Une étude expérimentale de la largeur d'arbre de données graphe du monde réel

Silviu Maniu
LRI, CNRS, Université Paris-Sud,
Université Paris-Saclay
Orsay, France
silviu.maniu@lri.fr

Pierre Senellart
DI ENS, ENS, CNRS,
PSL Research University
& Inria Paris
& LTCI, Télécom ParisTech
Paris, France
pierre@senellart.com

Suraj Jog
University of Illinois at
Urbana–Champaign
Urbana-Champaign, USA
sjog2@illinois.edu

ABSTRACT

Treewidth is a parameter that measures how tree-like a relational instance is, and whether it can reasonably be decomposed into a tree. Many computation tasks are known to be tractable on databases of small treewidth, but computing the treewidth of a given instance is intractable.

This article is the first large-scale study of treewidth and tree decompositions of real-world graph instances (25 datasets from 8 different domains, with sizes ranging from a few thousand to a few million nodes). The goal is to determine which data, if any, can benefit of the wealth of algorithms for databases of small treewidth. For each dataset, we obtain upper and lower bound estimations of their treewidth, and study the properties of their tree decompositions. Our main finding is that graphs of relatively low treewidth do exist in practice: this is the case of transport and road networks, which make them especially amenable to techniques based on treewidth. We also show that, even when treewidth is high, using partial tree decompositions can still result in data structures that can assist algorithms.

ACM Reference Format:
Silviu Maniu, Pierre Senellart, and Suraj Jog. 2018. Une étude expérimentale de la largeur d'arbre de données graphe du monde réel. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION AND RELATED WORK

A number of data management tasks related to query evaluation are computationally intractable when rich query languages or complex tasks are involved, even when the query is assumed to be fixed (that is, when we consider *data complexity* [55]). For example:

- query evaluation of Boolean monadic second-order (MSO) queries is hard for every level of the polynomial hierarchy [4];
- unless P = NP, there is no polynomial-time enumeration or counting algorithm for first-order (FO) queries with free second-order variables [26, 52];
- computing the probability of conjunctive queries (CQs) over tuple-independent databases, a very simple model of probabilistic databases, is #P-hard [23];
- unless P = NP, there is no polynomial-time algorithm to construct a deterministic decomposable negation normal form (d-DNNF) representation of the Boolean provenance of some CQ [23, 37].

Other problems may yield complexity classes usually considered to be within the realm of tractability, such as Boolean query evaluation which is PTIME for Datalog programs and AC_0 for FO [1], but may still result in impractical running times on large database instances.

To face this intractability and practical inefficiency, one possible approach has been to determine conditions on the structure of databases that ensure tractability, often through a series of *algorithmic meta-theorems* [41]. This has led, for instance, to the introduction of the notions of *locally tree-decomposable structures* for near-linear-time evaluation of Boolean FO queries [30], or to that of *structures of bounded expansion* for constant-delay enumeration of FO queries [38].

Treewidth. A particularly simple and widely used way to restrict database instances that ensures a wide class of tractability results is to bound the *treewidth* of the instance (this is actually a special case of both locally tree-decomposable and bounded expansion). *Treewidth* [51] is a graph-theoretic parameter that characterizes how tree-like a graph, or more generally a relational instance, is, and hence whether it can be reasonably transformed into a tree structure (a *tree decomposition*). Indeed:

- query evaluation of MSO queries is linear-time over bounded-treewidth structures [22, 29];
- counting [11] and enumeration [7, 12] of MSO queries on bounded-treewidth structures is linear-time;
- computing the probability of MSO queries over a bounded-treewidth tuple-independent database is linear-time assuming constant-time rational arithmetic [8];
- a linear-sized d-DNNF representation of the provenance of any MSO query can be computed in linear-time [9].

These results mostly stem from the fact that, on trees, MSO queries can be rewritten to tree automata [53], though this process is non-elementary in general (which impacts the *combined complexity* [55], but not the data complexity). We can see these results as *fixed-parameter tractability*, with a complexity in $O(f(|Q|,k)\times|D|)$ where $|D|$ is the size of the database, k its treewidth, $|Q|$ the size of the query, and f some computable function. Note that another approach for tractability, out of the scope of this paper, is to restrict the queries

Conference'17, July 2017, Washington, DC, USA
2018. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

instead of the instances, e.g., by enforcing low treewidth on the queries [34, 36].

Such results have been, so far, of mostly theoretical interest – mostly due to the high complexity of the function f of $|Q|$ and k. However, algorithms that exploit the low treewidth of instances have been proposed and successfully applied to real-world and synthetic data: for shortest path queries in graphs [49, 57], distance queries in probabilistic graphs [44], or ad-hoc queries compiled to tree automata [46].

Sometimes, treewidth even seems to be the *sole* criterion that may render an intractable problem tractable, under some technical assumptions: [42] shows that MSO_2 query evaluation is intractable over *subinstance-closed* families of instances of treewidth *strongly unbounded poly-logarithmically* unless the exponential-time hypothesis fails; [9, 31] show that MSO query evaluation is intractable over *subinstance-closed* families of instances of treewidth that are *densely unbounded poly-logarithmically* (a weaker notion); [9] shows that counting MSO query results is intractable over *subinstance-closed* families of instances *of unbounded treewidth and treewidth-constructible* (an even weaker notion); finally, [8, 9] shows that one can exhibit FO queries whose probability evaluation is polynomial-time on structures of bounded treewidth, but #P-hard on *any* treewidth-constructible family of instances of unbounded treewidth. For this reason, and because of the wide variety of problems that become tractable on bounded-treewidth instances, treewidth is an especially important object of study.

Treewidth of real-world databases. If there is any hope for practical applicability of treewidth-based approaches, one has to study the following two questions: *Can one efficiently and reliably compute the treewidth of real-world databases?* and *Which real-world data do have low treewidth?* The latter question is the central problem addressed in this paper.

The answer to the former question is that, unfortunately, treewidth cannot reliably be computed efficiently in practice. Indeed, computing the treewidth of a graph is an NP-hard problem [11] and, in practice, exact computation of treewidth is possible only for very small instances, with no more than dozens of vertices [16]. An additional theoretical result of interest is that, given a width w, it is possible to check whether a graph has treewidth w and produce a tree decomposition of the graph in linear time [15]; however, the large constant terms make this procedure impractical.

A more realistic approach is to compute *estimations* of the treewidth, i.e., an interval formed of a *lower bound* and an *upper bound* on the treewidth. Upper bound algorithms (surveyed in [17]) use multiple approaches for estimation, which all output a tree decomposition. One particularly important class of methods for generating tree decompositions relies on *elimination orderings*, that also appear in junction tree algorithms used in belief propagation [43]. For lower bounds (surveyed in [18]), where no decomposition can be obtained, one can use degree-based or minor-based measures on graphs, which themselves act as proxies for treewidth.

Some of these upper bound and lower bound algorithms have been implemented and experimented with in [17, 18, 54]. However, in all cases these algorithms were evaluated on graphs that are either very small (of the order of dozens of vertices), as in [54], or on slightly larger synthetic graphs (with up to 1 000 vertices) generated with exact treewidth values in mind, as in [17, 18]. The main purpose of these experiments was to evaluate the estimators' performance. In the last years, the PACE challenge has had a track dedicated to the estimation of treewidth [24], again on relatively small synthetic graphs.

Another relevant work is [3], which studied the *core-periphery structure* of social networks, by building tree decompositions via node elimination ordering heuristics, but without establishing any treewidth bounds. In this work, we use the same heuristics to compute bounds on treewidth.

Contributions. Our aim in this study is twofold. First, using previously studied algorithms for treewidth estimation, we set out to find classes of real-world data that may exhibit relatively low values of treewidth, thus identifying potential cases in which treewidth-based approaches are of practical interest. For this, after formally defining tree decompositions and treewidth (Section 2), we select the algorithms that are able to deal with large-scale data instances, for both lower- and upper-bound estimations (Section 3). Our aim here is not to exhaustively evaluate treewidth estimation algorithms, but rather to find algorithms that can give acceptable treewidth estimation values in reasonable time. Then, we use these algorithms to obtain lower and upper bound intervals on treewidth for 25 databases from 8 different domains (Section 4). We mostly consider graph data, for which the notion of treewidth was initially designed (the treewidth of an arbitrary relational instance is simply defined as that of its Gaifman graph). The graphs we consider, all obtained from real-world applications, have between several thousands and several millions of vertices. To the best of our knowledge, this is the first comprehensive study of the treewidth of large-scale real-world data of a variety of application domains.

