CS 170

Phase 3: Project Reflection
Spring 2020

1 Group Contact Info

Team Name: wouldhaveratherdonethisinmatlab

Name	SID	Email
Benjamin Chang	26452778	benjachang12@berkeley.edu
Kelvin Pang	26162370	kelvinpang@berkeley.edu

2 Algorithm Description

The algorithm ouputs a subgraph T of input graph G that represents an optimal cell tower network obeying the following requirements:

1. T's nodes form a dominating set over the nodes in G
2. T is a tree
3. T has minimal average pairwise distances

This optimization problem is NP-hard, and thus we design an approximation algorithm that finds multiple valid solutions from different approaches. We take the minimum average pairwise distance score over this "ensemble" of solvers and output corresponding tree as the best solution.

The different approaches are outlined below:

1. **Maximally Leafy Tree with Smart Pruning:**

   This approach is where the bulk of our algorithmically interesting contributions lie. We implement the algorithm specified by Lu et al. to approximate a maximum leaf spanning tree in almost linear time with an approximation ratio of 3.[2] Finding the maximum leafy spanning tree came from the understanding that maximizing the number of leaves is equivalent to minimizing height, and thereby the average pairwise distance in the tree. Additionally, more leaves increase the impact of pruning, which can further reduce the cost of the tree. There are of course edge cases where these properties do not hold true, but overall this heuristic is still performs well.

   The algorithm first greedily grows disjoint subtrees from root vertices to create a maximally leafy forest. The authors of the paper suggested an improvement to the approximation ratio could be had by growing the maximally leafy forest by the descending order of the degree of nodes in G. We implement and further extend this by

designing a heuristic to determine the priority in which we select root nodes to grow subtrees from.

$$priority(v) = deg(v) + \frac{\alpha}{average\ edge\ cost\ to\ neighbors(v)}$$

The tunable hyperparameter, α, specifies the following:

- $\alpha = 0$: prioritizes nodes with higher degree
- $\alpha = small$: higher priority to nodes with high degree
- $\alpha = large$: higher priority to nodes with low average edge cost to neighbors

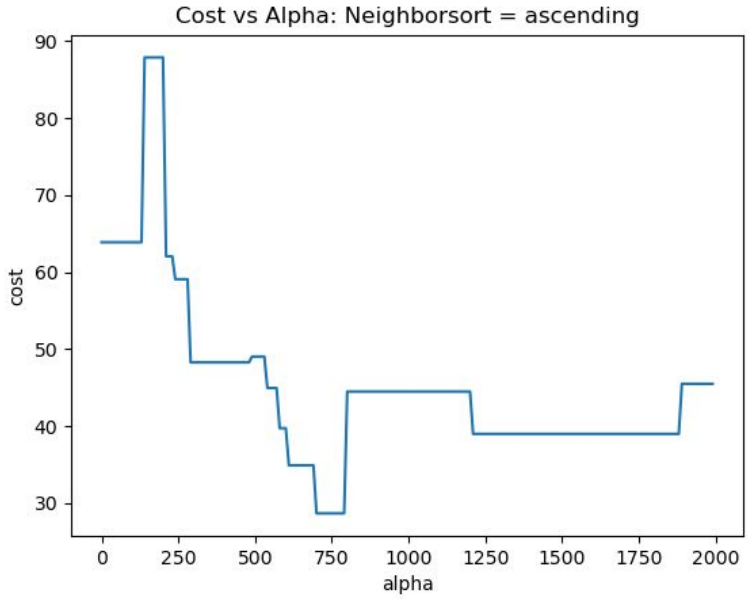

Figure 1: Cost vs α for a single input graph

In our final submission, we let α range from [0, 3000] and tune via validation (shown in Figure 1) for each input graph to produce the optimal leafy forest .

