The Prize-Collecting Steiner Forest Problem

Kiss Bendegúz
Supervisor: Dr. Király Tamás

December 2024

Introduction

This semester I started to get familiar with the prize-collecting Steiner forest problem. First, we need to take a look at the Steiner forest problem, a more general version of the Steiner tree problem. In this problem, given an undirected graph, $G(V, E)$ with non-negative edge costs, $c : E \mapsto \mathbb{R}_0^+$. We also have a set of pairs of vertices called demands denoted by $\mathcal{D} = \{(v_1, u_1), (v_2, u_2), \dots (v_m, u_m)\}$. Our goal is to find a forest F, which satisfies these demands and minimize its total edge cost. Satisfying a (v_i, u_i) demand means that v_i and u_i are connected in F.

In the prize-collecting version, which we will refer to as PCSF, we have non-negative penalties for all demands, which means we don't need to satisfy all demands, but for every (i, j) pair, where i and j are not connected, we pay a penalty π_{ij}. We define a $\pi : V \times V \mapsto \mathbb{R}_0^+$ penalty function over the vertex pairs as follows. For every (i, j) demand pair $\pi(i, j) = \pi_{ij}$, and for the other pairs, we set the value at 0. Now our goal is to find an F forest and a subset of demands $Q \subseteq \mathcal{D}$, where all the demands not satisfied by F is in Q, and (F, Q) minimizes the following expression:

$$\sum_{e \in F} c_e + \sum_{(i,j) \in Q} \pi_{ij}$$

A 3-Approximation Algorithm for both problems

Since these problems are NP-hard, we use approximation algorithms to solve them. In [1] I studied a 3-approximation algorithm for both problems. For brevity, I am just pointing out the main ideas of these procedures, so I will not get into the implementation techniques.

In the Steiner forest problem, we start with an empty forest for our solution. We will describe the algorithm as an iterative process. At each moment we maintain an FC set, which refers to the connected components. We will consider a set to be active, if it needs to be connected with other sets to satisfy the demands it cuts. We also maintain a set for these sets noted as $ActS$. We consider the edges as curves with length equals to their costs.

In each iteration we do the following: each active set has a distinct color. An active set starts to color its cutting edges (have exactly one endpoint within the set) with its color at the same speed. Accordingly, an edge between two active sets going to be colored twice as fast as the others. We will refer to this coloring procedure as static coloring. We do this until an edge is fully-colored. This happens when the coloring duration of an e edge equals its cost c_e. We add this edge to F. Then, we update FC and $ActS$ by removing the active sets that are connected with this edge and replace them with their union. It is easy to see that F remains a forest during the algorithm. The process ends when there are no remaining active sets.

At the end of the algorithm we delete those edges, which do not contribute to satisfy any demand. The remaining F' forest will be our solution. It can be shown that this is indeed a 3-approximation algorithm.

For PCSF we will use the previous algorithm but we introduce a new coloring scheme called dynamic coloring. In this scheme we assign a distinct color to each $(i, j) \in V \times V$ and we color an S set with an (i, j) pair's color if S cuts the (i, j) demand. This will imitate a static coloring procedure but in each iteration we have to start it over because it relies on the current static coloring. The use of a color assigned to an $(i.j)$ pair is constrained by the penalty π_{ij}.

In each iteration we decide which coloring scheme will be used, based on how long we can color the cutting edges without violating any constraints. We compute these durations separately; we take a minimum and color the edges for this long. After that, we update FC and $ActS$ as seen above. Before the next iteration we decide which active sets can be deactivated (removing them from $ActS$). It means we get a better solution if we pay the penalties for the demands cut by the set. This algorithm also guarantees a 3-approximation solution for this problem.

Future Plans and Work So Far

I studied the 3-approximation algorithm for PCSF and started implementing it in Python. It's almost ready just need some minor corrections. I have several plans to continue this project. Firstly, I want to compare the 3-approximation algorithm with heuristic algorithms using my Python code. Secondly, there is a 2-approximation algorithm described in [1], so I will explore what improvements can be made.

Hivatkozások

[1] Ahmadi, Ali, et al. "2-approximation for prize-collecting Steiner forest." Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). Society for Industrial and Applied Mathematics, 2024.