Reflection on CS170 Project

Weixiao Zhan, Weishi Li

Apr. 2020

Introduction

From lecture/textbook, we learned that there are general three way to solve or approximate an NP in a methodological perspective: Brute search on small data set, Approximate algorithms or Continue optimizing algorithms.
For the min-average-cost dominate tree problem(MDT), we decide to go for the second approach. The first reason is the large or medium graphs are already too big for brute search approach. Plus, the function of finding MDT has poor continuity since the dominate tree can differ so much because of a tiny weight change in some corner cases.
Our approximate algorithm takes on two step. The first step is to find the n min-cost spanning trees of G. Second step is go from a ST, we greedily to using DP to delete some edges to approach MDT.

Generate ST in cost-increasing order

The main reason why we reduce a graph to ST is based on an observation: the MDT are more likely consist of lighter edges. This is vary intuitive since we are optimizing the average pair wise distance. And we are generating minimum n ST instead of just one MST to increasing the probability that the real MDT is a subgraph of some ST we generated. From here, we may find a better DT by remove redundant edges. Plus, if we have more time or compute-power, we can just increase n to reach out more ST, and this leads to improvement in final output.
The genST algorithm is from Sörensen Kenneth and Janssens Gerrit, Generating all spanning trees of a graph in order of increasing cost, in 2000. We implemented this algorithm via python and networkx library.

remove edges to approach MDT

First of all, some notation are defined as follow:

G: a input graph
T: a ST of G
DT: a dominate tree that is a subgraph of T
LV: a leaf vertex of DT
LE(u): the edge that connected leaf vertex u

This step consist of two stage, first is from a T to m DT, second is from m DT to a greedily OPT DT, and return.

1. Generate m DT by deleting the heaviest(sorted by its LE cost) m LV respectively.
2. each DT will generate m derive DT by deleting its heaviest(sorted by its LE cost) and delectable m LV respectively. Now, there are m^2 driven DT, and we sort them by average-pairwist-distance. Then re-apply stage 2 on first m DT, until no more delectable LV or deleting any LV will cause average-pairwist-distance increasing.
3. return the minimum DT of all m DT.

For any given instance of G, we apply this scheme to all its n ST. Each ST generate only 1 DT, and we return the one with min average-pairwist-distance.
Also note that: if we have more time or compute-power, we can increase m to improve the final output.

Parallel

we use multiprocessing library to boost the computation, also divide the computation by graph on Hive machine and two personal laptop to finish computing.

Hyperparameter

In this algorithm, the search depth(n) of each graph and the search width(m) of each ST are hyper parameters.
As of writing, n = 1000, m = 3.