PACE Solver Description: Tweed-Plus: A Subtree-Improving Heuristic Solver for Treedepth

James Trimble
School of Computing Science, University of Glasgow, Scotland, UK
j.trimble.1@research.gla.ac.uk

Abstract

This paper introduces Tweed-Plus, a heuristic solver for the treedepth problem. The solver uses two well-known algorithms to create an initial elimination tree: nested dissection (making use of the Metis library) and the minimum-degree heuristic. After creating an elimination tree of the entire input graph, the solver continues to apply nested dissection and the minimum-degree heuristic to parts of the graph with the aim of replacing subtrees of the elimination tree with alternatives of lower depth.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Algorithm design techniques

Keywords and phrases Treedepth, Elimination Tree, Heuristics

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.35

Supplementary Material DOI of code submitted to PACE Challenge: `https://doi.org/10.5281/zenodo.3881441`. The latest version of the code can be found on GitHub: `https://github.com/jamestrimble/pace2020-treedepth-solvers`.

Funding *James Trimble*: This work was supported by the Engineering and Physical Sciences Research Council (grant number EP/R513222/1).

Acknowledgements Many thanks to the program committee of PACE 2020 and the optil.io team for an enjoyable contest.

1 Introduction

A *treedepth decomposition* of graph $G = (V, E)$ is a rooted forest F with node set V, such that for every edge $\{u, v\} \in E$ we have either that u is an ancestor of v in F or v is an ancestor of u in F. The *depth* of a treedepth decomposition is the maximum number of vertices in a path from the root to a leaf. For example, if G is the graph at the left of Figure 1 then each of the two trees in the figure is a treedepth decomposition of G; the first has depth 5 and the second has depth 4.

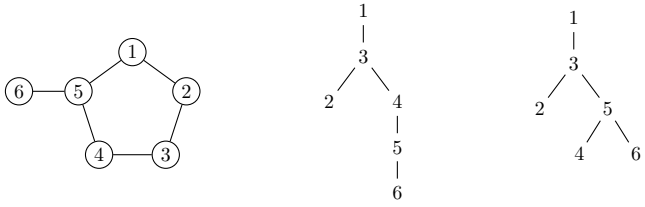

Figure 1 A graph G and two treedepth decompositions of G.

In his PhD dissertation, Fernando Sánchez Villaamil suggests as future work the following strategy for reducing the depth of a treedepth decomposition. "It would be possible to throw away a part of the decomposition and compute a new treedepth decomposition for this part of the graph. ... This suggests a straightforward way of using different heuristics on different parts of the graph." ([5], page 118)

This paper introduces the heuristic solver *Tweed-Plus*, which seeks to find a treedepth decomposition of low depth for a given connected graph. The solver's overall strategy is exactly the one proposed by Sánchez Villaamil, with the parts of the decomposition that are thrown away and replaced being subtrees.

I will use the term *subtree replacement* to denote this process of replacing a subtree of a treedepth decomposition with a new decomposition of the corresponding part of the input graph. To give an example of this process, let G be the graph in Figure 1, and let T_0 and T_1 be the two treedepth decompositions of G shown beside it. To transform T_0 into T_1, the subtree rooted at vertex 4 – which has vertex set $\{4, 5, 6\}$ – is replaced with a different tree on the same set of vertices; moreover, this tree is a treedepth decomposition of the subgraph of G induced by $\{4, 5, 6\}$. In general, the process of subtree replacement is as follows. First, a subtree T rooted at some non-root vertex v is removed from the initial treedepth decomposition. Next, a new forest T' on the vertex set of T is found; this forest must be a treedepth decomposition of the subgraph of G induced by the vertex set of T. Finally, each root of T' is made a child of the vertex that was v's parent in the initial treedepth decomposition.

The algorithm described in this paper depends on the fact, formalised in the following proposition, that the process of subtree replacement maintains the treedepth decomposition property.

▶ **Proposition 1.** *Let F_0 be a treedepth decomposition of a graph G, and let F_1 be the result of applying a subtree replacement to F_0. Then F_1 is also a treedepth decomposition of G.*

Proof. Let $\{u, v\}$ be an edge in G. Without loss of generality, assume that u is an ancestor of v in F_0. There are three cases. We show that in each one, u and v have an ancestor-descendant relationship in F_1; thus, F_1 is a treedepth decomposition of G.

- *Case 1: Neither u nor v is in the replaced part of the decomposition.* The path between u and v in the decomposition is not changed by the subtree replacement. Therefore u remains an ancestor of v in F_1.
- *Case 2: Vertex v is in the replaced part of the decomposition, but u is not.* Let w be the parent in F_0 of the root of the replaced subtree. In F_1, vertex u remains either equal to w or an ancestor of w, since both u and w are in the unchanged part of the decomposition. There must also be a path from w to v in F_1. Therefore u is an ancestor of v in F_1.
- *Case 3: Both u and v are in the replaced part of the decomposition.* By the definition of subtree replacement, the new part of the decomposition is itself a treedepth decomposition of part of G containing u, v, and an edge between these two vertices. Therefore, either u is an ancestor of v in F_1 or vice versa. ◀

If a subtree replacement replaces a subtree with a forest of strictly smaller depth, we call this replacement a *subtree improvement*.

Tweed-Plus makes use of two solvers – which will be referred to as the *sub-solvers* – each of which is itself a heuristic for the treedepth problem. The first sub-solver is a a variant of the well-known *minimum-degree heuristic*; the second is a small wrapper around the Metis library [2]. The overall strategy of the Tweed-Plus solver is to generate an initial treedepth decomposition of the input graph using one of the sub-solvers, then to make repeated additional calls to the sub-solvers with the aim of making subtree improvements.

