ARTICLE IN PRESS

Discrete Applied Mathematics ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

Communication

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

Rim van Wersch *, Steven Kelk *

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

ARTICLE INFO

Article history:
Received 10 August 2016
Received in revised form 7 September 2016
Accepted 10 September 2016
Available online xxxx
Communicated by Endre Boros

Keywords:
Treewidth
Tree decomposition
Graphs
Database
Algorithms

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.

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 visualization of tree decompositions and requires no installation of any specialist software, or from popular programming languages such as Java.

* Corresponding authors.
E-mail addresses: rim@treedecompositions.com (R. van Wersch), steven.kelk@maastrichtuniversity.nl (S. Kelk).

http://dx.doi.org/10.1016/j.dam.2016.09.023
0166-218X/

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 purposes, 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 colouring 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 centralized 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 graphs,[1] 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 slowdown 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 Neighbour 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].

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 is available or computing a fast heuristic result on the server if it is 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.

[1] The 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.

Please cite this article in press as: R. van Wersch, S. Kelk, ToTo: An open database for computation, storage and retrieval of tree decompositions, Discrete Applied Mathematics (2016), http://dx.doi.org/10.1016/j.dam.2016.09.023