Software Tool to Calculate the Tree-depth of a Graph

Uddhav Agarwal

Sebastian Ordyniak

COM3610 – Dissertation Project

20/11/2018

The University of Sheffield

This report is submitted in partial fulfilment of the requirement for the degree of Computer Science by Uddhav Agarwal.

All sentences or passages quoted in this report from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). Any illustrations that are not the work of the author of this report have been used with the explicit permission of the originator and are specifically acknowledged. I understand that failure to do so amounts to plagiarism and will be considered grounds for failure in this project and the degree examination as a whole.

-Uddhav Agarwal

Abstract

Computing the tree-depth of a graph is a relatively new topic but has recently seen a spike in theoretical research due to the recent finding which prove its algorithmic advantage over other similar parameters. This preliminary report highlights the main objectives of the project and the several problems faced when calculating the tree-depth decomposition and tree-depth itself. I also cover the significance of calculating the tree-depth for a graph. The final aim of this project towards the end is to develop a software using Python, which implements and computes the tree-depth decomposition to find the tree-depth value for a given graph by various heuristic approaches and already published algorithms. Starting from a basic DFS algorithm, an attempt to implement algorithms that are more complex will be made.

Contents

[Introduction 5](#_Toc531372142)

[Definition and Properties 5](#_Toc531372143)

[About the Project 7](#_Toc531372144)

[Literature Survey 7](#_Toc531372145)

[Requirements and Analysis 9](#_Toc531372146)

[Progress 10](#_Toc531372147)

[Until now 10](#_Toc531372148)

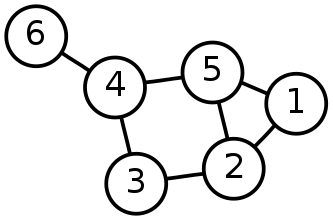
[Future 10](#_Toc531372149)

[References 10](#_Toc531372150)

# Introduction

Graphs can take many forms based on the structure they have such as a complete graph, bipartite graph, cyclic graph, directed graphs, connected graph, etcetera. These structures can be combined further to form many different and more complex structures of graphs. Mathematicians and Computer Scientists alike have been interested in graphs and its invariants due their application in many different fields. Countless problems have come up in last century that such as subgraph isomorphism problem, Hamiltonian path problem, travelling salesman problem, vertex cover problem, etcetera. Often times, to (or at least attempt to) solve these problems, researchers rely on certain pre-processed parameters such as the tree-width, tree decomposition, tree-path or tree-depth. Many of the aforementioned problems are NP-hard or even NP-complete but depending on the characterisation of the graphs computing those parameters is often an NP-hard problem on to itself. This aim of this project is to develop a software tool to compute the tree-depth for a variety of graphs.

## Definition and Properties

Before we look into tree-depth we must first define a graph and its various components. We can define a graph as an ordered pair where is a (finite) set of vertices (also known as nodes) and is a set of 2-subsets (pairs) of or edges.

A tree is an acyclic-connected graph. A forest can be defined as a collection of trees or a graph where the connected components are trees (Supervisor and Chen, 2015). The height of a vertex in a rooted forest is defined as number of nodes in the path from the root (of the component belongs to) to that node and is denoted by . Hence, we can deduce that height of a forest as a whole is the maximum height of the vertices of . Next, we look into the ancestors in a graph. Let be two vertices of a forest . Then is an ancestor of if in , belongs to the path where is connected to the root of . Closure of a rooted forest or is a graph with vertex set and edges such that (Nešetřil and Ossona de Mendez, 2012).

Figure 1. Example of a graph with six vertices and seven edges. Here, V = {1, 2, 3, 4, 5, 6} and E = {{1,2}, {1,5}, {2,5} ,{2,3} ,{5,4}, {3,4}, {4,6}}

Now that we know some basic building blocks of a graph, we can define tree-depth. Tree-depth is considered an equivalent or similar notion to rank function(Nešetřil and Shelah, 2003), the vertex ranking number, the minimum height of an elimination tree (Letters and 1989, no date; Deogun *et al.*, 1994; Bodlaender *et al.*, 1998), etcetera. (Nešetřil and Ossona de Mendez, 2012) define the tree-depth of a graph as the minimum height of a rooted forest such that , where stands for closure of a rooted forest (see Figure 2.)

