then the minimum size source set of G can be formed by picking exactly one arbitrary vertex from each C_i such that $i \in X^{SCC}$.

Justification: We first claim that in any DAG G, it is necessary and sufficient to include in the minimum source set X every vertex v whose in-degree is 0. To see this, for any vertex v of in-degree 0, there can be no path from any other vertex in G to v so it must be in X. Conversely, for any vertex v of in-degree greater than 1, we can walk backwards until we reach a vertex of in-degree 0 (no cycles will be encountered since G is a DAG).

Now see that by definition G^{SCC} is a DAG. By the claim above, X^{SCC} is the minimum source set of G^{SCC}. Recall that in addition, if a vertex can reach some vertex in a SCC, it can also reach all vertices in the same SCC; it is thus enough to include in X one vertex from each SCC in X^{SCC}. It is also easy to see that it is necessary to include one vertex from each SCC in X^{SCC}: since they have zero in-degree, there's no other way to reach them.

Runtime Analysis: SCCs in a directed graph can be found in $O(m+n)$ using DFS. We determine the SCCs that will result in vertices with in-degree 0 in G^{SCC} by traversing the adjacency list of G in $O(m+n)$ time. Note that for a directed edge $(u, v) \in E$ such that u and v are in different strongly connected components, the in-degree of the vertex in G^{SCC} induced by the component of v is greater than 0. We therefore obtain an $O(m+n)$ time algorithm that computes X.

3. The edge connectivity of an undirected graph is the minimum number k of edges that must be removed to disconnect the graph. For example, the edge connectivity of a tree is 1, and the edge connectivity of a cycle is 2. Note that edge connectivity can never be more than $(n-1)$ since you can always disconnect a graph by removing all edges incident on any one

vertex in the graph.

Suppose you are given a connected undirected graph $G(V, E)$ in the adjacency-list representation. Using Ford-Fulkerson algorithm, design an $O(mn^2)$ time algorithm to determine the edge connectivity of the graph G. A clear description of your algorithm, an analysis of its time complexity, and a justification of the logic behind your approach is sufficient. No detailed proof of correctness is needed.

Solution:

The Algorithm: We first create a flow network G' from G by replacing each undirected edge with two oppositely directed edges, each of capacity 1. We then pick an arbitrary vertex s and compute the minimum s-t cut value between s and every other vertex t by solving the maximum flow problem between s and t (by Max Flow Min Cut theorem, this also gives us the minimum cut value). We then output the value of the smallest s-t cut found in this manner.

Suppose first there is a set E_0 of k edges whose removal disconnects G. Suppose it partitions G into two components A and B. Without loss of generality, assume that our chosen vertex s belongs to the set A. Then for any choice of a vertex $t \in B$, the minimum s-t cut value will be at most k since there is an s-t cut which has only k edges going across it. Conversely, suppose there is an s-t cut (A, B) of size k for some choice of vertex t. Then we claim that the graph G can be disconnected by removing at most k edges. If we remove all edges in G that connect vertices in A to B, we disconnect G and there are only k such edges.

This means the minimum s-t cut value found by the process above is exactly the smallest number of edges whose deletion disconnects the graph G.

Runtime Analysis: Note that each s-t maximum flow computation can be done in $O(mn)$ time using the Ford-Fulkerson algorithm. This follows from the fact that each flow network created above has integer capacities and that the maximum flow is bounded by $(n-1)$ (the source vertex can not have more than $(n-1)$ outgoing edges). Since we do this computation for $(n-1)$ choices of t, the total runtime is $O(mn^2)$.

4. Given a graph and k colors, the k-coloring problem is to assign a color to each of the vertices of the graph so that any two adjacent vertices are assigned different colors (if there exists such a coloring). 3-coloring is known to be NP-complete. Use this to show that 4-coloring is NP-complete.

Solution:

To show that the problem is in NP, consider the certificate as the function $c : V \to \{1, 2, 3, 4\}$

which maps the vertices to colors. This certificate is linear in the size of the input. Then, the verifier checks the following two properties:

(a) c is indeed a function from the domain V to the codomain $\{1, 2, 3, 4\}$

(b) For every edge $(v, u) \in E$, $c(v) \neq c(u)$.

This can be done in polynomial time, hence the problem is in NP. (For full points, one should specify the details of the verifier.)

To show that the problem is NP-hard, we reduce from 3-coloring. Let the given instance of 3-coloring be the graph (V, E). Consider a vertex w that is new (that is not in V). Let $V' = V \cup \{w\}$. Let $E' = E \cup \{(v, w) : v \in V\}$. We show that:

(a) Any 3-coloring of (V, E) defines a 4-coloring of (V', E'). Let the 3-coloring be defined by $c : V \to \{1, 2, 3\}$. Then, $c' : V' \to \{1, 2, 3, 4\}$ can be defined as $c'(v) = c(v)$ if $v \in V$ and $c'(v) = 4$ otherwise. It is easy to see that c' is a valid 4-coloring.

(b) Any 4-coloring of (V', E') defines a 3-coloring of (V, E). Let the 4-coloring be defined by $c : V' \to \{1, 2, 3, 4\}$. Without loss of generality, let $c(w) = 4$. Then, as $(v, w) \in E'$ for all $v \in V$, we know that $c(v) \neq 4$. Therefore, $c(v) \in \{1, 2, 3\}$. Hence $c|_V : V \to \{1, 2, 3\}$ is a 3-coloring of (V, E).

Hence 4-coloring is at least as hard as 3-coloring. As 3-coloring is NP-complete, 4-coloring is NP-hard.