

Tree Decompositions, Treewidth, and NP-Hard Problems

A Survey Paper on Recent Findings in the Field

Gerrod Voigt

Mathematics
Massachusetts Institute of Technology

Abstract

This survey paper provides an introduction to the class of bounded treewidth graphs, for which many NP-hard problems can be solved efficiently. The notions of tree decomposition and treewidth are explained, and subclasses of bounded treewidth graphs are analyzed. Furthermore, recent findings about tree decomposition and treewidth are summarized. Some (but not all) problems which are efficiently solvable on bounded treewidth graphs are mentioned. In particular, solutions to two of them, maximum weight independent set and coloring, are fully sketched in this paper.

Introduction

Much of the difficulty in solving NP-Hard problems lies in unraveling the network of connections between constraints. However, if we restrict the situation to where there are more organized or fewer constraints, some of these NP-Hard problems become much less difficult, and are often solvable in polynomial time via dynamic programming. Thus, for NP-Hard problems, it is important to know which problem instances have efficient algorithms/solutions available.

This survey paper shows how some (but not all) NP-hard graph problems can be solved efficiently if the graph under consideration has certain properties (namely, bounded treewidth - treewidth to be defined in section 2). Since trees are generally algorithmically easy to deal with, mathematicians and computer scientists seek graph classes that behave similarly to trees. The class of bounded treewidth graphs turns out to have precisely these behaviors.

Since its invention in 1984 by Robertson and Seymour [1], treewidth has been shown to be bounded for several important classes of graphs:

- Trees / Forests (treewidth 1)
- Series-parallel graphs (treewidth 2)
- Outerplanar graphs (treewidth 2)
- Halin graphs (treewidth 3)

However, it should be noted that not all classes of graphs have bounded treewidth. Among the most notable of these classes of graphs that do not have bounded treewidth are planar graphs and bipartite graphs.

According to Bodlaender [2], many practical problems rely heavily on graphs with bounded treewidth. Among these problems are:

- *Expert systems* - Graphs modelling certain types of expert systems have been observed to have small treewidth, which allows otherwise time-consuming statistical computations for reasoning to be performed efficiently on these graphs.

- *Evolution theory* - In phylogeny, one wants to construct evolution trees for species and their characteristics for the purpose extrapolating possible extinct ancestors. One variant of these clustering techniques is the perfect phylogeny problem, a graph problem with a vertex colored graph. It has been shown that the necessary condition for the solution to this perfect phylogeny problem is that the treewidth of the graph be smaller than the number of colors.
- *Natural language processing (NLP)* - It has been observed, that dependency graphs of the major syntactic relations among words typically have small tree-width. Thus, these dependency graphs can thereby be efficiently exploited for language processing.

In this survey paper, the relationships between tree decomposition, treewidth, and NP-Hard problems are discussed. In section 1, a motivating example to the upcoming concepts of treewidth and tree decomposition is shown. Namely, a short example is provided for which, in general, no deterministic polynomial time algorithm is known, but which turns out to be efficiently solvable for bounded treewidth graphs. In section 2, tree decomposition and treewidth are introduced, and some of their more important properties are explored. In section 3, equivalent definitions of treewidth are analyzed as a means of bounding treewidth for certain graphs. Finally, in section 4, efficient algorithms are presented for bounding the treewidth of a graph G and for finding a tree decomposition corresponding to that bounded treewidth. Additionally, a generalization is made about which NP-Hard problems can be solved efficiently using tree decomposition, and more organized tree decomposition variants are presented for effective use with dynamic programming algorithms.

1. Dynamic Programming on Trees

As an introductory example to the upcoming concepts of tree decomposition and treewidth, consider the maximum weight independent set problem on trees. The input is a tree $G(V, E)$ (with set of vertices V and set of undirected edges E) and a weight function $w : V \rightarrow \mathbb{R}$. We wish to find an independent set (a set of vertices in which no two vertices are adjacent to one another) of maximum weight. Although the maximum weight independent set problem is NP-Hard for general graphs, it can be solved in polynomial time for trees using the following dynamic programming algorithm.

