PACE 2020 Solver Description

Tom C. van der Zanden
Department of Data Analytics and Digitalisation, Maastricht University, The Netherlands
T.vanderZanden@maastrichtuniversity.nl

Abstract

This document provides a description of the solver (doi: 10.5281/zenodo.3893430) that we submitted to the PACE 2020 challenge.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases treedepth, solver

1 Solver description

Basic dynamic programming

The solver uses bottom-up dynamic programming. Given a graph G, a *partial treedepth decomposition of depth d* is simply a treedepth decomposition of depth d of an (induced) subgraph G' of G such that $V(G') \cup Nb(V(G'))$ induces a connected subgraph. The *characteristic of a partial treedepth decomposition is the subgraph G'*. Our algorithm computes, for increasing $d = 1, 2, \ldots, n$, the list of characteristics of partial treedepth decompositions of width at most d.

For $d = 1$ this is simply the set of singleton sets of vertices. To compute the characteristics for depth $d+1$ from the list of characteristics for d, we use two operations: *extend* and *join*.

For the extend operation, we consider, for each characteristic of a partial solution with depth d, all vertices in the separator separating G' from the rest of G (i.e., the vertices in $Nb(V(G')) \setminus V(G')$). For each such vertex v, $G \cup \{v\}$ is a characteristic of a partial solution of depth $d+1$.

Next, for the join operation, we consider characteristics pertaining to disjoint subgraphs G_1, G_2 such that $Nb(V(G_1)) \cap Nb(V(G_2)) \neq \emptyset$. The union of two such characteristics (of depth $\leq d+1$) will be a characteristic for depth $d+1$.

In essence, the solver is a bottom-up application of the following well-known characterization of treedepth [1]:

$$td(G) = \begin{cases} 1, & \text{if } |G| = 1; \\ \min_{v \in V} td(G \setminus v) + 1, & \text{if } G \text{ is connected;} \\ \max_{G' \text{ is a connected component of } G} td(G'), & \text{otherwise.} \end{cases}$$

Pruning

In the spirit of PID dynamic programming [2], we use an upper bound on d that we increase until we find a solution. While at first this may seem redundant as the previously described algorithm generates partial solutions in order of increasing depth, we can actually use the upper bound on d to prune some partial solutions of depth less than d if it is clear they cannot be extended to a solution of depth at most d. We use the following simple lower bound: the lower bound for a partial solution with characteristic G' of depth d' is equal to $d' + |Nb(V(G')) \setminus V(G')|$ since all vertices separating G' from the rest of G will need to appear on a path from the root of the decomposition of G' to the root of the final decomposition of G.

This method of obtaining a lower bound is simple, but in practice can result in a large reduction of the number of partial solutions considered.

Implementation

Our implemenation is written in C#. We represent a partial solution simply as a bitset of the vertices contained in G'. For convenience, we also explicitly store the separator as another bitset.

Generated partial solutions are stored on a queue, from which they are taken to be processed by joining or extending. A hash table is used to filter out duplicate partial solutions. The extend operation is straightforward to implement; for join we simply iterate over all previously generated partial solutions and test whether each one can be extended by the newly generated partial solution. This process could possibly be made more efficient by using for example a trie datastructure to more quickly find suitable candidates.

Source code

The source code for the solver is available on GitHub [3].

References

1 Jaroslav Nešetřil and Patrice Ossona De Mendez. *Sparsity: graphs, structures, and algorithms*, volume 28. Springer Science & Business Media, 2012.

2 Hisao Tamaki. Positive-instance driven dynamic programming for treewidth. *Journal of Combinatorial Optimization*, 37(4):1283–1311, 2019.

3 Tom C. van der Zanden. BasicTreedepthSolver. Accessed 14-06-2020. URL: `https://github.com/TomvdZanden/BasicTreedepthSolver`.