

ToTo: An open database for computation, storage and retrieval of tree decompositions

Rim van Wersch^{a,*}, Steven Kelk^{a,*}

^a*Department of Data Science and Knowledge Engineering (DKE), Maastricht University, P.O. Box 616, 6200 MD Maastricht, The Netherlands*

Abstract

Many NP-hard problems on graphs become tractable on graphs of low treewidth, but the corresponding algorithms require access to a tree decomposition of the graph of low (ideally, minimum) width. Unfortunately computation of treewidth is itself NP-hard and a wide variety of exact, heuristic and approximation algorithms have been proposed for this problem. To support this ongoing research we present here ToTo, an open graph database for computation, storage and rapid retrieval of tree decompositions. We hope that the database will become both a central repository for important graphs and benchmark datasets and extend the use of treewidth beyond the usual communities: the database and associated algorithms can be accessed via a web browser and do not require installation of any specialist software.

Keywords: Treewidth, Tree decomposition, Graphs, Database, Algorithms.

1. Introduction

At the heart of modern algorithmic graph theory is the parameter *treewidth*, which is defined as follows. Given an undirected graph $G = (V, E)$, a *bag* is simply a subset of V . A *tree decomposition* of G consists of a tree $T_G = (V(T_G), E(T_G))$ where $V(T_G)$ is a collection of bags such that the following holds: (1) every vertex of V is in at least one bag; (2) for each edge $\{u, v\} \in E$, there exists some bag that contains both u and v ; (3) for each vertex $u \in V$, the bags that contain u induce a connected subtree of T_G . The *width* of a tree decomposition is equal to the cardinality of its largest bag, minus 1. The *treewidth* of a graph G , denoted $tw(G)$, is equal to the minimum width, ranging over all possible tree decompositions of G [12].

The intuition behind this rather technical definition is that treewidth measures, in an algorithmic sense, how far a graph is from being a tree. Its importance stems from the fact that very many NP-hard optimization problems become (fixed parameter) tractable on graphs of bounded treewidth, by applying dynamic programming over low-width tree decompositions [9]. Unfortunately, computing minimum-width tree decompositions (i.e. treewidth) is itself NP-hard [1] and software implementations often require expert knowledge to set up and operate.

To address this situation, we describe here our open database ToTo. Users can query whether the database already has tree decompositions for a given graph, upload improved tree decompositions for existing graphs, or use the embedded exact and heuristic algorithms to generate tree decompositions on the fly (which are then dynamically added to the database). Graphs can be submitted, and tree decompositions downloaded, in a variety of formats. The platform can be accessed via its web interface, which supports visualisation of tree decompositions and requires no installation of any specialist software, or from popular programming languages such as Java.

Although the concept of a dynamic, lazily populated treewidth database is new, it is of course not the first graph database *per se*: a number of other graph databases, tailored to different pur-

*Corresponding authors.

poses, have been implemented and published in recent years. Examples include a larger database for the storage and retrieval of isomorphisms [10] and a smaller database of graphs with particularly interesting invariants [8]. In addition to these databases which differ in scope, size and application, there are various benchmark graph collections available that are commonly used in treewidth computations and by the combinatorial optimization community more generally. The specialized DIMACS coloring instances [15] are typically used for comparative studies, while generic collections [17] provide more exhaustive graph listings. However, the results of computations on these collections are typically only reported in individual articles and are not available in a centralised repository, thereby lacking the query facilities (and the comparative context) a database would provide. To support the centralization process we have already uploaded a number of these datasets to ToTo.

ToTo can be accessed at <http://treedecompositions.com>. Note that, at the present time, the database only stores graphs with up to 150 vertices. If there is sufficient demand this can be expanded to accommodate larger graphs¹, but we remark that graphs of up to 150 vertices are often already beyond the reach of existing exact algorithms for treewidth [4]. Note also that the embedded heuristics can nevertheless still be applied to much larger graphs to obtain bounds and tree decompositions. The only difference is that the generated results are not stored in the database.

In the remainder of this note we describe the main features of ToTo. In the appendix we have provided a number of screen captures highlighting its various functions and visualizations.