Additionally, we improve upon the author's algorithm by growing the subtrees with neighbors sorted in ascending order of edge costs. Neighbors with low edge costs are more likely to be added to the subtree. The algorithm continues by joining these disjoint subtrees (CCs) with the minimum weight edge via Kruskal's algorithm to form a maximally leafy spanning tree. Finally, the spanning tree is smart pruned. (See note on smart pruning below)

2. **Minimum Spanning Tree with Smart Pruning**

   The minimum spanning tree was computed using networkx's built-in function. The MinST is then smart pruned. (See note on smart pruning below)

3. **Maximum Spanning Tree with Smart Pruning**

   The maximum spanning tree was also computed using networkx's built-in function. The MaxST is then smart pruned. (See note on smart pruning below)

4. **Minimum Vertex Cover Spanning Tree**

The minimum vertex cover was found using networkx's in-built function. This returns a minimum weight vertex cover with an approximation ratio of 2. The set of nodes in the vertex cover were then connected using Kruskal's algorithm to form a spanning tree over the vertex cover.

5. **Minimum Edge Dominating Set Spanning Tree**

   The minimum edge dominating set was found using networkx's in-built function. This returns a minimum cardinality edge dominating set with an approximation ratio of 2. The set of edges in the dominating set were then connected using Kruskal's algorithm to form a spanning tree over the dominating set.

Note: Smart pruning greedily removes leaf nodes from a spanning tree only if its removal improves the average pairwise distance of the tree. Pruning a leaf does not always lower the average pairwise cost. By pruning only if cost is decreased, we guarantee the tree's cost to only improve. Pruning leaves is a safe operation on spanning trees because the result still satisfies the condition that every vertex or its neighbor must be in the tree.

By computing an ensemble of solvers, we leverage the individual advantages that these approaches have over others for different cases of input graphs. Notably, we validate this strategy by running a tally over all 1006 of the input graphs. The best solutions for our final leaderboard submission were distributed as follows:

Maximally Leafy Tree	**Minimum Spanning Tree**	**Maximum Spanning Tree**	**Minimum Vertex Cover**	**Minimum Dominating Set**
559	406	26	15	0

3 Other Attempted Approaches and Ideas

1. **Maximally Leafy Tree with Smart Pruning, Sorted Descending/No Order**

   In the creation of the maximally leafy subtrees, we experimented with the order of neighbors that were chosen to grow the subtrees from. We tried unsorted, random, ascending, and descending order of edge cost to the root. Over all inputs, on average, it was found that ascending order yielded the best performance and thus was chosen for the final implementation. In addition, we explored the effect of smart pruning vs pruning all leaves.

2. **Minimum Weight Dominating Set Spanning Tree**

   Networkx has two different approximation algorithms for dominating set: a min weighted dominating set and a min edge dominating set. The minimum weighted dominating set spanning tree was implemented, but found to produce the exact same solution graph and score as the minimum vertex cover spanning tree, so we implement the min edge dominating set spanning tree instead.

A more detailed change log of our leaderboard submission and offline experiment history can be found below in the appendix. These contain additional information about the different approaches that were either kept, improved upon, or discarded in the final ensemble of solutions.

4 Computational Resources

Computation to generate the outputs were all done on our local machines. In order to speed up runtime, we implemented a parallelized solver to improve the runtime of the individual iterations over α, the root priority heuristic hyperparameter. This improved overall runtime by 4X, on a 12-Core CPU

5 References

[1] Wu, Bang Ye; Lancia, Giuseppe; Bafna, Vineet; Chao, Kun-Mao; Ravi, R.; Tang, Chuan Yi (2000). "A polynomial-time approximation scheme for minimum routing cost spanning trees". *SIAM Journal on Computing*. 29 (3): 761–778

[2] Lu, Hsueh-I & Ravi, Ramamoorthi. (1998). Approximating Maximum Leaf Spanning Trees in Almost Linear Time. Journal of Algorithms. 29. 132-141.10.1006/jagm.1998.0944.

