Treewidth

Kimberly Crosbie

1 Introduction

The notion of treewidth is a powerful tool for solving graph theoretic problems. Treewidth generalizes the property of having small seperators. Intuitively, treewidth represents how "tree-like" a given graph is. Many problems that are NP-hard on general graphs can be solved in polynomial time for graphs with small treewidth. In this paper we will introduce the concepts of tree-decomposition and treewidth. Section 3 shows how problems that are NP-hard in general can be solved in polynomial time using tree-decopositions of small width, if the graph has bounded treewidth. Specifically, we detail an efficient algorithm for finding the size of the maximum independent set on a graph with bounded treewidth using a tree-decomposition. Finally, section 4 explains the problem of finding treewidth and tree-decompositions and summerizes some known results.

2 Definitions

We will now formally define important terms and give a few examples.

A *tree decomposition* of a graph $G = (V, E)$ is a tree $T = (I, F)$, where each node $i \in I$ has a label $X_i \subseteq V$ such that:

- $\bigcup_{i \in I} X_i = V$, or equivalently, every vertex $v \in V$ is contained in at least one label
- for any edge $(u, v) \in G$, there exists an $i \in I$ with $u, v \in X_i$
- for any $v \in V$, the vertices containing v in their label form a connected subtree of T, or equivalently, $i, j, k \in I$, if j is on the path from i to k in T, then $X_i \cap X_k \subseteq X_j$

To minimize confusion, common practice is to refer to v as a vertex when $v \in V$ and to refer to the nodes of T.

The *width* of a tree-decomposition $(\{X_i | i \in I\}, T = (I, F))$is $max_{i \in I} |X_i| - 1$. The *treewidth* of a graph G is the minimum width over all possible tree decompositions of G.

We use figure 1 as an example graph. Figure 2 gives an example of a tree-decomposition of the graph G that has width 3. Figure 3 gives an example of a tree-decomposition of G that has width 2, which is the treewdith of G. The colouring in figures 2 and 3 has been added for ease of understanding.

We can now make a few general observations. First of all, for any graph $G = (V, E)$, we always have a tree-decomposition, $(\{X_i | i \in I\}, T = (I, F))$, with a single node $i \in I$, that

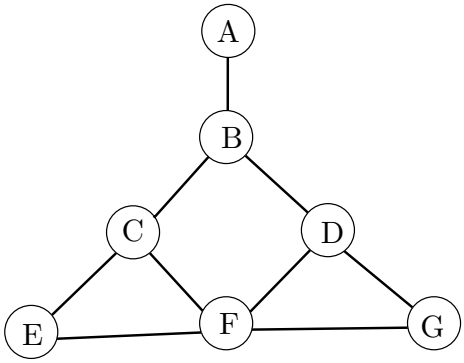

Figure 1: An example graph G with treewidth 2

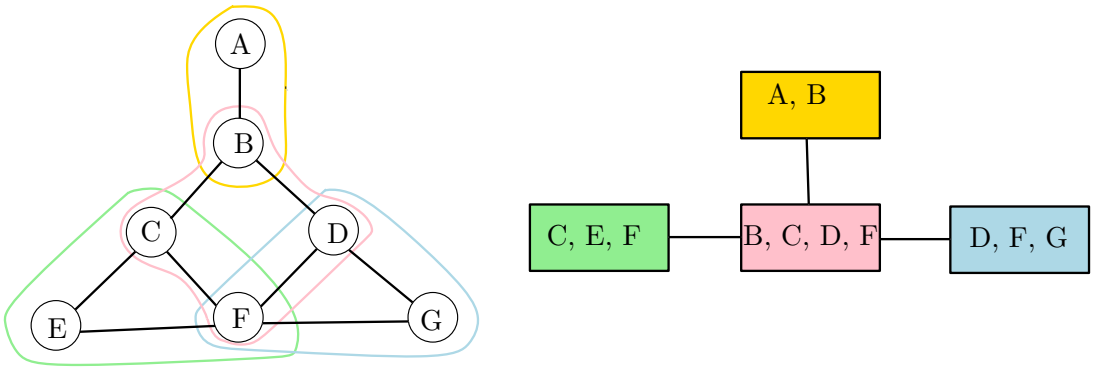

Figure 2: Graph G and a tree-decomposition of G with width 3

has a label X_i that contains all of the vertices in V. Such a tree-decomposition has width $n-1$. Next, we observe that if a graph is a tree or forest then it has treewidth 1. On the other hand, the complete graph K_n, has tree width $n-1$.

3 Algorithms with Bounded Treewidth

One of the nice properties of treewidth is that many graph problems that are NP-hard in the general case become easier to solve on graphs with small treewidth. In fact, many problems can be solved in polynomial time on graphs with small treewidth.