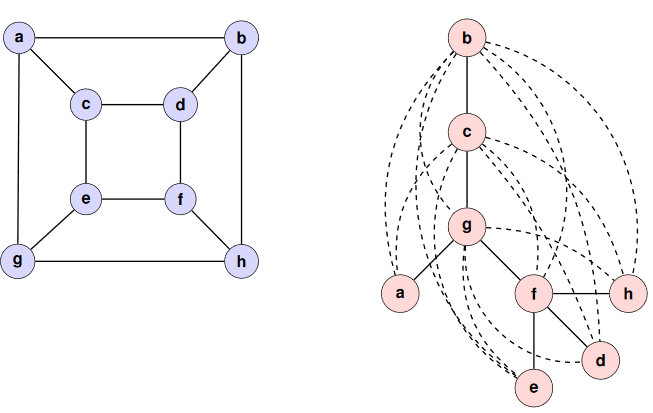


Figure 2. Adopted from (Supervisor and Chen, 2015). The graph G and tree T are on the left and right respectively. The dotted lines in T represent the clos(T). From the definition stated earlier, we know that G ⊆ clos(T) and height(T) = 5, hence td(G) is 5

In addition to the definition above (Nešetřil and Ossona de Mendez, 2012) and (Supervisor and Chen, 2015) provide a similar representation of the tree-depth as a recursive function on the elimination tree of a graph . The elimination tree of a graph with vertex set defined as follows:

* If , then
* If is connected and , then is chosen as the root of the of and an elimination forest is created for . The roots of this elimination forest will be the children of in .
* Otherwise, if is not connected, then is the union of the elimination forest of each component of

(Nešetřil and Ossona de Mendez, 2012) proves the lemma that tree-depth of a graph is the minimum height of an elimination tree for and provides the following recursive formula.

## About the Project

This preliminary knowledge about graphs is important to understand the aims and scope of this project. The aim of this project is to develop a software tool using Python and various libraries to obtain a tree-depth decomposition using a simple depth-first search in the graph and then look into the possibility of improving the result by implementing already proposed algorithms.

The language that the software will be written in is Python, as it provides a versatile coding environment, Python is a fast for prototyping, it has a vast array of libraries and also because of my previous experience coding for my other personal and university projects. The libraries and packages used are:

* NetworkX (Hagberg hagberg *et al.*, 2008)
* IPython (Sos, 1989)
* Matplotlib (Hunter, 2007)
* SciPy (Oliphant, 2007)

IPython, Matplotlib and SciPy are the more commonly used libraries and their applications are known and are expected to be known by the reader. NetworkX is a python library used to study graphs and networks, which is the main focus of this report. NetworkX streamlines the process of graph representation and can read/write various graph formats. The nodes in NetworkX can store any Python object and the edges can also store arbitrary data.

For the graph data, the ‘famous’ DIMACS graph format will be used. This format for the undirected graphs was defined by DIMACS (Centre for Discrete Mathematics and Theoretical Computer Science) and has been used as a standard format for undirected graphs. (DIMACS (Center for Discrete Mathematics and Theoretical Computer Science), no date)

In the Chapter 2 ahead, I will discuss the review of various journals and papers that talk about possible heuristics to improve the time complexity of obtaining the tree-depth decomposition and hence finding the tree-depth of the graph. In chapter 3, I will provide the details of my aims and objectives with the project, which includes the parts covered by the project and the scientific evaluation. In chapter 4, I will discuss my progress so far and briefly describe my roadmap. Chapter 5 will be a conclusion of this preliminary report followed by the bibliography.