6 Appendix

Submission History:

#	Rank	Average Rank	Average Cost	Notes
1	166/201	132.8121	123.700	Pruned leafy tree, no smart pruning, non-sorted neighbors (Bug: edges don't have weights)
2	156/202	122.8201	115.083	Pruned leafy tree, with smart pruning, non-sorted neighbors (Bugfix: edges now have weights)
3	115/204	93.5457	89.241	Take minimum over: - leafyT pruned (unsorted) - minST & maxST pruned
4	115/204 128/228	92.9404 104.7107	88.951	Take the minimum over: - leafyT pruned (unsorted) - leafyT pruned (sorted ascending) - leafyT pruned (sorted descending) - minST & maxST pruned
5	118/228	98.0099	86.728	Take the minimum over: - leafyT pruned (ascending): alpha=[0:10:100] - minST & maxST pruned Runtime: 608s
6	115/234	95.8529	85.430	Take the minimum over: - leafyT pruned (ascending): alpha=[0:10:500] - minST & maxST pruned

				Runtime: 2553s
7	119/242	98.662	85.0684	Take the minimum over: - leafyT pruned (ascending): alpha=[0:10:2000] - minST & maxST pruned **Runtime: 2921s**
8	138/271	115.2932	84.982	Take the minimum over: - leafyT pruned (ascending): alpha=[0:5:3000] - minST & maxST pruned - minVCST (bug) - minDSST (bug) **Runtime: 7817s**
9	149/286	124.5646	84.837	Take the minimum over: - leafyT pruned (ascending): alpha=[0:5:3000] - minST & maxST pruned - minVCST (bugfix) - minDSST (bugfix) **Runtime: 7870s**

Experiment Results:

#	Total Runtime	Average Cost	Notes
1	80s	89.258	Take the minimum over: - leafyT pruned (sorted ascending) - minST pruned **MinST is chosen 611 out of 1006 inputs (mostly on small/medium inputs)**
2	87s	89.510	Take the minimum over: - leafyT pruned (sorted descending) - minST pruned **Descending order is worse**
3	80s	89.433	Take the minimum over: - leafyT pruned (unsorted) - minST pruned **Sorting has little effect on runtime** **Unsorted is worse --> use ascending order**
4	110s	89.061	Take the minimum over: - leafyT pruned (ascending) - minST & maxST pruned **Adding maxST has little effect on runtime, but slight improvement on score. MaxST is chosen a couple of times** **CONCLUSION: use ascending order and min/maxST**
5	111s	89.061	Take the minimum over: - leafyT pruned (ascending): alpha=[0]

			- minST & maxST pruned **Alpha=0 should produce same exact results as experiment 4 ✔**
6	260s	87.887	Take the minimum over: - leafyT pruned (ascending): alpha=[0,10,20,30] - minST & maxST pruned **Better results, but over 2x increase in runtime (this is expected). Ready for submission 5**
7	137s	89.056	Take the minimum over: - leafyT pruned (ascending): alpha=[0] - minST & maxST pruned - minVCST (bug) **Tiny tiny improvement over experiment 5, minVCST was chosen 3 out of 1006 inputs**
8	142s	89.039	Take the minimum over: - leafyT pruned (ascending): alpha=[0] - minST & maxST pruned - minVCST (bug) - minDSST (bug) **Tiny tiny improvement over experiment 7, minDSST was chosen 3 out of 1006 inputs** **SOLUTION COUNTER:** **LeafyT: 362 MinST: 604 MaxST: 34 VC: 3 DS: 3**
9	151s	88.777	Take the minimum over: - leafyT (ascending): alpha=[0] - minST & maxST pruned - minVCST (bug fix) - minDSST (bug fix) **Improvement over experiment 8, VC and DS bugfix** **VC is now chosen 22 out of 1006 inputs** **SOLUTION COUNTER:** **LeafyT: 355 MinST: 593 MaxST: 34 VC: 22 DS: 2**