HW 10: Solved Problem

Instructors: Hassanieh, Miller

CS/ECE 374 B: Algorithms & Models of Computation, Spring 2020 Version: **1.0**

1 We have seen some variants of the maximum weight subset problem under constraints in the lecture slides on greedy algorithms. Consider the following problem which generalizes the two variants we saw. The input is a rooted tree $T = (V, E)$. Each edge $e \in E$ has a non-negative integer capacity $u(e)$. For a given $e \in E$ let T_e be the sub-tree of T under the edge e. The items of interest are the leaves L of T. Each leaf v has a non-negative weight $w(v)$. The goal is to find a maximum weight subset $S \subseteq L$ such that the following holds: for each edge e the number of leaves in S that are in the sub-tree T_e is at most $u(e)$; formally $|S \cap L(T_e)| \leq u(e)$ for each $e \in E$ where $L(T_e)$ is the set of leaves of T_e. Describe a greedy algorithm for this problem and prove its correctness.

Solution:

We are given a rooted tree $T = (V, E)$. Assume that the vertices are labeled from $1, \dots, n$ and the edges are labeled $1, \dots, n-1$. In addition, assume that the edge capacities are given in the array $u[1..n-1]$.

A leaf is *feasible* if it can be added to the subset S without violating the capacity constraints. First, the greedy algorithm preprocesses the tree to obtain the leaves and sorts them in decreasing order of weight. Then, the algorithm considers the leaves in that order. For each leaf, it is added to S if it still maintains the feasibility, otherwise the leaf is discarded. It ultimately returns a set S of chosen leaves.

Greedy Algorithm:

- Build a set of the leaves $\{v_1, \dots, v_k\}$ by performing a whatever-first search to find vertices in the tree that do not have children.
- Sort the leaves in decreasing order of weight so that $w(v_1) \geq \cdots \geq w(v_k)$.
- $S \leftarrow \varnothing$.
- For $i \leftarrow 1$ to k:
  - Find the path from v_i to the root either by using whatever-first search or by repeatedly moving towards the parent until the root is found.
  - If **every** edge e in the path has positive capacity $u[e]$, then again traverse the path from v to the root, setting $u[e] \leftarrow u[e] - 1$ for each edge e, and insert v into S.
- Return S

Here, $n = |V|$ and k is the number of leaves of T. Thus, $k \leq n$. Building the set of leaves can be done in $O(n)$ time, and sorting them can be done in $O(n \log n)$ time. The rest of the algorithm can be done in $O(n^2)$ time because there are $O(n)$ loop iterations to consider all the leaves, and each loop iteration does $O(n)$ work to traverse the tree and update the edge capacities. Therefore, the total running time of this algorithm is $\boldsymbol{O(n^2)}$.

If desired, the actual total weight of the maximum weight leaf subset may be calculated by summing $w(v)$ for all v in S in $O(n)$ time.

Proof (Exchange argument and induction): We begin with the exchange argument.

Lemma 10.1. *There exists an optimum solution that contains the first leaf Greedy would pick, g_1.*

Proof: Suppose OPT is any optimum solution for T. If OPT contains g_1, then the proof of the lemma is complete. Suppose OPT does not contain g_1. Note that g_1 must be feasible if no leaves have been chosen yet, for otherwise Greedy would not choose g_1.

Suppose that g_1 is still feasible after all vertices in OPT have been chosen. Then $OPT \cup \{g_1\}$ would also satisfy all the constraints, and its weight is no less than that of OPT.

Suppose that g_1 is no longer feasible after all vertices in OPT have been chosen, and let e be the lowest edge that prevents g_1 from being a feasible edge. Because g_1 was feasible before any leaf is chosen, $u(e)$ must be positive, and let v be one of the $u(e)$ chosen vertices in OPT under T_e. Note that $OPT' = (OPT \setminus \{v\}) \cup \{g_1\}$ satisfies all the edge constraints, because the number of chosen vertices under T_e remains the same, and e is the lowest problematic edge. Because g_1 is the first leaf greedy would pick, $w(g_1) \geq w(v)$. Therefore, the weight of OPT' is no less than that of OPT.

In either case, there exists an optimum solution OPT' that contains g_1. ■

Lemma 10.2. *Greedy returns an optimum solution.*

Proof: For simplicity, we'll say that $G(T)$ means the greedy solution. Now, we apply induction on the number of leaves of T.

Base case: $G(T)$ returns the the empty set if T has no more leaves feasible to pick.

Inductive hypothesis: Suppose for T with at least one leaf that is not the root node and fewer than k leaves total, $G(T)$ is an optimum solution.

Inductive step: Now let T be an arbitrary tree with k leaves. $T' \leftarrow T \setminus \{g_1\}$, where g_1 is the first leaf Greedy would pick, and where T' has had its edge capacites decreased correctly along the path to the root. By the inductive hypothesis, assume $G(T')$ is an optimum solution for T'. By the lemma, suppose there is an optimum solution OPT which also contained g_1. Let $OPT' \leftarrow OPT \setminus \{g_1\}$. Now OPT' is a solution for T' because no edge capacity has been violated (otherwise, OPT could not have picked g_1 before). So $G(T')$ is an *optimum* solution for T', and OPT' is a solution for T'.

Let $|S|$ denote the total weight of S. Then $|G(T')| \geq |OPT'|$. Add the weight of g_1 to both sides. $|G(T')| + |g_1| \geq |OPT'| + |g_1|$. This is the same as saying $|G(T)| \geq |OPT|$ since g_1 would have been the first choice of Greedy made from the original tree and OPT contained the same leaf as well. Therefore $G(T)$ is at least as good as optimum solution OPT, so $G(T)$ is an optimum solution. ■

<u>*Rubric:*</u> Out of 10 points, we would allocate:

- 5 points for the greedy algorithm
- 5 points for proof of correctness

Grading Notes:

- It is okay to return the maximum weight of S rather than the set S itself.
- It is okay to handle (or not handle) the case where $E = \varnothing$ in any way.