# Literature Survey

Tree-depth is related to two other parameters of a graph, tree-width and path-width (Fomin, Giannopoulou and Pilipczuk, 2015). A graph with a bounded tree-width, can enable a number of NP-hard problems solvable in polynomial time as very famously theorised in Courcelle’s Theorem. Courcelle’s Theorem tells us that all problems expressible in MSO (monadic second-order) are solvable in linear time on graphs of bounded treewidth (Arnborg *et al.*, no date; Courcelle, 1989). But as proven by (Gutin, Jones and Wahlström, 2015) that in problems like the Mixed Chinese Postman Problem, which is a W[1]-hard when parametrised by tree-width, is FPT when parametrised by tree-depth. This also tells us that graphs with bounded tree-depth are minor closed, which means that every minor of a graph in F is also in F. Hence, as per (Robertson and Seymour, 1995) Graph Minor Theorem, tree-depth closed by a minor are characterised by a finite set of minors. Mathematicians and Computer Scientists have also found tree-depth to be the better parameter for analysis when compared to tree-width in context of counting perfect matchings using little space (Fürer and Yu, 2017) and when finding the space complexity of deciding MSO formulas on graph of bounded treewidth (Elberfeld, 2012).

One of the approaches to finding the tree-depth of a graph is to compute the tree-depth decomposition. Once we have the tree-depth decomposition, calculating the tree-depth becomes trivial as tree-depth is the minimum height of any tree-depth decomposition of a graph. But computing the tree-depth decomposition itself is an NP-hard problem and because the research on tree decomposition has mostly been theoretical, there is a lack of practical implementation of any algorithms. But, if the input graph is trivial such as a binary tree, tree-depth also admits polynomial time algorithms for specific graph classes. (Reidl *et al.*, 2014) has proposed an algorithm which – given an n-vertex graph, a tree decomposition of the graph of width w and an integer – decides whether the tree-depth of the graph is at most t, in time . They introduce a notion of partial decomposition to represent the tree-depth decomposition that will have the same height as the tree-depth decomposition they represent. Partial decomposition is a triple such that is a forest of rooted trees, is and is a height function in which for two nodes where is an ancestor of , . The algorithm creates a table of partial decomposition and then for each operation of the algorithm picks a set(s) of partial decomposition to check if the tree-depth of the given graph is at most integer .

In addition to the beforementioned algorithm (Reidl *et al.*, 2014) also provided two algorithms based on their previous algorithm, which do not require a tree decomposition as part of their input, a ‘simple algorithm’ and a ‘fast algorithm’ which decided the tree-depth in and time respectively. It is important to note that the algorithms decide whether the for a fixed . They also answer an open question posed by (Nešetřil and Ossona de Mendez, 2012) as to whether deciding three-depth admits an algorithm with a linear running time (for every fixed ) that does not rely on Courcelle’s Theorem or other heavy machinery.

Recently, (Ganian *et al.*, 2019) have shown very strong results when computing the tree-depth decomposition based on encoding the methods to calculate the tree-decomposition to the propositional satisfiability problem (SAT). There has been relatively fast SAT encoding for treewidth based on the characterisation of treewidth in terms of elimination orderings (Samer and Veith, 2009) and is one of the most efficient methods to calculate the tree decomposition. (Ganian *et al.*, 2019) use a characterisation that is based on sequences of partitions of the vertex set where upon satisfying assignment of formula , where G is the graph and , in linear time in terms of the number of variables of can be used to construct a tree-depth decomposition. The results are inline and depending on different width measures sometimes even surpasses the currently best SAT-encoding.

Finding the tree-depth for a graph is an NP-hard problem. (Bodlaender *et al.*, 1995) has given an approximation for a graph with n nodes, the tree-depth can be calculated within a factor of O(log2n). They point out that the approximation depends on the size of the separators in the graph. It should be noted that the (Bodlaender *et al.*, 1995) approximation is for the minimum elimination tree height which is a similar notion to that of tree-depth.