Fix an arbitrary vertex R as the root of the tree, and orient all edges away from R . The subtree rooted at v , denoted by T_v , includes v and all vertices reachable from v under this orientation of edges (implying that $T_R = G(V, E)$). The children of v , denoted by C_v are all those vertices u with an edge $(v, u) = (u, v)$. Leaves of the tree have no children. This tree is drawn below, with labeled vertex v and T_v highlighted in red:

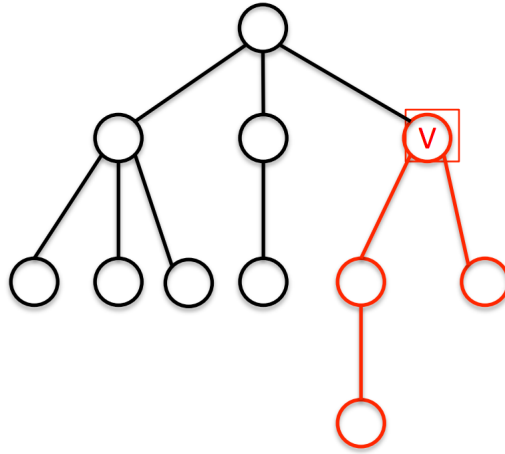


Figure 1: A tree with labelled vertex v and subtree T_v colored red

Let $W[v]$ denote the maximum weight of an independent set of T_v . We want to compute $W[r]$. Let $W^+[v]$ denote the maximum weight of an independent set of T_v that contains v , and let $W^-[v]$ denote the maximum weight of an independent set of T_v that does not contain v . This then provides the following recursive relations:

$$W^+[v] = w(v) + \sum_{u \in C_v} W^-[u] \quad (1)$$

and

$$W^-[v] = \sum_{u \in C_v} \max\{W^-[u], W^+[u]\} \quad (2)$$

In order to achieve the desired result, the dynamic programming algorithm adheres to the following procedure:

1. Select a vertex v such that for all of its children $u \in C_v$ we have already computed $W^-[u]$ and $W^+[u]$.
2. Compute $W^-[v]$ and $W^+[v]$ using the recursive relations defined above.

Iteratively applying this dynamic programming algorithm, we eventually output $\max\{W^-[r], W^+[r]\}$. In other words, we eventually find the maximum weight independent set of the graph. Since every vertex is visited once and applying the recursive relations takes $O(1)$ time, then the overall runtime is $O(V)$. Thus, although the maximum weight independent set problem is NP-Hard for general graphs, we have just shown that it can be solved in linear time for trees.

In fact, many NP-hard graph problems are polynomially solvable on trees (maximum independent set, Tutte polynomial calculation, coloring, etc.). However, it must be noted that not all NP-hard problems are polynomially solvable on trees (namely finding bandwidth). We shall show how the class of graphs for which the above dynamic programming approach works can be significantly generalized.

2. Tree Decomposition and Treewidth

In this section, we introduce the notions of tree decomposition and treewidth, and we examine some of their more general properties.

2A. Tree Decomposition and Treewidth Terminology

Definition 1: A *tree decomposition* of a graph $G(V, E)$ is a tree T , where

1. Each vertex i of T is labeled by a subset $B_i \subset V$ of vertices of G , referred to as a “bag”
2. Each edge of G is in a subgraph induced by at least one of the B_i (i.e. is in at least one of the “bags” of T)
3. The subtree of T consisting of all “bags” containing u is connected, for all vertices u in G .

It is important to note that a graph may have several different tree decompositions (tree decompositions are generally not unique). Similarly, the same tree decomposition can be valid for several different graphs. An example of a tree decomposition is shown in Figure 2.

Every graph has a trivial tree decomposition for which T has one vertex including all of V . However this is ineffective for the purpose of solving NP-hard problems, so we define the following:

Definition 2: The *width* of a tree decomposition is one less than the maximum bag size of that tree decomposition.

The reason for subtracting 1 in the above definition for width is so that we can define forests as having treewidth 1 (this proposition is proven later in this section).

Definition 3: The *treewidth* of $G(V, E)$ is the minimum integer k such that there exists a tree decomposition of $G(V, E)$ with “bags” of size at most $k + 1$

From this definition, we can readily observe that the tree-width of a graph is equal to the maximum of the tree-widths of its connected components.