2. Main features

2.1. Integrated algorithms for computation of treewidth

One of the most accessible and comprehensive studies focusing on the practical computation of tree decompositions (as opposed to “theoretical” algorithms, see e.g. [3]) remains *libtw* [13]. It implements a library with a large number of heuristic algorithms for the computation of upper and lower bounds for treewidth as well as several exact algorithms. There are many heuristic algorithms available, but most recent developments are primarily variations on and refinements of long-standing algorithmic strategies (often leveraging the link between treewidth and *chordalizations*) [5, 6]. Hence, although *libtw* does not incorporate the very latest refinements (see e.g. [5] for more recent examples), it still gives a good overall picture of how the core heuristics perform. Exact solutions by contrast are less widespread and all are susceptible to the exponential slow-down inherent in solving NP-hard problems exactly. Although various new developments have been proposed and implemented since treewidth computation started in earnest with the early QuickTree algorithm [19], the QuickBB branch and bound algorithm [14] arguably remains the baseline tool for standalone exact computations. Based on the above analyses and with a view to seeking a good balance between the quality of the bounds produced, execution time and memory requirements, we have incorporated the following three algorithms within ToTo:

- The polynomial-time Maximum Minimum Degree with Least Common Neighbor contraction strategy (*MMD+LC*) heuristic for computation of *lower bounds* ([7, 6]);
- The polynomial-time *GreedyMinFill* heuristic to compute *upper bounds* ([16, 5]);
- The *QuickBB* branch-and-bound algorithm for exact computation of treewidth ([14]).

¹The current limitation exists primarily for reasons of convenience. An URL for a GET request has an effective limit of 2048 characters, which is sufficient to accommodate graph identifiers up to about 150 vertices. Graphs with more vertices can be supported, but this would require a POST request which is far less convenient to access for the user, particularly when accessing the database from the API.

Note that the *QuickBB* runs client-side in the user’s web-browser to avoid overloading the server, while the other two algorithms run server-side.

The treewidth of a graph is lazily evaluated on demand in response to a web query from a user, returning the result from the database if it’s available or computing a fast heuristic result on the server if it’s not. This trusted heuristic result is then also stored, organically populating the database with more and more graphs and reliable data on lower and upper bounds. If the user chooses to execute the computationally intensive *QuickBB* algorithm then the results of this algorithm can also be stored in the database. (In this way users can use, if desired, idle-time on their computers to help improve the quality of the tree decompositions in the database.)

2.2. *Small and nice tree decompositions.*

A tree decomposition is called *small* if no bag is a subset of another (equivalently: no bag is a subset of one of its neighbours). Since a *small* decomposition is a more concise representation and any decrease in the number of bags is useful for reducing storage and bandwidth usage, TOTO strictly stores and returns *small* decompositions. Furthermore the design of dynamic programming algorithms operating on tree decompositions is often simplified if one has access to a so-called *nice* tree decomposition (see e.g. [2]). Moreover, nice tree decompositions do not sacrifice optimality. For this reason the TOTO front-end automatically transforms the tree decompositions produced by its internal algorithms into nice tree decompositions, without raising their width.

2.3. *Isomorphism handling.*

TOTO uses server-side tools based on the well-known Nauty package [18] to generate canonical *graph6* identifiers for graphs queried by the user (if two graphs are isomorphic, they have the same canonical *graph6* identifier). These identifiers are used internally to store and access tree decompositions on the canonical isomorphism of any queried graph, allowing results to be shared across all isomorphic graphs. This leads to an interesting technicality. Tree decompositions stored on the canonical isomorphism are expressed in terms of the canonical graph labelling. This means that a tree decomposition retrieved from the database will likely refer to different vertex labels than the graph that is being queried. To address this, we again employ Nauty-based tools to obtain the vertex mapping between the canonical labelling and the target labelling, which is then used to translate the stored tree decomposition into the labelling of the queried graph. Note that, if the upper-bound heuristic computes a tree decomposition no worse than that already stored in the database, the output of the heuristic is used to avoid the second isomorphism calculation and to lessen the load on the server.