Second, we investigate how a *relaxed (or partial) decomposition* can be used on real-world graphs. In short, we no longer look for complete tree decompositions; instead, we allow the graph to be only partially decomposed. In complex networks, there often exists a dense core together with a tree-like fringe structure [47]; it is hence possible to decompose the fringe into a tree, and to place the rest of the graph in a dense "root". It has been shown that this approach can improve the efficiency of some graph algorithms [5, 44, 57]. In Section 5, we analyze its behavior on real-world graphs. We conclude the paper in Section 6 with a discussion of lessons learned, as to which real-world data admit (full or partial) low-treewidth tree decompositions, and how this impacts query evaluation tasks.

2 PRELIMINARIES ON TREEWIDTH

To make the concepts in the following clear, we start by formally introducing the concept of *treewidth*. Following the original definitions in [51], we first define a tree decomposition:

Definition 1 (Tree Decomposition). *Given an* undirected graph $G = (V, E)$*, where* V *represents the set of vertices (or nodes) and* $E \subseteq V \times V$ *the set of edges, a* tree decomposition *is a pair* (T, B) *where* $T = (I, F)$ *is a tree and* $B : I \to 2^V$ *is a labeling of the nodes of* T *by subsets of* V *(called* bags*), with the following properties:*

(1) $\bigcup_{i \in I} B(i) = V$;
(2) $\forall (u, v) \in E, \exists i \in I$ *s.t.* $\{u, v\} \subseteq B(i)$*; and*

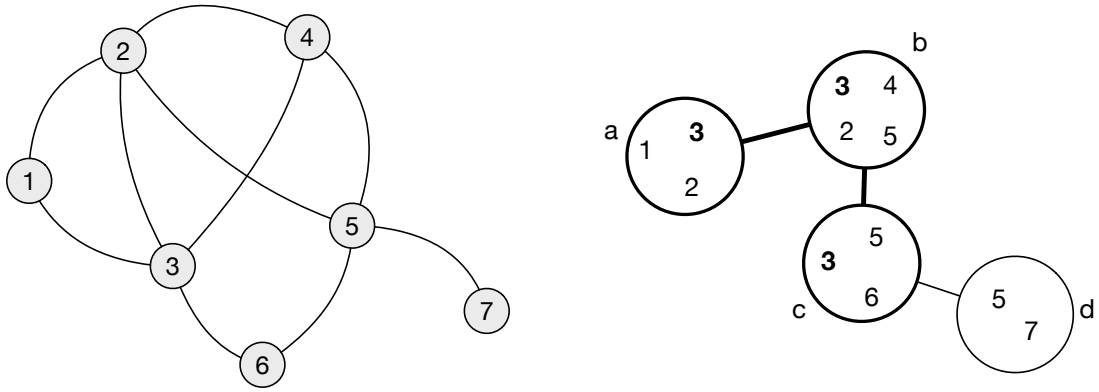


Figure 1: Example undirected, unlabeled, graph (left) and decomposition of width 3 (right)

(3) $\forall v \in V$, $\{i \in I \mid v \in B(i)\}$ *induces a subtree of* T.

Intuitively, a tree decomposition groups the vertices of a graph into bags so that they form a tree-like structure, where a link between bags is established when there exists common vertices in both bags.

Example 2. *Figure 1 illustrates such a decomposition. The resulting decomposition is formed of 4 bags, each containing a subset of the nodes in the graph. The bags containing node* 3 *(in bold) form a connected subtree of the tree decomposition.*

Based on the number of vertices in a bag, we can define the concept of *treewidth*:

Definition 3 (Treewidth). *Given a graph* $G = (V, E)$ *the* width *of a tree decomposition* (T, B) *is equal to* $\max_{i \in I}(|B(i)| - 1)$. *The* treewidth *of* G, $w(G)$, *is equal to the minimal width of all tree decompositions of* G.

It is easy to see that an isolated point has treewidth 0, a tree treewidth 1, a cycle treewidth 2, and a $(k+1)$-clique (a complete graph of k nodes) treewidth k.

Example 4. *The width of the decomposition in Figure 1 is 3. This tells us the graph has a treewidth of at most 3. The treewidth of this graph is actually exactly 3: indeed, the 4-clique, which has treewidth 3, is a* minor *of the graph in Figure 1 (it is obtained by removing nodes* 1 *and* 7, *and by contracting the edges between* 3 *and* 6 *and* 5 *and* 6*), and treewidth never increases when taking a minor (see, for instance, [35]).*

As previously mentioned, the treewidth of an arbitrary relational instance is defined as that of its *Gaifman graph*, the graph whose vertices are constants of the instances and where there is an edge between two vertices if they co-occur in the same fact. We will therefore implicitly represent relational database instances by their Gaifman graphs in what follows.

We are now ready to present algorithms for lower and upper bounds on treewidth.

3 TREEWIDTH ESTIMATION

The objective of our experimental evaluation is to obtain reasonable estimations of treewidth, using algorithms with reasonable execution time on real-world graphs. This eliminates from the start known exact treewidth computation algorithms [16] – they may be usable on small graphs, but they are impossible to apply for our purposes. Indeed, in [16], the largest graph for which the algorithms finished running had at most 40 nodes.

Once we know we do not have the luxury of an exact computation of the treewidth, we are left with estimations of the range of possible treewidths, between a *lower bound* and an *upper bound*. For the purposes of this experimental survey, we restrict ourselves to the most efficient estimation algorithms from the literature. We refer the reader to [17] and [18], respectively, for a more complete survey of treewidth upper and lower bound estimation algorithms on synthetic data.

3.1 Treewidth Upper Bounds

As we have defined, the treewidth is the smallest width among all possible tree decompositions. In other words, the width of any decomposition of a graph is an upper bound of the actual treewidth of that graph. A treewidth upper bound estimation algorithm can thus be seen as an algorithm to find a decomposition whose width is as close as possible to the treewidth of the graph. To understand how one can do that, we need to introduce the classical concept of *elimination orderings* and to explain their connection to treewidth.

We start by introducing *triangulations* of graphs, which transform a graph G into a graph G^{Δ} that is *chordal*:

Definition 5. *A* chordal *graph is a graph* G *such that every cycle in* G *of at least four vertices has a* chord *– an edge between two non-successive vertices in the cycle.*

A triangulation *(or* chordal completion*) of a graph* G *is a minimal chordal supergraph* G^{Δ} *of* G*: a graph obtained from* G *by adding a minimal set of edges to obtain a chordal graph.*

Example 6. *The graph in Figure 1 is not chordal, since, for example, the cycle* 3–4–5–6–3 *does not have a chord. If one adds an edge between*

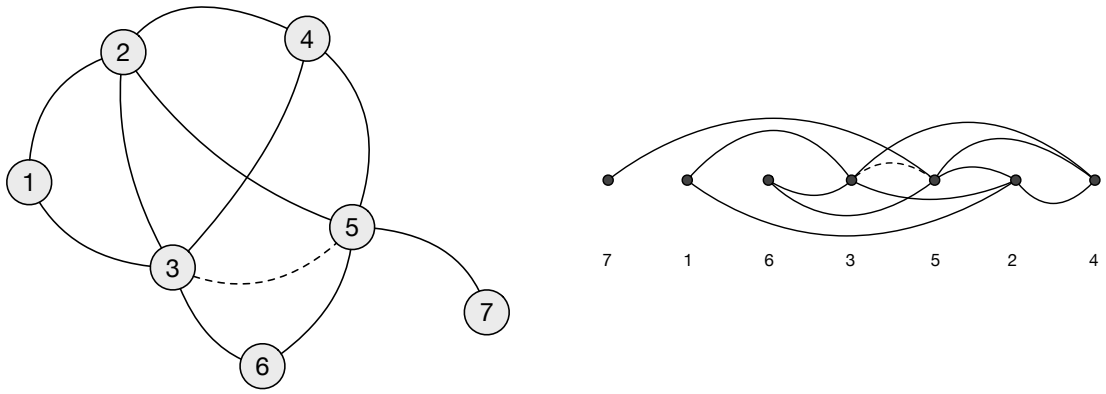


Figure 2: Graph triangulation for the graph of Figure 1 (left) and its elimination ordering (right)

3 *and* 5*, as in Figure 2 (left), one can verify that the resulting graph is chordal, and thus a triangulation of the graph of Figure 1.*

One way to obtain triangulations of graphs is *elimination orderings*. An elimination ordering ω of a graph $G = (V, E)$ of n nodes is an ordering of the vertices of G, i.e., it can be seen as a bijection from V onto $\{1, \ldots, n\}$. From this ordering, one obtains a triangulation by applying sequentially the following *elimination procedure* for each vertex v: first, edges are added between remaining neighbors of v as needed so that they form a clique, then v is *eliminated* (removed) from the graph. For every elimination ordering ω, G along with all edges added to G in the elimination procedure forms a graph, denoted G^{Δ}_{ω}. This graph is chordal (indeed, we know that the two neighbors of the first node of any cycle we encounter in the elimination ordering have been connected by a chord by the elimination procedure). It is also a supergraph of G, and it can be shown it is a minimal chordal supergraph, i.e., a triangulation of G.