3.1 Maximum Independent Set

We now consider the problem of finding the size of the maximum independent set. In the general case, this problem is known to be NP-Complete [19]. However, if we are given a tree-decomposition of the graph where its treewidth is bounded by some constant k, then

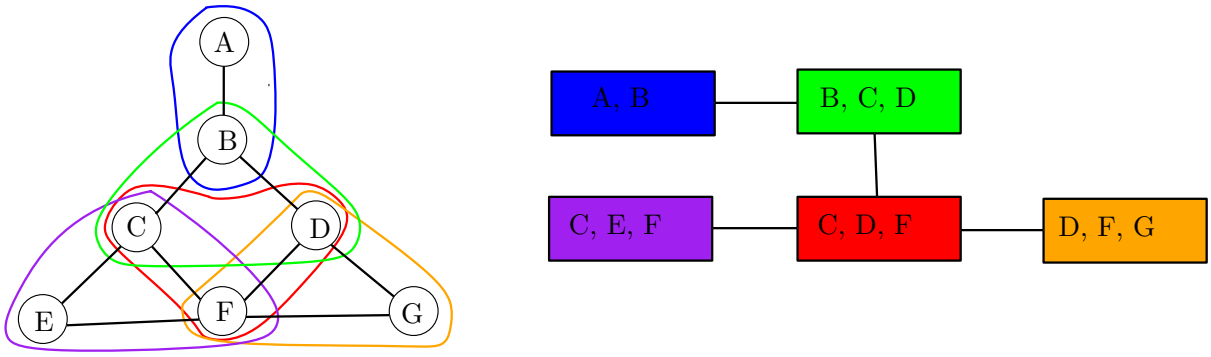

Figure 3: Graph G and a tree-decomposition of G with width 2

we can solve the problem in polynomial time.

As a reminder, the maximum independent set problem is as follows:
Given a graph $G = (V, E)$, we are looking for the maximum size of a set $W \subseteq G$ such that for all $u, v \in W$, the edge $uv \notin E$.

We will present Bodlaender's algorithm [6] to solve the problem by using dynamic programming.
Given a tree-decomposition of the graph G with treewidth k, it is easy to form another tree-decomposition, with treewidth k, that is a rooted binary tree. Suppose we have such a tree-decomposition $\{X_i \mid i \in I\}, T = (I, F)$ where r is the root of T. Then for each $i \in I$, we define $Y_i = \{v \in X_j \mid j = i$ or j is a descendent of $i\}$. Let $G[Y_i]$ be the subgraph of G induced by the vertices of Y_i.

While building the algorithm, the key observation is that when we have an independent set W of $G[Y_i]$ and want to extend that to an independent set of the full graph G we only need to know what vertices of X_i that belong to W. We need not consider what vertices of $Y_i - X_i$ are in W, we only need to know their number. This follows from the definition of tree-decomposition.

For $i \in I$, $Z \subseteq X_i$, define $S_i(Z)$ to be the maximum size of an independent set W in $G[Y_i]$ with $W \cap X_i = Z$. If no such set exists, we set $S_i(Z) = -\infty$.

By using dynamic programming, our algorithm solves the probelm by computing all tables of S_i for all nodes $i \in I$. We solve this problem in a bottom-up fashion by computing the tables of S_i after the tables for i's children have been computed.

To start, we compute the tables for each leaf node. For a leaf node i, we compute all $2^{|X_i|}$ values in the table S_i by using the following formula:

$$S_i(Z) = \begin{cases} |Z| & \text{if } \forall\ u, v \in Z : uv \notin E \\ -\infty & \text{if } \exists\ u, v \in Z : uv \in E \end{cases}$$

For an internal node i with children j and k, we compute S_i using the following formula:

$$S_i(Z) = \begin{cases} max\{S_j(Z') + S_k(Z'') + |Z \cap (X_i - X_j - X_k)| - |Z \cap X_j \cap X_k| & \\ |Z \cap X_j = Z' \cap X_i \text{ and } Z \cap X_k = Z'' \cap X_i\} & \text{if } \forall\ u, v \in Z : uv \notin E \\ -\infty & \text{if } \exists\ u, v \in Z : uv \in E \end{cases}$$

Intuitively, the idea for the previous formula is, we take the maximum over all sets Z' and Z'' where the sets Z and $Z' \subseteq X_j$ agree on which of their common elements are included in the maximum independent set and, similarly, Z and $Z'' \subseteq X_k$ agree on which of their common elements are included in the maximum independent set. The vertices in Z that are in X_i, but not X_j, nor X_k have not yet been counted, so we add $|Z \cap (X_i - X_j - X_k)|$. The vertices in $Z \cap X_j \cap X_k$ have been counted twice, hence we must subtract their number once.