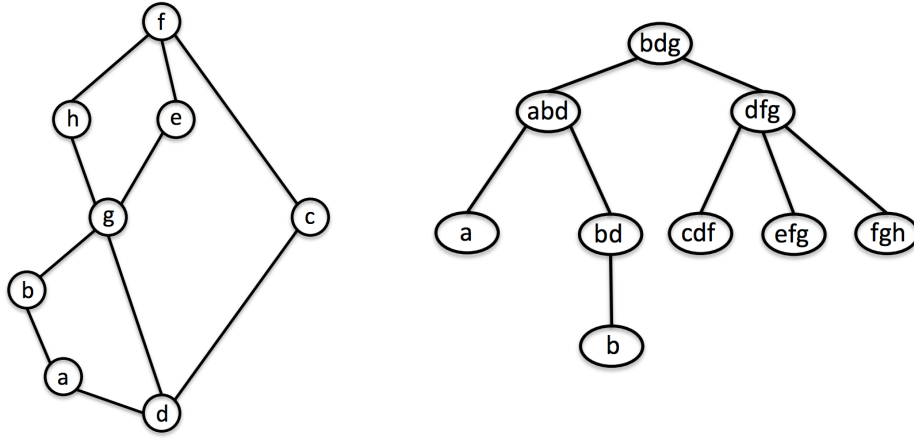


Figure 2: A graph $G(V,E)$ (left), and a valid tree decomposition (right)

2B. Facts About Treewidth

The following facts hold about tree decompositions and treewidth:

Proposition 1: Every graph with tree-width k has a vertex of degree at most k

Proof: For graph G , consider a tree decomposition T with width k . Consider a leaf l of T . Its “bag” includes a vertex v that is not in the “bag” of its neighbor in T , and hence not in any other “bag” either. As $|B_l| \leq k + 1$, it follows that the degree of l in G is at most k .

Proposition 2: An n -vertex graph has tree-width $n - 1$ iff it is a clique

Proof: If the graph is not a clique, then some edge (u, v) must be missing. In this case, we can take $B_1 = V \setminus \{u\}$ and $B_2 = V \setminus \{v\}$ as a tree decomposition with two nodes and treewidth of $n - 2$.

Conversely, if G is connected and has a tree decomposition with treewidth below $n - 1$, then it has a vertex of degree at most $n - 2$ (from Proposition 1), and thus cannot be a clique.

Proposition 3: G has treewidth at most 1 iff it is a forest

Proof: Consider an arbitrary tree G . We can build a tree decomposition for it with treewidth 1 as follows: 1) Pick a root R in G , and orient the edges away from R . Each oriented edge (u, v) in G makes a node labeled $\{u, v\}$ in T (The label $\{u, v\} = \{v, u\}$, but we will keep the orientation of the edge for clarity’s sake). Two nodes $\{u_1, v_1\}$ and $\{u_2, v_2\}$ are adjacent in T iff $v_1 = u_2$. This then provides a tree decomposition of G .

Conversely, assume that G has tree-width 1. Then it has a vertex of degree at most 1 (from Proposition 1). Removing this vertex from G , the tree-width cannot increase. Hence we can recursively remove vertices of degree at most 1, until we exhaust all vertices in G . This implies that G is a forest.

2C. Series-Parallel Graphs

Definition 4: A series-parallel graph is a graph obtained from an independent set using the following operations:

1. Add a new node and connect it to an existing node by an edge
2. Add a self loop
3. Add an edge in parallel to an existing edge
4. Subdivide an edge by creating a node in the middle

Figure 3 illustrates the two ways that two series/parallel circuits may be combined.

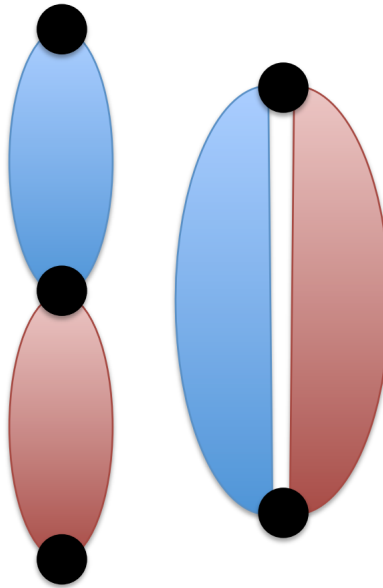


Figure 3: Series combination (left), parallel combination (right)

Proposition 4: G has tree-width at most 2 iff it is a series-parallel graph

Proof: Let G be a series-parallel graph (SPG). We prove this proposition by building a tree decomposition for the SPG inductively, without ever exceeding a treewidth of 2. We do this using the following inductive construction of G . The most involved step is step 4 in the construction of series-parallel graphs. Suppose edge (u, v) was subdivided (by step 4), introducing a vertex z . Prior to the subdivision, the tree decomposition included a vertex j such that $u, v \in B_j$. Create a new node labeled by $\{u, v, z\}$ and connect it to j .