Example 7. *Figure 2 (right) shows a possible elimination ordering* $(7, 1, 6, 3, 5, 2, 4)$ *of the graph of Figure 1. The elimination procedure adds a single edge, when processing node* 6*, between nodes* 3 *and* 5*. The resulting triangulation is the graph on the left of Figure 2.*

Elimination orderings are connected to treewidth by the following result:

Theorem 8. [17] *Let $G = (V, E)$ a graph, and $k \leqslant n$. The following are equivalent:*

1. *G has treewidth k.*
2. *G has a triangulation G^{Δ}, such that the maximum clique in G^{Δ} has size $k + 1$.*
3. *There exists an elimination ordering ω such that the maximum clique size in G^{Δ}_{ω} is $k + 1$.*

Obtaining the treewidth of the graph is thus equivalent to finding an optimal elimination ordering. Moreover, constructing a tree decomposition from an elimination ordering is a natural process: each time a vertex is processed, a new bag is created containing the vertex and its neighbors.

Example 9. *In the triangulation of Figure 2 (left), corresponding to the elimination ordering on the right, the maximum clique has size 4: it is induced by the vertices* 2, 3, 4, 5*. This proves the existence of a tree decomposition of width* 3*. Indeed, it is exactly the tree decomposition in Figure 1 (right): bag* d *is constructed when* 7 *is eliminated, bag* a *when* 1 *is eliminated, bag* c *when* 6 *is eliminated, and finally bag* b *when* 3 *is eliminated.*

Finding a "good" upper bound on the treewidth can thus be done by finding a "good" elimination ordering. This is still an intractable problem, of course, but there are various heuristics for generating elimination orderings leading to good treewidth upper bounds. One important class of such elimination ordering heuristics are the greedy heuristics. Intuitively, the elimination ordering is generated in an incremental manner: each time a new node has to be chosen in the elimination procedure, it is chosen using a criterion based on its neighborhood. In our study, we have implemented the following greedy criteria (with ties broken arbitrarily):

- MinDegree. The node with the minimum degree is chosen. [14, 45]
- MinFillIn. The node with the minimum needed "fill-in" (i.e., the minimum number of missing edges for its neighbors to form a clique) is chosen. [17]
- MinDegree+FillIn. The node with the minimum sum of degree and fill-in is chosen.

Example 10. *The elimination ordering of Figure 2 (right) is an example of the use of MinDegree+FillIn. Indeed,* 7 *is first chosen, with value 1, then* 1 *with value 2, then* 6 *with value 2 + 1 = 3. After that, the order is arbitrary since* 2, 3, 4*, and* 5 *form a clique (and thus have initial value 3).*

Previous studies [17, 39, 54] have found these greedy criteria give the closest estimations of the real treewidth. An alternative way of generating an elimination ordering is based on maximum cardinality search [17, 39]; however, it is both less precise tha the greedy algorithms – due to its reliance on how the first node in the ordering is chosen – and slower to run.

3.2 Treewidth Lower Bounds

In contrast to upper bounds, obtaining treewidth lower bounds is not constructive. In other words, lower bounds do not generate

decompositions; instead, the estimation of a lower bound is made by computing other measures on a graph, that are a proxy for treewidth. In this study, we implement algorithms using three approaches: subgraph-based bounds, minor-based bounds, and bounds obtained by constructing improved graphs.

Given a graph $G = (V, E)$, let $\delta(G)$ be its lowest degree, and $\delta_2(G) \geqslant \delta(G)$ its second lowest degree (i.e., the degree of the second vertex when ordered by degree). It is known that $\delta_2(G)$ is itself a lower bound on the treewidth [40]. This, however, is too coarse an estimation, and we need better bounds. We shall use two degeneracy measures of the graphs. The first, the *degeneracy* of G, $\delta D(G)$, is the maximum value of $\delta(H)$ over all subgraphs H of G. Similarly, the *δ_2-degeneracy*, $\delta_2 D(G)$, of a graph is the maximum value of $\delta_2(H)$ over all subgraphs H.

We have the following lemma:

Lemma 11. [18] *Let $G = (V, E)$ be a graph, and $W \subseteq V$ be a set of vertices. The treewidth of the subgraph of G induced by W is at most the treewidth of G.*

A corollary of the above lemma is that the values $\delta D(G)$ and $\delta_2 D(G)$ are themselves lower bounds of treewidth:

Corollary 12. [18] *For every graph G, the treewidth of G is at least $\delta_2 D(G) \geqslant \delta D(G)$.*

To compute $\delta D(G)$ and $\delta_2 D(G)$ exactly, the following natural algorithms can be used [18, 40]: repeatedly remove a vertex of smallest degree – or smallest except for some fixed node v, respectively – from the graph, and keep the maximum value thus encountered. As in [18], we refer to these algorithms as Mmd (Maximum Minimum Degree) and Delta2D, respectively. Ties are broken arbitrarily.

Example 13. *Let us apply the Mmd algorithm to the graph of Figure 1 (left). The algorithm may remove, in order,* 7 *(degree 1),* 1 *(degree 2),* 6 *(degree 2),* 3 *(degree 2),* 5 *(degree 2),* 4 *(degree 1),* 2 *(degree 0). This gives a lower bound of 2 on the treewidth, which is not tight as we saw.*

An equivalent of Lemma 11 on treewidth also holds for *minors* of a graph G: if H is a minor of G, then the treewidth of H is at most the treewidth of G [35]. A minor H of a graph G is a graph obtained by allowing, in addition to edge and node deletion as when taking subgraphs, *edge contractions* – an operation where an edge is removed and its endpoints merged. Then the concepts of *contraction degeneracy*, $\delta C(G)$, and *δ_2-contraction degeneracy*, $\delta_2 C(G)$, are defined analogously to $\delta D(G)$ and $\delta_2 D(G)$ by considering all minors instead of all subgraphs:

Lemma 14. [18] *For every graph G, the treewidth of G is at least $\delta_2 C(G) \geqslant \delta C(G)$.*

Unfortunately, computing $\delta C(G)$ or $\delta_2 C(G)$ is NP-hard; hence, only heuristics can be used. One such heuristic for $\delta C(G)$ is a simple change to the Mmd algorithm, called Mmd+ [19, 33]: at each step, instead of removing a vertex, a neighbor is chosen and the corresponding edge is contracted. Choosing a neighbor node to contract requires some heuristic also; in line with previous studies, in this study we choose the neighbor node that has the least overlap in neighbors – this is called the *least-c* heuristic [58].

Finally, another approach to treewidth lower bounds that we consider are *improved graphs*, an approach that can be used in combination with any of the lower bound estimation algorithms presented so far. Consider a graph G and an integer k and the following operation: while there are non-adjacent vertices v and w that have at least $k+1$ common neighbors, add the edge (v, w) to the improved graph G'. The resulting graph G' is the $(k+1)$-neighbor improved graph of G. Using these improved graphs can lead to a lower bound on treewidth: the $(k+1)$-neighbor improved graph G' of a graph G having at most treewidth k also has treewidth at most k.

To use this property, one can start from an already computed estimation k of a lower bound (by using Mmd, Mmd+, or Delta2D for example) and then repeatedly generate a $(k+1)$-neighbor improved graph, estimate a new lower bound on treewidth, and repeat the process until the graph cannot be improved. This algorithm is known as Lbn in the literature [21], and can be combined with any other lower bound estimation algorithm. A refinement of Lbn, that alternates improvement and contraction steps, Lbn+, has also been proposed [19].