For each node $i \in I$ we compute the table S_i in bottom-up order until we have computed the table S_r. Now, we can easily find the size of the maximum independent set of the graph G by taking $max_{Z \subseteq X_r} S_r(Z)$. Therefore, we can solve the problem in $O(2^{3k}n)$ time.

3.2 Monadic Second Order Logic

Interestingly, it has been proven that for a large class of problems, there is a linear time algorithm to solve a problem from the class if a tree-decomposition with constant bounded treewidth is provided. Specifically, Courcelle's results [17][15][16], which were extened by Borie et al. [12], Arnborg et al. [3] and Courcelle and Mosbah [18], state that if a graph problem can be expressed in monadic second order logic, then it can be solved in linear time on graphs with bounded treewidth. Monadic second order logic is a language to describe graph properties that uses the following constructions: logic operations ($\wedge, \vee, \neg, \Rightarrow$), membership tests (e.g, $e \in E$, $v \in V$), quantifications over vertices, edges, sets of vertices, sets of edges (e.g., $\forall v \in V$, $\exists e \in E$, $\exists I \subseteq V$, $\forall F \subseteq E$), adjacency tests (u is an endpoint of e), and some extensions that allow for things such as optimization.

For example, take the 3-colouring problem for graphs. For a graph $G = (V, E)$, this problem asks if it is possible to assign each vertex $v \in V$ one of the 3 colours such that no two adjacent vertices are assigned the same colour. This problem can be expressed in monadic second order logic as follows:
$\exists W_1 \subseteq V : \exists W_2 \subseteq V : \exists W_3 \subseteq V : \forall v \in V : (v \in W_1 \vee v \in W_2 \vee v \in W_3) \wedge \forall v \in V : \forall w \in W : (v, w) \in E \Rightarrow (\neg(v \in W_1 \wedge w \in W_1) \wedge \neg(v \in W_2 \wedge w \in W_2) \wedge \neg(v \in W_3 \wedge w \in W_3))$, where W_1, W_2, W_3 represent the subsets of vertices having each of the 3 colours. Therefore, by Courcelle's results, the 3-colouring problem can be solved in linear time for a graph given a tree-decomposition of bounded constant treewidth.

4 Finding Tree-decompositions

In this section, we summarize the results known for the problem of finding the treewidth of a graph and finding tree-decompositions.

4.1 Finding Treewidth

Given a graph $G = (V, E)$ and an integer k, the problem to determine if the treewidth of G is at most k had be proven to be NP-complete [2]. However, for some special classes of graphs, the treewidth can be computed in polynomial time (see for example, [22], [23], [21], [24]). Additionally, it has been shown that this problem is NP-complete some special classes (e.g. graphs with maximum degree at most 9 [11]; see also [20]). An interesting open problem is if this problem is NP-Complete for planar graphs. Table 1 [14] gives an overview of the current results for many classes of graphs.

Class	**Complexity**
Tree/Forest	Constant
Series-parallel graph	Constant
Outerplanar graph	Constant
k-Outerplanar graph	Constant
Hanlin graph	Constant
Chordal graph	Polynomial
Starlike chordal graph	Polynomial
k-Starlike chordal graph	Polynomial
Co-chordal graph	Polynomial
Split graph	Polynomial
Permutation graph	Polynomial
Circular permutation graph	Polynomial
Cograph	Polynomial
Chordal bipartite graph	Polynomial
Interval graph	Polynomial
Circular arc graph	Polynomial
Distance hereditary graph	Polynomial
Bounded Degree	NP-complete
Bipartite graph	NP-complete
Cocomparability graph	NP-complete
Planar graph	open

Table 1: Complexity status for some classes of graphs

4.2 Approximation Algorithms

The first approximation algorithm for finding a tree-decompostion was a polynomial time approximation algorithm that returns a tree-decomposition with width at most $O(\log n)$ times the optimal treewidth [10]. More recently, an approximation algorithm that gives a tree-decomposition of width at most $O(k \log k)$, where k is the treewidth of the input graph was presented [13].

If k is constant, then we have the result that there exists an $O(n \log n)$ algorithm, that given a graph $G = (V, E)$, either outputs that the treewidth of G is larger than k or outputs a tree-decomposition of G with treewidth at most $3k + 2$ (see [25], [6] for more details). Recently, an algorithm with runtime $O(c^k n)$ was presented [9]. The algorithm either outputs that the treewidth of G is larger than k or outputs a tree-decomposition of G with treewidth at most $5k + 4$. This algorithm is the first algorithm that gives a constant factor approximation for treewidth and runs in single-exponential in k and linear in n time.