2.4. *Support for multiple input and output formats and submission validation.*

To include the benchmark graph collections typically used in treewidth algorithm evaluations as well as more generic exhaustive graph listings, support was implemented for various graph formats. Internally the database and service employs the *graph6* format discussed earlier, but transparent imports from adjacency matrices and the DIMACS [15] and PACE [11] formats were also added. Decompositions may be downloaded in a general purpose JSON format, while the PACE *.td* format was adopted for accepting uploaded submissions of tree decompositions. This syntactically rigid format is critical in maintaining the integrity of the database: TOTO checks that uploaded tree decompositions are in fact valid tree decompositions of the graph in question. Valid decompositions that improve upon the heuristic results are then stored as a new best upper bound for the graph. Although a valid decomposition certifies that a reported upper bound is valid, it cannot guarantee that a bound reported as an exact solution is in fact optimal, unless it matches a known lower bound. As such, client submissions are regarded as ‘votes’ on whether a given value is optimal.

2.5. Full web interface and back-end access via common programming languages.

All features of the database can be accessed via the web interface, so there is no need for the user to install any software. (The client-side algorithms run in Javascript inside the browser of the user). The web interface includes a number of visualisation tools which draw graphs and tree decompositions and, for the implementation of the QuickBB algorithm, summarize the width and depth of the branch-and-bound search tree. The database also has a small API which allows access via popular programming languages such as Java and PHP; a number of examples are included on the ToTo website.

2.6. Auxiliary statistics.

Via the user-interface the user can ask for lists of graphs in which the gap between the best-known lower and upper bounds falls within a certain range. This allows the developers of new treewidth software to quickly identify graphs for which exact computation of treewidth seems challenging.

3. Conclusion

We hope that ToTo will become a central repository for the storage, retrieval and incremental improvement of tree decompositions and the first port of call for users wishing to compute treewidth without installing specialist software. Depending on demand we entertain the possibility of expanding in the near future the storage capacity and server-side processing power of the database. Scientifically ToTo offers a number of exciting future directions for further development. For example, new and exact algorithms can be incorporated as and when they become available. Another option is that, when the load on the server is quiet, the database can be automatically “mined” (e.g. for subgraphs/supergraphs/minors) in an attempt to tighten the best known upper and bounds for particularly difficult graphs.

Bibliography

- [1] Stefan Arnborg, Derek G Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k -tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.
- [2] Hans L Bodlaender. Treewidth: Algorithmic techniques and results. In *Proceedings 22nd International Symposium on Mathematical Foundations of Computer Science, MFCS 1997*, volume 1295 of *LNCS*, pages 19–36. Springer, 1997.
- [3] Hans L Bodlaender, Pål Grnås Drange, Markus S Dregi, Fedor V Fomin, Daniel Lokshtanov, and Michal Pilipczuk. A $c^k n$ 5-approximation algorithm for treewidth. *SIAM Journal on Computing*, 45(2):317–378, 2016.
- [4] Hans L Bodlaender, Fedor V Fomin, Arie MCA Koster, Dieter Kratsch, and Dimitrios M Thilikos. On exact algorithms for treewidth. *ACM Transactions on Algorithms (TALG)*, 9(1):12, 2012.
- [5] Hans L Bodlaender and Arie MCA Koster. Treewidth computations i. upper bounds. *Information and Computation*, 208(3):259–275, 2010.
- [6] Hans L Bodlaender and Arie MCA Koster. Treewidth computations ii. lower bounds. *Information and Computation*, 209(7):1103–1119, 2011.
- [7] Hans L Bodlaender, Arie MCA Koster, and Thomas Wolle. Contraction and treewidth lower bounds. In *Proceedings 12th Annual European Symposium on Algorithms, ESA 2004*, volume 3221 of *LNCS*, pages 628–639. Springer, 2004.
- [8] Gunnar Brinkmann, Kris Coolsaet, Jan Goedgebeur, and Hadrien Melot. House of graphs: A database of interesting graphs. *Discrete Applied Mathematics*, 161(1):311–314, 2013.