Example 15. *Let us illustrate the use of Lbn together with Mmd on the graph of Figure 1 (left). As shown in Example 13, a first run of Mmd yields $k = 2$. We compute a 3-neighbor improved graph for G by adding an edge between nodes* 3 *and* 5 *(that share neighbors* 2, 4, 6*).*

Now, running Mmd one more time yields the possible sequence 7 *(degree 1),* 1 *(degree 2),* 6 *(degree 2),* 3 *(degree 3),* 5 *(degree 2),* 4 *(degree 1),* 2 *(degree 0). We thus obtain a lower bound of* 3 *on the treewidth, which is this time tight.*

3.3 Computational Analysis

An important aspect of estimations of treewidth are how much computational power they use, as a function of the size of the original graph.

In the case of *upper bounds*, the cost depends quadratically on the upper bound w found by the decomposition. This is due to the fact that, at each step, when a node is chosen, a fill-in of $O(w^2)$ edges is computed between the neighbors. Depending on the greedy algorithm used and the criteria for generating the ordering, the complexity of updating the queue for extracting the new node is $O(w \log w)$ for Degree, and $O(w^2 \log w)$ for the others (the neighbors of neighbor have to be taken into account as possible updates). Hence, in the worst case, the complexities of the greedy heuristics are the following: for MinDegree $O(|V|^2 \log |V|)$ and for MinFillIn, MinDegreeFillIn the cost increases to $O(|V|^3 \log |V|)$.

For *lower bounds*, the cost greatly depends on the algorithm chosen. In the case of Mmd and Mmd+, the costliest operation is to sort and update a queue of degrees, and the whole computation can be done in $O(|E| + |V| \log |V|)$. For Delta2D the cost is known to be $O(|V| \times |E|)$ [40]. Lbn and Lbn+ add a $O(|V|)$ factor.

Surprisingly, as we shall see in the experiments, the best estimations do not necessarily come from the costliest algorithms. Indeed, in our experiments, the simplest algorithms tend to also give the best estimations.

4 ESTIMATION RESULTS

We now present our main experimental study, first introducing the 25 datasets we are considering, then upper and lower bound results, running time and estimators, and an aside discussion of the treewidth of synthetic networks. All experiments were made

on a server using an 8-core Intel Xeon 1.70GHz CPU, having 32GB of RAM, and using 64bit Debian Linux. All datasets were given at least two week to finish, after which the algorithms were stopped and the best lower and upper bounds were recorded.

4.1 Datasets

For our study, we have evaluated the treewidth estimation algorithms on 25 datasets from 8 different domains. We detail them below:

Infrastructure. For this domain, we have collected four types of datasets. The first are road networks of three US states, CA, PA, and TX, downloaded from the site of the 9th DIMACS Challenge on shortest paths[1]. Second, we have extracted, from the XML export of OpenStreetMap[2], the city maps of BUCHAREST, HONGKONG, PARIS, and LONDON. The OpenStreetMap format consists of *ways* which represent routes between the nodes in the graph. We have extracted all the ways appearing in each map, and eliminated the nodes which appeared in more than one way. For evaluating public transport graphs, we used the STIF dataset from the open public transit data of the Paris metropolitan area[3], where the nodes are the stations of the system (bus, train, metro) and an edge appears if at least one transport line exists between the two stations. Finally, we used the Western US power grid network, USPOWERGRID, first used in [56], in which the edges are the major power lines and the nodes represent electrical equipment (transformers, generators, etc.).

Social Networks. The networks used in this domain are mainly taken from the Stanford SNAP repository[4]; this is the case for the FACEBOOK (ego networks), ENRON (Enron email conversations), WIKITALK (convesations between Wikipedia contributors), CITHEPH (citations between authors in high-energy physics), and LIVEJOURNAL (social network of the Livejournal site). Other social datasets were extracted from the StackOverflow Q&A site[5], for the mathematics sub-domain (STACK-MATH) and the theoretical computer science sub-domain (STACK-TCS). The nodes represent users of the site, and an edge exists when a reply, vote, or comment occurs between the edge endpoints.

Web Networks. For Web-like graphs, we evaluated treewidth on the WIKIPEDIA network of articles, and the GOOGLE Web graph (provided by Google during its 2002 programming contest); the versions we used were downloaded from the Stanford SNAP website.

Hierarchical. The next category is of data that has by nature a hierarchical structure. ROYAL is extracted from a genealogy of royal families in Europe, originally published in GEDCOM format by Brian Tompsett and that has informally circulated on the Web since 1992[6]. MATH is a graph of academic advisor–advisee data in mathematics and related areas, crawled from http://www.genealogy.ams.org/.

Ontologies. We used subsets of two of the most popular knowledge bases available on the Web: YAGO[7] and DBPEDIA[8]. For YAGO, we downloaded its core facts subset, and removed the semantics on the links; hence, an edge exists if there exists at least a fact between two concepts (nodes). For DBPEDIA, we downloaded the ontology subset containing its RDF type statements, and we generated the edges using the same approach as for YAGO.

Others. In addition to the above, we have evaluated treewidth on other types of networks: a communication network (GNUTELLA) [50], a protein–protein interaction network (YEAST) [20], and the Gaifman graph of the TPCH relational database benchmark (TPCH)[9], generated using the official tools given and using the default parameters.

Table 1 summarizes the datasets, their size, and the best treewidth estimations we have been able to compute. For reproducibility purposes, all datasets, along with the code that has been used to compute the treewidth estimations, can be freely downloaded from https://github.com/smaniu/treewidth/.

4.2 Upper Bounds

We show in Figure 3 the results of our estimation algorithms. Lower values mean better treewidth estimations. Focusing on the upper bounds only (red circular points), we notice that, in general, FILLIN does give the smallest upper bound of treewidth, in line with previous findings [18]. Interestingly, the DEGREE heuristic is quite competitive with the other heuristics. This fact, coupled with its lower running time, means that it can be used more reliably in large graphs. Indeed, as can be seen in the figure, on some large graphs only the DEGREE heuristic actually finished at all; this means that, as a general rule, DEGREE seems the best fit for a quick and relatively reliable estimation of treewidth.

We plot both the absolute values of the estimations in Figure 3a, but also their relative values (in Figure 3b, representing the ratio of the estimation over the number of nodes in the graph), to allow for an easier comparison between networks. The absolute value, while interesting, does not yield an intuition on how the bounds can differ between network types. If we look at the relative values of treewidth, it becomes clear that infrastructure networks have a treewidth that is much lower than other networks; in general they seem to be consistently under one thousandth of the original size of the graph. This suggests that, indeed, this type of network may have properties that make them have a lower treewidth. For the other types of networks, the estimations can vary considerably: they can go from one hundredth (e.g., MATH) to one tenth (e.g., WIKITALK) of the size of the graph.

The bounds obtained here on infrastructure networks are consistent with a conjectured $O(\sqrt[3]{n})$ bound on the treewidth of road networks [25]. One relevant property is their *low highway dimension* [2], which helps with routing queries and decomposition into contraction hierarchies. Even more relevant to our results is the fact that they tend to be "almost planar". More specifically, they are

[1] http://www.dis.uniroma1.it/challenge9/
[2] https://www.openstreetmap.org/
[3] https://opendata.stif.info/page/home/
[4] https://snap.stanford.edu/
[5] http://stackoverflow.com/
[6] https://www.hull.ac.uk/php/cssbct/genealogy/royal/nogedcom.html
[7] https://www.yago-knowledge.org/
[8] https://www.dbpedia.org
[9] http://www.tpc.org/tpch/

Table 1: Datasets and summary of lower and upper bounds. Bounds with a * are partial.

type	Dataset name	nodes	edges	Lower width	Upper width
infrastructure	Ca	1 965 206	2 766 607	5	252
	Pa	1 088 092	1 541 898	5	333
	Tx	1 379 917	1 921 660	5	189
	Bucharest	189 732	223 143	21	139
	HongKong	321 210	409 038	32	145
	Paris	4 325 486	5 395 531	55	521
	London	2 099 114	2 588 544	57	507
	Stif	17 720	31 799	28	86
	USPowerGrid	4 941	6 594	10	18
social	Facebook	4 039	88 234	142	247
	Enron	36 692	183 831	257	1 989
	WikiTalk	2 394 385	4 659 565	1 113	12 843
	CitHeph	34 546	420 877	469	9 498
	Stack-TCS	25 232	69 026	143	717
	Stack-Math	1 132 468	2 853 815	850	11 100
	LiveJournal	3 997 962	34 681 189	360	919 532*
web	Wikipedia	252 335	2 427 434	1 007	19 876
	Google	875 713	4 322 051	621	17 571
communication	Gnutella	65 586	147 892	244	9 374
hierarchy	Royal	3 007	4 862	11	24
	Math	101 898	105 131	56	515
ontology	Yago	2 635 315	5 216 293	836	79 059*
	DbPedia	7 697 211	30 622 392	28	538 805*
database	Tpch	1 381 291	79 352 127	699	124 316*
biology	Yeast	2 284	6 646	54	255

k-planar: each edge can allow up to k crossing in their plane embedding. It has been shown in [27] that k-planar graphs have treewidth $O(\sqrt{(k+1)n})$, a relatively low treewidth that is consistent with our results.