(Fomin, Giannopoulou and Pilipczuk, 2015) gives an exact exponential time algorithm for computing the tree-depth for a graph with n vertices in ﻿O∗(2n) time. The algorithm is based on combinatorial results revealing the structure of minimal rooted trees whose closures contain . Starting with the formula devised by (Nešetřil and Ossona de Mendez, 2012), employing a dynamic programming yields the time complexity of O∗(2n). But, as the number of subsets of V(G) inducing connected subgraphs can be as large as O(2n), they reduce the complexity further by pruning the space of states. Space of states is defines as for some

The space of states is stored in the algorithm as a collection of binary vectors of length n in a prefix tree, so as a result duplicates can be avoided when constructing . Thus, they deduce the following pruned programming algorithm that for every computes value

The time complexity of Algorithm (Fomin, Giannopoulou and Pilipczuk, 2015, p. 214) is slightly quicker when compared to the previous rendition i.e. before the pruning of the space of states. It runs in time. The algorithm by (Fomin, Giannopoulou and Pilipczuk, 2015) might prove to be a good starting point.

# Requirements and Analysis

The main objective for the final project is to calculate the tree-depth decomposition for given graph in the DIMACS graph format and by extension calculate the tree-depth for the given graph. As a first step, I will first obtain a depth-first search tree for a graph. As the DFS tree of a graph is a tree-depth decomposition albeit non-optimal, the height of which is a -approximation of the tree-depth . I will then look into heuristics approaches to improve the results or consider the algorithms mentioned above in Chapter 2. Another possibility is to start with the simple algorithm mentioned in (Fomin, Giannopoulou and Pilipczuk, 2015) and then implement the improvements suggested in the paper.

The software tool will not have a GUI and the user will have to interact using the Terminal window. When running the software tool, the user will have to specify the file containing the DIMACS file containing graph data. Depending on the algorithms used, the user might have the option to specify which algorithm to use. The program should return the tree decomposition that was attained with the resulting tree-depth. The program should also display a visual representation of the input graph and the tree-depth decomposition.

Due to the scientific nature of the project, evaluation is a key part. I plan to test and compare the running time (CPU time) of the program with random graphs that were obtained from the DIMACS website and also various standard graphs such as complete-bipartite graphs , cyclic graphs , complete graphs , binary trees of varying sizes. The results from the experiments should help verify and compare different heuristic approaches and also the algorithm.

# Progress

## Until now

I spent a good part of the initial stages getting used to the very theoretical nature of the topic. Although I have come across data structures such as graphs and trees in the past as part of my degree, graph theory has proven to be a bit of a challenge due to my learning disability but at the same time, finally getting used to it was one of the very rewarding experience I have had. I have also started writing the code to calculate the DFS tree for a given graph.

## Future

I plan to research more published journals and articles to get a better understanding of the topic and also find other practical implementations of various algorithms for tree-depth and related topics.

# References

Arnborg, S. *et al.* (no date) ‘Easy problems for tree-decomposable graphs’, *Citeseer*. Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.12.2544&rep=rep1&type=pdf (Accessed: 29 November 2018).

Bodlaender, H. L. *et al.* (1995) ‘Approximating treewidth, pathwidth, frontsize, and shortest elimination tree’, *Journal of Algorithms*, 18(2), pp. 238–255. doi: 10.1006/jagm.1995.1009.

Bodlaender, H. L. *et al.* (1998) ‘Rankings of Graphs’, *SIAM Journal on Discrete Mathematics*, 11(1), p. 168. doi: 10.1137/S0895480195282550.

Courcelle, B. (1989) ‘The monadic second-order logic of graphs: Definable sets of finite graphs’, in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pp. 30–53. doi: 10.1007/3-540-50728-0\_34.

Deogun, J. S. *et al.* (1994) ‘On vertex ranking for permutation and other graphs’, in, pp. 747–758. doi: 10.1007/3-540-57785-8\_187.