5 Bibliographic Notes

Treewidth was first introduced by Robertson and Seymour [27]. At the same time, Arnborg and Proskurowski [1][4][5] were doing work on partial k-trees, which are equivalent to graphs that treewidth k. Many equivelent notions of treewidth exist; for an overview see [8]. A concept closely related to treewidth is pathwidth, which was first intoduced by Robertson and Seymour [26]. [6], [14] and [7] provide indepth surveys of results and concepts related to treewidth. The algorithm presented in section 3 was taken from [6]. Table 1 was taken from [14].

6 Exercises

1) From the definition of tree-decompostion, prove that the following two statements are equivalent:

- for any $v \in V$, the vertices containing v in their label form a connected subtree of T
- for all $i, j, k \in I$, if j is on the path from i to k in T, then $X_i \cap X_k \subseteq X_j$

2) a) Prove that a forest has treewdith 1.

b) Prove that the complete graph K_n has treewidth $n - 1$.

3) Given a graph $G = (V, E)$ with treewidth k, and a tree-decompoisiton $\{X_i \mid i \in I\}, T = (I, F)$ of G with width k, give an efficent algorithm that outputs a maximum independent set $V' \subseteq V$ of G.

References

[1] S. Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposabilitya survey. *BIT Numerical Mathematics*, 25(1):1–23, 1985.

[2] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in ak-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.

[3] S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2):308–340, 1991.

[4] S. Arnborg and A. Proskurowski. Characterization and recognition of partial k-trees. *Congr. Numer*, 47:69–75, 1985.

[5] S. Arnborg and A. Proskurowski. Characterization and recognition of partial 3-trees. *SIAM Journal on Algebraic Discrete Methods*, 7(2):305–314, 1986.

[6] H. L. Bodlaender. A tourist guide through treewidth. *Acta cybernetica*, 11(1-2):1, 1994.

[7] H. L. Bodlaender. *Treewidth: Algorithmic techniques and results*. Springer, 1997.

[8] H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical computer science*, 209(1):1–45, 1998.

[9] H. L. Bodlaender, P. G. Drange, M. S. Dregi, F. V. Fomin, D. Lokshtanov, and M. Pilipczuk. An o (c^ kn) 5-approximation algorithm for treewidth. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 499–508. IEEE, 2013.

[10] H. L. Bodlaender, J. R. Gilbert, H. Hafsteinsson, and T. Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *Journal of Algorithms*, 18(2):238–255, 1995.

[11] H. L. Bodlaender and D. M. Thilikosz. Treewidth and small separators for graphs with small chordality. 1995.

[12] R. B. Borie, R. G. Parker, and C. A. Tovey. Deterministic dcomposition of recursive graph classes. *SIAM Journal on Discrete Mathematics*, 4(4):481–501, 1991.

[13] V. Bouchitté, D. Kratsch, H. Müller, and I. Todinca. On treewidth approximations. *Discrete Applied Mathematics*, 136(2):183–196, 2004.

[14] D. Bronner and B. Ries. An introduction to treewidth. Technical report, 2006.

[15] B. Courcelle. Graph rewriting: An algebraic and logic approach. *Handbook of theoretical computer science*, pages 194–242, 1990.

[16] B. Courcelle. Graph grammars, monadic second-order logic and the theory of graph minors. *Contemporary Mathematics*, 147:565–565, 1993.

[17] B. Courcelle. The monadic second order logic of graphs vi: On several representations of graphs by relational structures. *Discrete Applied Mathematics*, 54(2):117–149, 1994.

[18] B. Courcelle and M. Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theoretical Computer Science*, 109(1):49–82, 1993.

[19] M. R. Garey and D. S. Johnson. Computers and intractability: a guide to np-completeness, 1979.

[20] M. Habib and R. H. Möhring. Treewidth of cocomparability graphs and a new order-theoretic parameter. *Order*, 11(1):47–60, 1994.

[21] T. Kloks. *Treewidth of circle graphs*. Springer, 1993.

[22] T. Kloks, H. L. Bodlaender, et al. *On the treewidth and pathwidth of permutation graphs*. Department of Computer Science, Utrecht University, 1992.

[23] T. Kloks, H. L. Bodlaender, et al. *Only few graphs have bounded treewidth*. Department of Computer Science, Utrecht University, 1992.

[24] T. Kloks and D. Kratsch. Treewidth of chordal bipartite graphs. *Journal of Algorithms*, 19(2):266–281, 1995.

[25] B. A. Reed. Finding approximate separators and computing tree width quickly. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 221–228. ACM, 1992.

[26] N. Robertson and P. D. Seymour. Graph minors. i. excluding a forest. *Journal of Combinatorial Theory, Series B*, 35(1):39–61, 1983.

[27] N. Robertson and P. D. Seymour. Graph minors. ii. algorithmic aspects of tree-width. *Journal of algorithms*, 7(3):309–322, 1986.