4.3 Lower Bounds

Figure 3 also reports on the lower bound estimations (blue rectangular points). Now, higher values represent a better estimation. The same differentiation between infrastructure networks and the other networks holds in the case of lower bounds – treewidth lower bounds are much lower in comparison with other networks. We observe that the variation between upper and lower bound estimations can be quite large. Generally, we find that degree-based estimators, Mmd and Delta2D, give bounds that are very weak. The contraction-based estimator, Mmd+, however, consistently seems to give the best bounds of treewidth, and returns lower bounds that are much larger than the degree-based estimations.

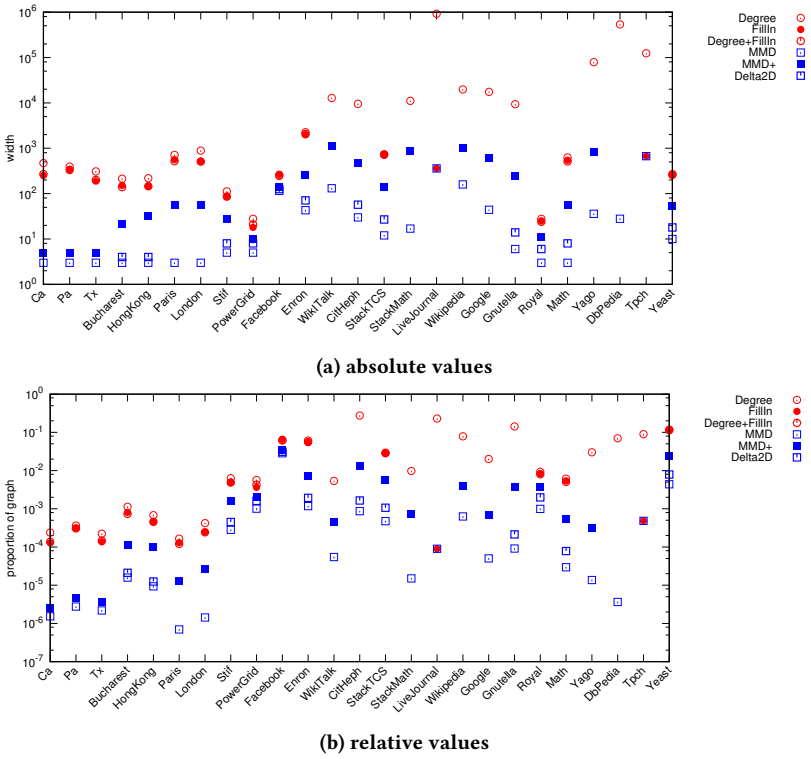
Interestingly, in the case of the networks Ca, Pa, and Tx, the values returned for Mmd+ and Mmd are always 5 and 3, respectively. This has been remarked on in [19] – there exist instances where heuristics of lower bounds perform poorly, giving very low lower bounds. In our case, this only occurs for some road networks, all coming from the same data source, i.e., the DIMACS challenge on shortest paths.

In Figure 3, we have not plotted the estimations resulting from Lbn and Lbn+. The reason is that we have found that these algorithms do not generally give any improvement in the estimation of the lower bounds. Specifically, we have found an improvement only in two datasets for the Delta2D heuristic: for Facebook from 126 originally to 157 for Lbn(Delta2D) and 159 for Lbn+(Delta2D); and for Stack-TCS from 27 originally to 30 for Lbn+(Delta2D). In all cases, however, Mmd+ is always competitive by itself with these estimations.

4.4 Running Time

As explained in Section 3.3, the complexity of different estimation algorithms varies a lot, ranging from quasi-linear for low-treewidth to cubic time. Even if all of the algorithms exhibit polynomial complexity, the cost can become quickly prohibitive, even for graphs of relatively small sizes. Figure 4 presents the running time for the upper and lower bound algorithms, on a logarithmic scale.

We need to first note that not all algorithms finished on all datasets within the time bound of roughly one *week* of computation time: indeed, only the fastest algorithms for each bound – Degree and Mmd respectively – finish processing all datasets. In some cases, for upper bounds, even Degree timed out – in this case, we took the value found at the moment of the time-out; this still represents an upper bound. The datasets for which this occurred are LiveJournal, Yago, DbPedia, and Tpch.



(a) absolute values

(b) relative values

Figure 3: Treewidth estimation of different algorithms (logarithmic scale)

For the upper bounds (Figure 4a), there is a clear distinction between Degree on the one hand and FillIn or Degree+FillIn on the other: Degree is always at least one order of magnitude faster than the others. Moreover, for larger networks, only Degree has a chance to finish. This also occurs for lower bounds (Figure 4b): Mmd and Mmd+ have reasonable running times, while Delta2D has a running time that is much larger – by several orders of magnitude. On the other hand, the Lbn and Lbn+ methods (Figure 4c) do not add much to the running time of Mmd and Mmd+; we conjecture that this occurs due to the fact that not many iterations are performed, which also explains the fact that they generally do not give improvements in the estimated bounds.

4.5 Phase Transitions in Synthetic Networks

We have seen that graphs other than the infrastructure networks have treewidth values that are relatively high. An interesting question that these results lead to is: *is it the case for all relatively complex networks, or is it a particularity of the chosen datasets?*

To give a possible answer to this, we have evaluated the treewidth of a range of synthetic graph models and their parameters:

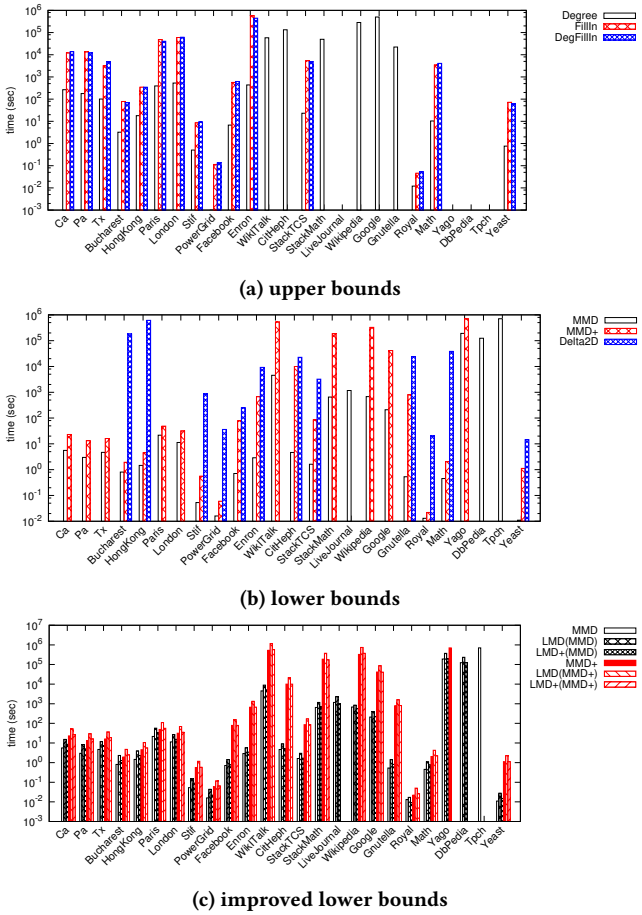
Random. This synthetic network model due to Erdős and Rényi [28] is an entirely random one: given n nodes and a parameter $p \in (0, 1]$, each possible edge between the n nodes is added with probability p. We have generated several network of $n = 10\,000$ nodes, and with values of p ranging from 10^{-5} to 10^{-3}.