2 The Sub-Solvers

Recall that each sub-solver called by Tweed-Plus is itself a heuristic algorithm for finding a treedepth decomposition. Both of the sub-solvers produce an *elimination tree* of the input graph. This is valid because every elimination tree is also a treedepth decomposition ([4], chapter 6). Note that the Tweed-Plus algorithm's overall strategy would still produce valid treedepth decompositions even if a different collection of sub-solvers were used, some or all of which produced treedepth decompositions that were not also elimination trees.

To give one of the several equivalent definitions, an elimination tree of connected graph G is any tree created by the following procedure. Visit the vertices of G one by one. On visiting vertex v, carry out the following two steps. First, modify G by adding all of the edges required to make the set of unvisited neighbours of v a clique. Second, find the set of visited neighbours of v that do not yet have a parent in the elimination tree, and make v the parent in the tree of each member of this set.

The first sub-solver uses the well-known minimum-degree heuristic [1], which constructs an elimination tree using the process described in the definition above. At each step, one of the vertices of minimum degree in the current graph is selected at random to be visited. Unlike the standard version of the algorithm, my implementation includes edges incident to previously-visited vertices in the degree calculation. (I have not carried out a rigorous investigation into the effect of this change.) Another small tweak is that on some calls to the minimum-degree algorithm, degrees are divided by a small integer and rounded down, in order to enlarge the set of "minimum degree" vertices that may be chosen.

The second sub-solver calls the Metis library [2] which constructs an elimination tree in a top-down fashion by nested dissection (a recursive process of finding small vertex separators to partition the graph). The Metis library returns an ordering of vertices for the elimination tree; the sub-solver simply constructs an elimination tree using this ordering.

3 The Tweed-Plus Algorithm

Let G be the input graph. The Tweed-Plus algorithm performs the following steps. First, one of the sub-solvers is called to produce an initial treedepth decomposition T of G. (Since G is assumed to be connected, T must be a tree rather than a forest composed of multiple trees.) Following this, the algorithm finds a subtree T' of T that contains a leaf of maximum depth in T; the subtree T' is a candidate for subtree improvement. One of the sub-solvers is then called in an attempt to find a replacement for T' that has lower depth. If such a subtree is found, this is used to replace T' in the overall decomposition T.

The algorithm periodically discards T completely, computes a new initial treedepth decomposition, and again carries out the process of attempted subtree improvements. Whenever the current decomposition has lower depth than any decomposition previously found, this decomposition is recorded as the incumbent best solution.

The algorithm as implemented is intended to be run for 30 minutes, as this is the time limit of the PACE Challenge. Two minutes before this time limit is reached, the best treedepth decomposition found so far is reloaded from memory, and additional subtree improvements are attempted until the time limit is reached. This simple strategy of returning to the best solution found so far for a final, relatively long improvement phase appears to be useful in some cases. For example, it brings the best depth found down from 138 to 135 on PACE Challenge public instance 069; no other solver entered into the challenge gives as good a result on this instance.

There are many small details of the algorithm that have not yet been described: How are candidate subtrees for improvement chosen? How many attempts at improvement are carried out before beginning with a new initial decomposition? When is each sub-solver used? Each of these decisions was made by hand-tuning to the PACE Challenge public instances, and further improvements are without doubt possible. To give a sketch of the choices made: Both sub-solvers are used (in different iterations) for producing an initial decomposition, and both are also used for attempting subtree improvements. Candidate subtrees for improvement always contain the lowest-indexed vertex of maximum depth, and have depth ranging from 1 to one less than the depth of the full treedepth decomposition. On early iterations, subtree improvements are not attempted at all; on later iterations, an increasing number of attempts at subtree improvements are made.

4 Implementation details

The Tweed-Plus algorithm is implemented in C. Two different representations of vertex neighbourhoods are used in an effort to save memory and reduce run time: small adjacency lists (up to 32 vertices) are stored as unsorted lists, while larger adjacency lists are stored as bitsets. Nevertheless, the implementation of the minimum-degree heuristic lacks many of the improvements that have been proposed in the literature [1].

The solver uses code from Nauty 2.6r12 [3] for the bitset data structure and random number generator (the latter of which is based in turn on code by Donald Knuth).[1] Metis 5.1.0 [2] is used to find a nested dissection ordering.[2]

5 Conclusion

This paper has briefly introduced the Tweed-Plus solver, which uses a portfolio of heuristic treedepth algorithms both for constructing an initial solution and for seeking subtree improvements. An interesting direction for future research would be to add several of the leading solvers from the PACE 2020 heuristic track as additional sub-solvers.

References

1 Alan George and Joseph W. H. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Review*, 31(1):1–19, 1989. doi:10.1137/1031001.

2 George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Scientific Computing*, 20(1):359–392, 1998. doi:10.1137/S1064827595287997.

3 Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *J. Symb. Comput.*, 60:94–112, 2014. doi:10.1016/j.jsc.2013.09.003.

4 Jaroslav Nesetril and Patrice Ossona de Mendez. *Sparsity - Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012. doi:10.1007/978-3-642-27875-4.

5 Fernando Sánchez Villaamil. About treedepth and related notions. PhD thesis, RWTH Aachen University, Germany, 2017. URL: https://tcs.rwth-aachen.de/~sanchez/about-treedepth-and-related-notions.pdf.

[1] Nauty is available at http://pallini.di.uniroma1.it/.
[2] Metis is available at http://glaros.dtc.umn.edu/gkhome/metis/metis/overview/.