DIMACS (Center for Discrete Mathematics and Theoretical Computer Science) (no date) *DIMACS graph definition*. Available at: http://lcs.ios.ac.cn/~caisw/Resource/about\_DIMACS\_graph\_format.txt (Accessed: 28 November 2018).

Elberfeld, M. (2012) ‘Space and Circuit Complexity of Monadic Second-Order Definable Problems on Tree-Decomposable Structures Michael Elberfeld’. Available at: http://www.lsv.ens-cachan.fr/Events/fmt2012/SLIDES/michaelelberfeld.pdf (Accessed: 30 November 2018).

Fomin, F. V., Giannopoulou, A. C. and Pilipczuk, M. (2015) ‘Computing Tree-Depth Faster Than 2n’, *Algorithmica*, 73(1), pp. 202–216. doi: 10.1007/s00453-014-9914-4.

Fürer, M. and Yu, H. (2017) ‘Space Saving by Dynamic Algebraization Based on Tree-Depth’, *Theory of Computing Systems*, 61(2), pp. 283–304. doi: 10.1007/s00224-017-9751-3.

Ganian, R. *et al.* (2019) ‘SAT-Encodings for Treecut Width and Treedepth’, in Kobourov, S. and Meyerhenke, H. (eds) *Proceedings of {ALENEX} 2019, the 21st Workshop on Algorithm Engineering and Experiments*.

Gutin, G., Jones, M. and Wahlström, M. (2015) ‘Structural parameterizations of the mixed Chinese postman problem’, in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pp. 668–679. doi: 10.1007/978-3-662-48350-3\_56.

Hagberg hagberg, A. A. *et al.* (2008) ‘<Exploring Network Structure, Dynamics, and Function using NetworkX.pdf>’. Available at: http://conference.scipy.org/proceedings/SciPy2008/paper\_2.

Hunter, J. D. (2007) ‘Matplotlib: A 2D graphics environment’, *Computing in Science and Engineering*, 9(3), pp. 99–104. doi: 10.1109/MCSE.2007.55.

Letters, A. S.-I. P. and 1989, U. (no date) ‘Optimal node ranking of trees in linear time’, *Elsevier*. Available at: https://www.sciencedirect.com/science/article/pii/0020019089901610 (Accessed: 30 November 2018).

Nešetřil, J. and Ossona de Mendez, P. (2012) *Sparsity*. Berlin, Heidelberg: Springer Berlin Heidelberg (Algorithms and Combinatorics). doi: 10.1007/978-3-642-27875-4.

Nešetřil, J. and Shelah, S. (2003) ‘On the order of countable graphs’, *European Journal of Combinatorics*, 24(6), pp. 649–663. doi: 10.1016/S0195-6698(03)00064-7.

Oliphant, T. E. (2007) ‘Python for scientific computing’, *Computing in Science and Engineering*, 9(3), pp. 10–20. doi: 10.1109/MCSE.2007.58.

Reidl, F. *et al.* (2014) ‘A Faster Parameterized Algorithm for Treedepth’, in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pp. 931–942. doi: 10.1007/978-3-662-43948-7\_77.

Robertson, N. and Seymour, P. D. (1995) ‘Graph Minors .XIII. The Disjoint Paths Problem’, *Journal of Combinatorial Theory, Series B*, 63(1), pp. 65–110. doi: 10.1006/jctb.1995.1006.

Samer, M. and Veith, H. (2009) ‘Encoding Treewidth into SAT’, in. Springer, Berlin, Heidelberg, pp. 45–50. doi: 10.1007/978-3-642-02777-2\_6.

Sos, T. A. (1989) ‘Case study: Embolization of a pulmonary arteriovenous malformation with the aid of the DVI-S’, *MedicaMundi*, 34(3), pp. 129–132. doi: 10.1109/mcse.2007.53.

Supervisor, M. and Chen, H. (2015) *About Tree-Depth*.