Preferential Attachment. This network model [6] is a generative model, and aims to simulate link formation in social networks: each time a new node is added to the network, it attaches itself to m other nodes, each link being formed with a probability proportional to the number of neighbors the existing node already has. We have generated graphs of $n = 10\,000$ nodes, with the parameter m varying between 1 and 20.

Small-World. The model [56] generates a small-world graph by the following process: first, the n nodes are organized in a ring graph where each node is connected with m other nodes on each side; finally, each edge is rewired (its endpoints are changed) with probability p. In the experiments, we have generated graphs of $n = 10\,000$, with $p = \{0.1, 0.2, 0.5\}$ and m ranging between 1 and 20.

Our objective with evaluating treewidth on these synthetic instances is to evaluate whether some parameters of these models lead to lower treewidth values, and if so, in which model the "phase transition" occurs, i.e., when a low-treewidth regime gives way to a high-treewidth one. For our purpose, and for reasons we explain in Section 5, we consider a treewidth under $\sqrt{n}$ to be low.

Our first finding – see Figure 5 – is that the high-treewidth regime arrives very early, relative to the parameter values. For random graphs, only relatively small value of p allow for a low treewidth; for small-world and preferential attachment only trivial values of m allow low-treewidth – and, even so, it is usually 1 or 2, possibly due to the fact that the graph, in those cases, is composed



(a) upper bounds

(b) lower bounds

(c) improved lower bounds

Figure 4: Running time of different algorithms (logarithmic scale)

of several small connected components, i.e., the well-known subcritical regime of random graphs [13]. The low-treewidth regime for random networks seems to be limited to values immediately after $p = \frac{1}{n}$; after this point, the treewidth is high, in line with findings of a linear dependency between graph size and treewidth in random graphs [32]. Moreover, we notice that there is no smooth transition for preferential attachment and small world networks; the treewidth jumps from very low values to high treewidth values. This is understandable in scale-free networks – resulting from the preferential attachment model – where a few hubs may exist with degrees that are comparable to the number of nodes in the graph. Comparatively, random networks exhibit a smoother progression – one can clearly see the shift from trivial values of treewidth, to relatively low values, and to high treewidth values. Finally, the gap between lower bound and upper bound tends to increase with the parameter; that is, however, not surprising since all three model graphs tend to have more edges with larger parameter values.

5 PARTIAL DECOMPOSITIONS

Our results show that, in practice, the treewidths of real networks are quite high. Even in the case of road networks, having relatively low treewidths, their value can go in the hundreds, rendering most algorithms whose time is exponential time in the treewidth (or worse) unusable.

In practical applications, however, we can still adapt treewidth-based approaches for obtaining data structures – not unlike indexes – which can help with some important graph queries like shortest distances and paths [5, 57] or probability estimations [10, 44].

The manner in which treewidth decomposition can be used starts from a simple observation made in studies on complex graphs, that is, that they tend to exhibit a *tree-like fringe* and a *densely connected core* [47, 48]. The tree-like fringe precisely corresponds to bounded-treewidth parts of the network. This yields an easy adaptation of the upper bound algorithms based on node ordering: given a parameter w representing the highest treewidth the fringe can be, we can run any greedy decomposition algorithm (MinDegree, MinFillIn, MinDegreeFillIn) until we only find nodes of degree $w + 1$, at which point the algorithm stops. At termination, we obtain a data structure resembling that of Figure 6: a set of treewidth w elements (w-trees) interfacing with a *core* (of unknown treewidth), through cliques that have size at most $w + 1$.

The resulting structure can be thought of as a *partial decomposition* (or *relaxed decomposition*), a concept introduced in [5, 57] in the context of answering shortest path queries, and used in [44] for probabilistic distance queries. A partial decomposition can be extremely useful. The tree-like fringe can be used to quickly precompute answers to partial queries (e.g., precompute distances in the graph). Once the precomputation is done, these (partial) answers are added to the core graph, where queries can be answered directly.

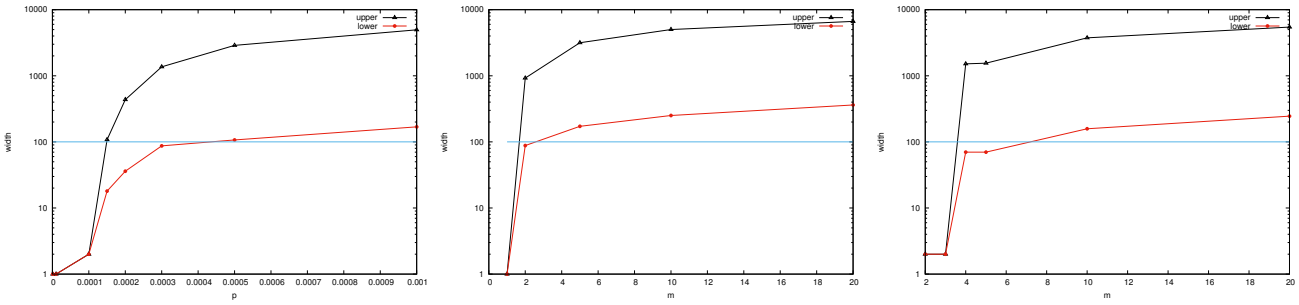
If the resulting core graph is much smaller than the original graph, the gains in running time can be considerable, as shown in [5, 44, 57]. Hence, the objective of our experiments in this section is to check how feasible partial decompositions are.

5.1 Evaluation

An interesting aspect of greedy upper bound algorithms is that they generate at any point a partial decomposition, with width equal to the highest degree encountered in the greedy ordering so far. The algorithm can then be stopped at any width, and the size of the resulting partial decomposition can be measured.

To evaluate this, we track here the size of the core graph, in terms of edges. We do this because most algorithms on graphs have a complexity that is directly related to the number of edges in the graph. As we discussed in the previous section, we aim that the size of the core graph to be as small as possible. Another aspect to keep in mind is the number of edges that are added by the fill-in process during the decomposition: each time a node of degree w is removed, $O(w^2)$ edges can be added to the graph. Finally, for a graph of treewidth k, the decomposition contains a root of size $O(k^2)$. To ensure that the core graph is smaller than the original graph we aim for the treewidth to be $O(\sqrt{n})$. As we saw in Section 4, this is only rarely the case, and it is more likely to occur in road networks.

Indeed, if we plot the core graph size in the partial decompositions of infrastructure networks (Figures 7a, 7b), we see that their size is only a fraction of the original size. We see a large drop in



(a) random (b) preferential attachment (c) small world

Figure 5: Lower and upper bounds of treewidth in synthetic networks, as a function of the generator parameters. The blue line represent the "low" width regime, where treewidth is less than 100.

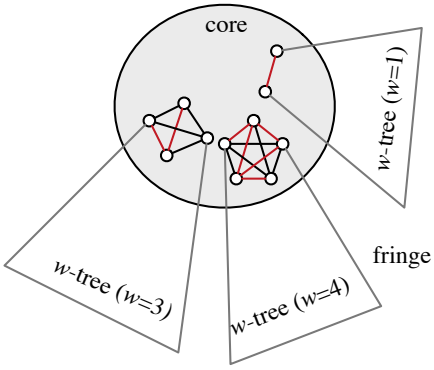


Figure 6: An abstract view of partial decompositions. Partial decompositions are formed of a *core* graph, which interfaces with w-trees through w-cliques (the *fringe*).

the size for low widths; this is logical, since the fill-in process does not add edges for $w = 1$ and $w = 2$. After this point, the size is relatively stable, and can go as low as 10% of the original size. In this sense, infrastructure networks seem the best fit for indexes based on decompositions, and represents further confirmation of the good behavior of these networks for hierarchical decompositions.