- [9] Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 4. Springer, 2015.
- [10] Massimo De Santo, Pasquale Foggia, Carlo Sansone, and Mario Vento. A large database of graphs and its use for benchmarking graph isomorphism algorithms. *Pattern Recognition Letters*, 24(8):1067–1079, 2003.
- [11] Holger Dell. The 1st parameterized algorithms and computational experiments challenge, 2016. See <https://pacechallenge.wordpress.com/>.
- [12] Reinhard Diestel. *Graph Theory (Graduate Texts in Mathematics)*. Springer, August 2005.
- [13] Thomas van Dijk, Jan-Pieter van den Heuvel, and Wouter Slob. Computing treewidth with libtw. Technical report, Utrecht University, 2006.
- [14] Vibhav Gogate and Rina Dechter. A complete anytime algorithm for treewidth. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence, UAI 2004*, pages 201–208. AUAI Press, 2004.
- [15] David S Johnson and Michael Trick. Graph coloring instances, 1996. See <http://mat.gsia.cmu.edu/COLOR/instances.html>.
- [16] Arie MCA Koster, Hans L Bodlaender, and Stan PM van Hoesel. Treewidth: Computational experiments. *Electronic Notes in Discrete Mathematics*, (8):54–57, 2001.
- [17] Brendan D McKay. Combinatorial data. See, <http://users.cecs.anu.edu.au/~bdm/>.
- [18] Brendan D McKay and Adolfo Piperno. Practical graph isomorphism, ii. *Journal of Symbolic Computation*, 60:94–112, 2014.
- [19] Kirill Shoikhet and Dan Geiger. A practical algorithm for finding optimal triangulations. In *Proceedings of the fourteenth national conference on artificial intelligence and ninth conference on Innovative applications of artificial intelligence, AAAI 1997*, pages 185–190. AAAI Press, 1997.

Appendix A. Selected screen captures from the ToTo web front-end

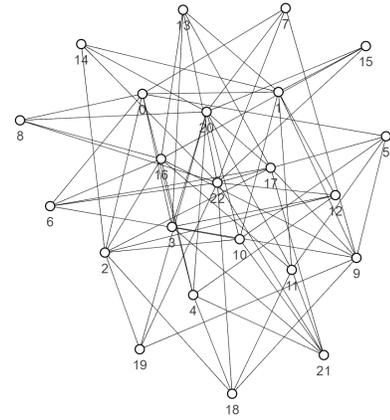
T Treewidth and decompositions

23 vertices 71 edges

Treewidth bounds

▼ Base upper bound	11	
▼ Reported upper bound	10	▼ 1 ✓ 1
— Exact solution	-	
▲ Base lower bound	8	

Help improve this »



Current best decomposition ⌵ ⌴

Nice tree decomposition ⌵ ⌴

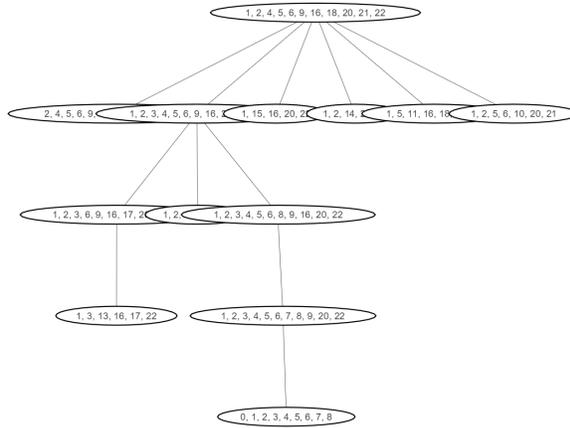


Figure A.1: Result visualization, here for the MYCIEL4 graph, including *small* and *nice* tree decompositions. Note that both tree decompositions may be downloaded in a generic JSON format and the TD format from the PACE16 competition.

Look for graphs

With at least and at most vertices

Return at most ordered

And a base bounds gap between and

Only return graphs without any submitted results so far

[Find me some graphs »](#)

Query result CSV DB

Graph6	Vertices	Edges	Lower bound	Upper bound	Gap	Submitted	Actions
U~ tuz~ v	22	148	11	12	1	12	
V????????	23	71	8	11	3	10	
VsaCCA?TRK	23	71	8	11	3	-	
X~ FKmnpFC	25	160	12	18	6	18	
Y????????	26	72	5	8	3	8	
YJGg?CB?wF	26	78	6	7	1	7	

Figure A.2: A database query result, showing how graphs and tree decompositions with various properties may be retrieved for further inspection and analysis.