Approximating Optimal Treedepth Decompositions using Betweenness Centrality

Miguel Bosch Calvo
Department of Data Science and Knowledge Engineering, Maastricht University, Netherlands

Dominik Jeurissen
Department of Data Science and Knowledge Engineering, Maastricht University, Netherlands

Steven Kelk
Department of Data Science and Knowledge Engineering, Maastricht University, Netherlands

Zhuoer Ma
Department of Data Science and Knowledge Engineering, Maastricht University, Netherlands

Alexander Reisach
Department of Data Science and Knowledge Engineering, Maastricht University, Netherlands

Borislav Slavchev
Department of Data Science and Knowledge Engineering, Maastricht University, Netherlands

Giorgia Carranza Tejada
Department of Data Science and Knowledge Engineering, Maastricht University, Netherlands

Abstract

We've developed an an algorithm for computating treedepth decompositions for large graphs. We use sample based betweenness centrality as a heuristic to recursively remove nodes from a graph to find a near-optimal treedepth decomposition. During computation, we use local and global pruning for early stopping of strictly dominated decompositions.

2012 ACM Subject Classification Design and analysis of algorithms

Keywords and phrases Treedepth, PACE 2020, Heuristic, Betweenness Centrality

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Acknowledgements We want to thank our supervisors Steven Kelk, Georgis Stamoulis, Matus Mihalak, and Gijs Schoenmakers

1 Overview

The algorithm is based on the recursive definition of treedepth: A treedepth decomposition of a connected graph $G = (V, E)$ is a rooted tree $T = (V, E_T)$ that can be obtained in the following recursive procedure: If G has one vertex, then $T = G$. Otherwise pick a vertex $v \in V$ as the root of T, build a treedepth decomposition of each connected component of $G - v$ and add each of these decompositions to T by making its root adjacent to v.

This definition is essentially an algorithm description. The challenge lies in selecting the right vertices at each step in a way that minimizes the depth of the resulting decomposition. To quickly find those vertices, we use a stochastic heuristic, which we will describe in further detail in Section 3. Due to the stochastic heuristic, the algorithm can output solutions with varying quality. Because of that, we generate treedepth decompositions until we reach a time limit. We then return the tree with the smallest depth found.

42nd Conference on Very Important Topics (CVIT 2016).
Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:1–23:3
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

2 Preprocessing

Before solving a graph G, we first remove all chains in G. We define a chain as an path $\{v_1, \ldots, v_n\}$ with a attachment node a that is connected to v_n, and it holds that $deg(a) > 2$. The path can easily be solved by recursively choosing the middle vertex $v_{\lfloor (n+1)/2 \rfloor}$ as root, and solving the left and right side in the same manner. The root of the path's treedepth decomposition is then made adjacent to the attachment node in the original graph's treedepth decomposition. This type of preprocessing was able to remove up to 300.000 vertices from a graph with 2 million nodes and can save a lot of computation time.

3 Betweeness-Centrality Heuristic

We note that splitting a graph into connected components is the basis of reducing the depth of a treedepth decomposition. We also note that the treedepth of a larger graph tends to exceed that of a smaller graph. In combination, this motivates the search for small cut sets that result in balanced graph partitions when looking for an ideal node removal order in treedepth decompositions. Betweenness centrality captures both of these components. Nodes with high betweenness centrality will be a part of many minimum cut sets between different node pairs. For this reason, removing nodes with high betweenness centrality will quickly split the graph in disconnected and roughly balanced components. Calculating exact betweenness centralities is computationally expensive for larger graphs so we rely on sampling-based methods. Despite inferior accuracy, such methods have the advantage of introducing stochasticity to the decompositions. This effectively relaxes the constraint put up by our heuristic and enables us to improve our result by re-computing decompositions multiple times and selecting the one with minimal depth.

4 Pruning

We use two pruning methods to reduce computation time. First, since we create multiple treedepth decompositions, we can use global pruning. This means that we discontinue completing unfinished decompositions when their depth is larger than a previously found solution.

The second pruning method is called local pruning. The idea is to quickly solve remaining connected components that will never increase the tree's depth. First, we have to introduce the naive solution. The naive treedepth decomposition for a graph is a path, where the order of the vertices doesn't matter. Since all vertices are in a parent-child relationship, the naive solution is always a valid treedepth decomposition, with depth equal to the number of vertices in the graph.

Now, if we have a tree with root r, then the tree's depth is equal to the maximum depth of the subtrees connected to r, plus one. That means if we already a subtree with depth d, we can generate the naive solution for connected components that will not exceed depth d. To do this, we compute an upper bound on the treedepth by counting the number of vertices in the connected component, plus the number of ancestors of the connected component.

5 Prioritizing

For large graphs with millions of nodes, the algorithm is often not able to find a single treedepth decomposition in time. Nonetheless, we can still use the unfinished treedepth decomposition by generating the naive solution for the unfinished connected components.

Although the naive solution can quickly solve an unfinished treedepth decomposition, the depth of the solution will be very high. To solve this problem, when the solver has to choose a vertex according to the betweeness-heuristic, we prioritize the connected component with the highest current upper bound on treedepth. This way, applying the naive solution doesn't increase the depth as much as previously. Additionally, this method works well in combination with local pruning.