This desired decomposition behavior no longer occurs for social networks (Figures 7c, 7d) and the other networks in our dataset (Figures 7e, 7h). In this case, the decomposition becomes too large and even unwieldy – for some networks, the decomposition started filling the main memory of our computers (32GB of RAM); for other networks, they did not finish in reasonable time (i.e., a few days of computation). For most of these networks, the resulting treewidth is much larger than our desired bound of $O(\sqrt{n})$; this results in core graph sizes that can be hundreds of times larger than the original size. The only exceptions are hierarchical networks, Royal and Math(Figures 7g, 7h), which are after all supposed to be close to a tree – despite having a relatively high treewidth (remember Figure 3b), partial decompositions work well on them.

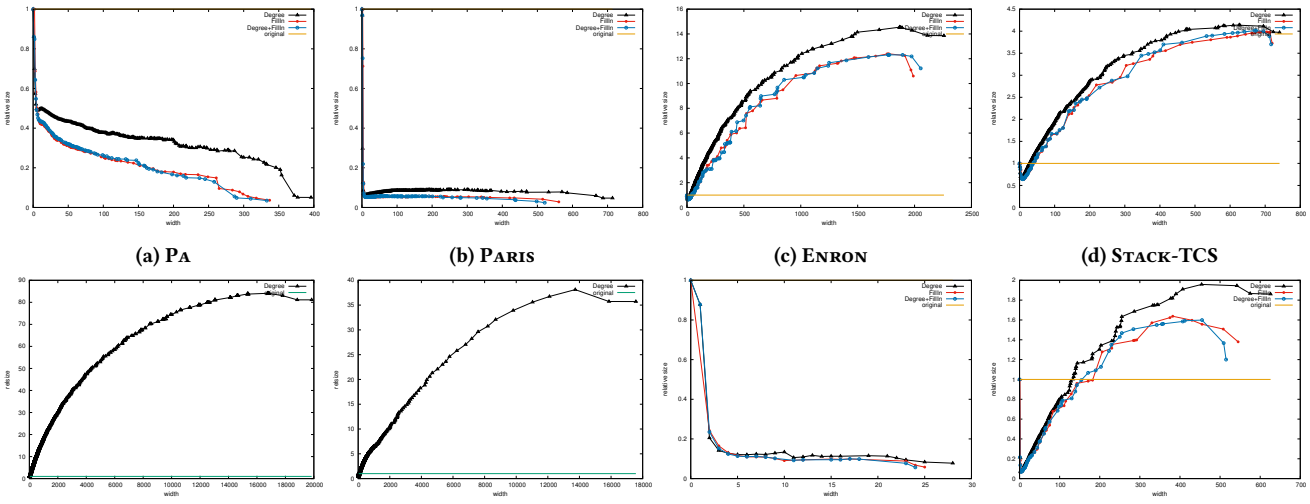
In practice, however, we do not need to compute full decompositions. For usable indexes, we may want to stop at low treewidths, which allow exact computing. This may be useful for graphs whose decompositions have large core graphs. To check this, we plot in Figure 8 decompositions up to a width of 25, for a selection of graphs, including the graphs for which no upper bound algorithm has finished (Yago, DbPedia, LiveJournal, Tpch). Immediately apparent is the fact that the minimal size of the core graph occurs at very low widths: in almost all cases, it occurs around a width between 5 and 10. This is low enough that running algorithms even doubly exponential in the width on the resulting fringe graphs can be performed. In terms of actual size, it can vary greatly. In road networks, it even reaches 10%, compared to around 50% for other graphs. The exception to this are denser networks (CitHeph and LiveJournal) where almost no benefit is visible for partial decompositions of small width. An interesting behavior occurs in Tpch, where the decomposition size changes in steps; we conjecture this is due to the fact that database instances contain many cliques – specifically, one clique for each tuple in each relation.

6 DISCUSSION

In this article, we have estimated the treewidth of a variety of graph datasets from different domains, and we have studied the running time and effectiveness of estimation algorithms. This study was motivated by the central role treewidth plays in many algorithms attacking query evaluation tasks that are otherwise intractable, where low treewidths are an indicator for low complexity.

In terms of *treewidth estimations*, we have discovered that, generally, the treewidth of real-world graphs is quite large. With few exceptions, most of the graphs we have tested in this study are *scale-free*. This may partly explain our findings – scale-free networks exhibit a number of high-degree, or *hub*, nodes that force high values for the treewidth. The few exceptions to this rule are *infrastructure networks*, where treewidths can be quite low. Indeed, we were able to reproduce a $O(\sqrt[3]{n})$ bound on the treewidth of road networks. We conjecture these low bounds are explained by characteristics of infrastructure networks: specifically, they are similar to very sparse random networks.

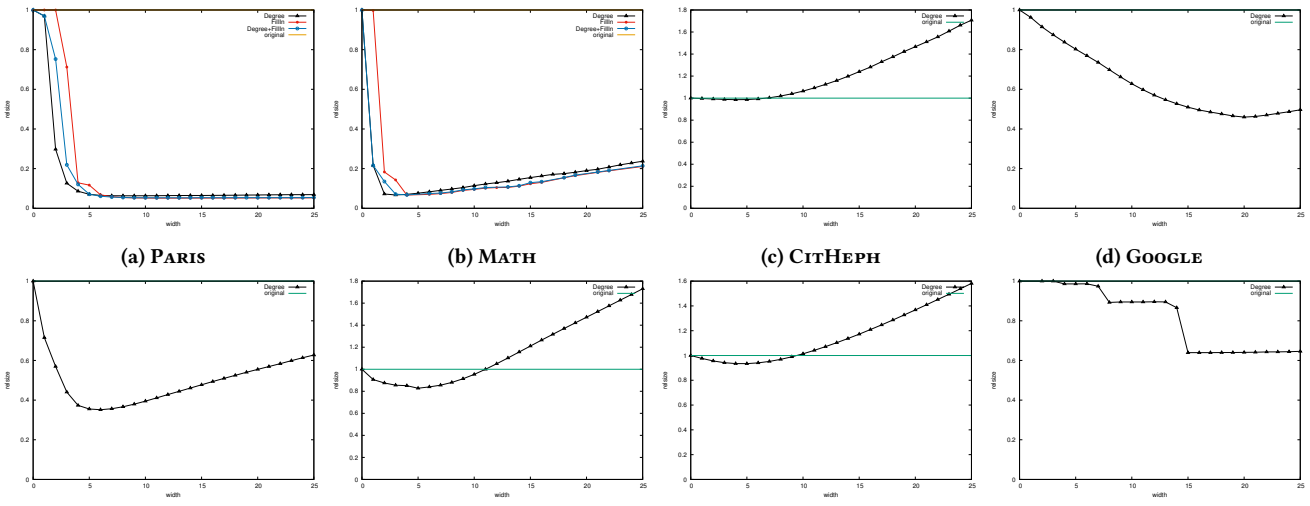
Our study on the *algorithms* for estimation leads to results that may seem surprising. For upper bounds, we discovered that greedy treewidth estimators, based on elimination orderings, provide the best cost–estimation trade-off; they also have the advantage of



(a) Pa (b) Paris (c) Enron (d) Stack-TCS

(e) Wikipedia (f) Google (g) Royal (h) Math

Figure 7: Relative sizes of core graphs in partial decompositions, after all bags of a given size have been removed in the decomposition.



(a) Paris (b) Math (c) CitHeph (d) Google

(e) Yago (f) DbPedia (g) LiveJournal (h) Tpch

Figure 8: Zoomed view of decompositions up to width 25

outputting readily-usable tree decompositions. In the case of lower bounds, we discovered that degeneracy-based bounds display the best behavior; moreover, the algorithms which aim to improve the bounds (LBN and LBN+) only very rarely do so.

Tree decompositions are useful for a variety of applications, ranging from query evaluation, to enumeration, probability estimation, knowledge compilation, etc. We give further evidence in this study that, when treewidth is small, the decomposition themselves are relatively "small" also. Here, by small, we mean a tree decomposition where the largest bag (usually the core graph) has a size that is only a fraction of the overall graph size. Such small decompositions can greatly impact the execution time of graph algorithms when they are used, and in general when pre-computations are made on paths in the graph. In cases where treewidth is high, and hence the decompositions are "large", we show in this study that decompositions can still be applied. Specifically, when a decomposition based on elimination orderings is stopped once bags of a certain size have been found (generally, between 5 and 10), the resulting decompositions can still be smaller than the original graphs, and thus still have a good potential to accelerate graph operations when they are used as indexes.

ACKNOWLEDGMENTS

We are grateful to Antoine Amarilli and Mikaël Monet for feedback on parts of this paper.

REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of databases*. Addison-Wesley, 1995.
[2] I. Abraham, A. Fiat, A. V. Goldberg, and R. F. Werneck. Highway dimension, shortest paths, and provably efficient algorithms. In *SODA*, 2010.
[3] A. B. Adcock, B. D. Sullivan, and M. W. Mahoney. Tree decompositions and social graphs. *Internet Mathematics*, 12(5), 2016.
[4] M. Ajtai, R. Fagin, and L. J. Stockmeyer. The closure of monadic NP. *JCSS*, 60(3), 2000.
[5] T. Akiba, C. Sommer, and K.-i. Kawarabayashi. Shortest-path queries for complex networks: Exploiting low tree-width outside the core. In *EDBT*, 2012.
[6] R. Albert and A.-L. Barabási. Statistical mechanics of complex networks. *Rev. Mod. Phys.*, 74, 2002.
[7] A. Amarilli, P. Bourhis, L. Jachiet, and S. Mengel. A circuit-based approach to efficient enumeration. In *ICALP*, 2017.
[8] A. Amarilli, P. Bourhis, and P. Senellart. Provenance circuits for trees and treelike instances. In *ICALP*, 2015.
[9] A. Amarilli, P. Bourhis, and P. Senellart. Tractable lineages on treelike instances: Limits and extensions. In *PODS*, 2016.
[10] A. Amarilli, S. Maniu, and M. Monet. Challenges for efficient query evaluation on structured probabilistic data. In *SUM*, 2016.
[11] S. Arnborg, D. G. Corneil, and A. Proskuworski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic and Discrete Methods*, 8(2), 1987.
[12] G. Bagan. MSO queries on tree decomposable structures are computable with linear delay. In *CSL*, volume 4207, 2006.
[13] A.-L. Barabási and M. Pósfai. *Network science*. Cambridge University Press, 2016.
[14] A. Berry, P. Heggernes, and G. Simonet. The minimum degree heuristic and the minimal triangulation process. *Graph-Theoretic Concepts in Computer Science*, 2880(Chapter 6), 2003.
[15] H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6), 1996.
[16] H. L. Bodlaender, F. V. Fomin, A. M. C. A. Koster, D. Kratsch, and D. M. Thilikos. On exact algorithms for treewidth. *ACM TALG*, 9(1), 2012.
[17] H. L. Bodlaender and A. M. C. A. Koster. Treewidth computations I. Upper bounds. *Information and Computation*, 208(3), 2010.
[18] H. L. Bodlaender and A. M. C. A. Koster. Treewidth computations II. Lower bounds. *Information and Computation*, 209(7), 2011.
[19] H. L. Bodlaender, A. M. C. A. Koster, and T. Wolle. Contraction and treewidth lower bounds. In *ESA*, 2004.
[20] D. Bu, Y. Zhao, L. Cai, H. Xue, X. Zhu, H. Lu, J. Zhang, S. Sun, L. Ling, N. Zhang, G. Li, and R. Chen. Topological structure analysis of the protein–protein interaction network in budding yeast. *Nucleic Acids Research*, 31(9), 2003.
[21] F. Clautiaux, J. Carlier, A. Moukrim, and S. Nègre. New lower and upper bounds for graph treewidth. In *WEA*, 2003.
[22] B. Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Inf. Comput.*, 85(1), 1990.
[23] N. N. Dalvi and D. Suciu. The dichotomy of conjunctive queries on probabilistic structures. In *PODS*, 2007.
[24] H. Dell, C. Komusiewicz, N. Talmon, and M. Weller. The PACE 2017 parametrized algorithms and computation experiments challenge: The second iteration. In *IPEC*, 2017.
[25] J. Dibbelt, B. Strasser, and D. Wagner. Customizable contraction hierarchies. *Journal of Experimental Algorithmics*, 21(1), 2016.
[26] A. Durand and Y. Strozecki. Enumeration complexity of logical query problems with second-order variables. In *CSL*, 2011.
[27] D. Eppstein and D. R. Wood. Structure of graphs with locally restricted crossings. *arXiv.org*, 2015.
[28] P. Erdős and A. Rényi. On random graphs I. *Publicationes Mathematicae (Debrecen)*, 6, 1959.
[29] J. Flum, M. Frick, and M. Grohe. Query evaluation via tree-decompositions. *J. ACM*, 49(6), 2002.
[30] M. Frick and M. Grohe. Deciding first-order properties of locally tree-decomposable structures. *J. ACM*, 48(6), 2001.
[31] R. Ganian, P. Hliněný, A. Langer, J. Obdržálek, P. Rossmanith, and S. Sikdar. Lower bounds on the complexity of MSO1 model-checking. *JCSS*, 1(80), 2014.
[32] Y. Gao. Treewidth of Erdős–Rényi random graphs, random intersection graphs, and scale-free random graphs. *Discrete Applied Mathematics*, 160(4–5), 2012.
[33] V. Gogate and R. Dechter. A complete anytime algorithm for treewidth. In *UAI*, 2004.
[34] M. Grohe, T. Schwentick, and L. Segoufin. When is the evaluation of conjunctive queries tractable? In *STOC*, 2001.
[35] D. J. Harvey. *On Treewidth and Graph Minors*. PhD thesis, The University of Melbourne, 2014.
[36] A. Jha and D. Suciu. On the tractability of query compilation and bounded treewidth. In *ICDT*, 2012.
[37] A. K. Jha and D. Suciu. Knowledge compilation meets database theory: Compiling queries to decision diagrams. *Theory Comput. Syst.*, 52(3), 2013.
[38] W. Kazana and L. Segoufin. Enumeration of first-order queries on classes of structures with bounded expansion. In *PODS*, 2013.
[39] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press, 2009.
[40] A. M. C. A. Koster, T. Wolle, and H. L. Bodlaender. Degree-based treewidth lower bounds. In *WEA*, 2005.
[41] S. Kreutzer. Algorithmic meta-theorems. *CoRR*, abs/0902.3616, 2009.
[42] S. Kreutzer and S. Tazari. Lower bounds for the complexity of monadic second-order logic. In *LICS*, 2010.
[43] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society Series B (Methodological)*, 50(2), 1988.
[44] S. Maniu, R. Cheng, and P. Senellart. An indexing framework for queries on probabilistic graphs. *ACM Trans. Database Syst.*, 42(2), 2017.
[45] H. M. Markowitz. The elimination form of the inverse and its application to linear programming. *Management Science*, 3(3), 1957.
[46] M. Monet. Probabilistic evaluation of expressive queries on bounded-treewidth instances. In *SIGMOD PhD Symposium*, 2016.
[47] M. E. J. Newman, S. H. Strogatz, and D. J. Watts. Random graphs with arbitrary degree distributions and their applications. *Phys. Rev. E*, 64, Jul 2001.
[48] M. E. J. Newman, D. J. Watts, and S. H. Strogatz. Random graph models of social networks. In *Proceedings of the National Academy of Sciences*, 2002.
[49] L. Planken, M. de Weerdt, and R. van der Krogt. Computing all-pairs shortest paths by leveraging low treewidth. *Journal of Artificial Intelligence Research*, 43, 2012.
[50] M. Ripeanu, A. Iamnitchi, and I. Foster. Mapping the Gnutella network. *IEEE Internet Computing*, 6(1), 2002.
[51] N. Robertson and P. D. Seymour. Graph minors. III. Planar tree-width. *J. Comb. Theory, Ser. B*, 36(1), 1984.
[52] S. Saluja, K. Subrahmanyam, and M. Thakur. Descriptive complexity of #P functions. *JCSS*, 50(3), 1995.
[53] J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Math. Systems Theory*, 2(1), 1968.
[54] T. van Dijk, J.-P. van den Heuvel, and W. Slob. Computing treewidth with LibTW. Technical report, University of Utrecht, 2006.
[55] M. Y. Vardi. The complexity of relational query languages (extended abstract). In *STOC*, 1982.
[56] D. J. Watts and S. H. Strogatz. Collective dynamics of small-world networks. *Nature*, 393, 1998.
[57] F. Wei. TEDI: efficient shortest path query answering on graphs. In *SIGMOD*, 2010.
[58] T. Wolle. *Computational aspects of treewidth: Lower bounds and network reliability*. PhD thesis, Utrecht University, 2005.