Conversely, assume a tree decomposition T with treewidth at most 2. We now prove that this corresponds to a series-parallel graph. We do this by: 1) removing a node from the tree decomposition, 2) showing that this corresponds to removing a vertex from the graph G , and 3) showing that G can be obtained by adding back the vertex using Definition 4. Again we illustrate only one case. Take a leaf l of T . Assume that it is labeled by three vertices u, v, z (since T has treewidth at most 2, it can't have more than three vertices). Then, without loss of generality, the neighbor of l in T does not contain z . Hence the only possible neighbors of z in G are u and v . If both are indeed neighbors, then z could have been obtained by subdividing an edge (u, v) , and the graph prior to subdividing also had tree width at most 2 (by removing z from l).

(Right now this proof is not finished. I'm working on simplifying it so that I may cover all cases in Definition 4 without having to take up pages in my paper for the proof).

3. Alternative Definitions of Treewidth

As a general measure of sparseness of a graph, treewidth has many equivalent definitions. This section aims to explore a few of these, specifically via chordal graphs and cops and robbers.

3A. Chordal Graphs

Definition 5: A graph G is chordal if every cycle on at least four vertices has at least one chord. We can add edges to a non-chordal graph to make it chordal. This process of adding edges to a non-chordal graph to make it chordal is referred to as triangulation.

A tree decomposition provides an inherent method of triangulation. More specifically, we can draw all edges between vertices in each “bag” that don't already exist. Figure 4 displays an example of triangulation. The graph on the left (in Figure 4) is prior to triangulation, while the graph on the right is after triangulation (with the added edges colored red).

Proposition 5: A graph G has treewidth at most k iff there is a triangulation of G with no $k + 2$ clique.

The forward implication is guaranteed by the triangulation of bags. The

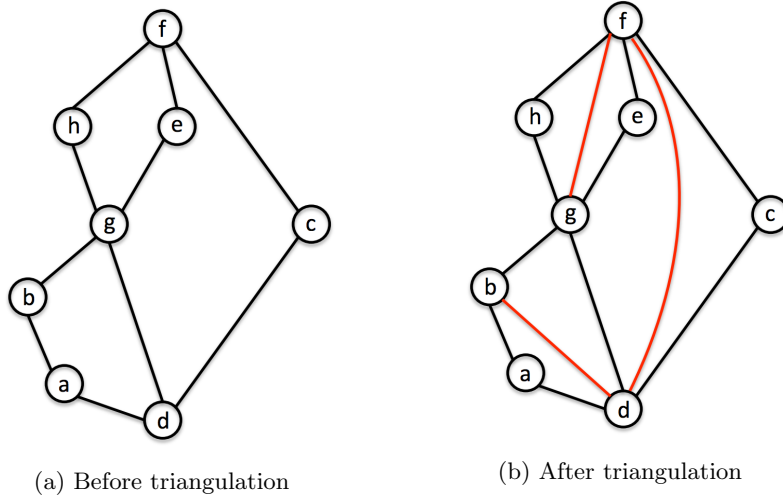


Figure 4: Figures displaying the process of triangulation

following concept and result provide a means for handling the converse implication:

Definition 6: A perfect elimination ordering is an ordering of the vertices of G such that the higher indexed neighbors of any vertex form a clique. That is, if we order the vertices V_1, V_2, \dots, V_n , then for all $i, j, k \in [n]$ satisfying $i < j < k$ for which V_i is adjacent to both V_j and V_k , V_j is adjacent to V_k .

Theorem 1: G is chordal if and only if there exists a perfect elimination ordering of the vertices of G . (for proof see [9])

It follows that any graph with treewidth at most k has an ordering of its vertices such that each vertex has at most k higher indexed neighbors (and is thus $k + 1$ - colorable). The converse, however, is not true.

3B. Cops and Robbers

Theorem 2: The treewidth of a graph G is at most k if and only if $k + 1$ cops can win the cops and robbers game on G .

Proof: In the cops and robbers game, you have the following:

- a robber attempts to escape capture by “hiding out” in a graph $G(V,E)$
- cops may decide either to station themselves at a vertex of $G(V,E)$ or hover above the graph, and, at any time, may land at or take off from a vertex

- the robber is allowed to move along the edges of $G(V,E)$ instantaneously, but is blocked by vertices occupied by cops
- when a cop begins to descend onto a vertex, the robber is able to see this, and may select to move wherever he desires before the cop lands
- the cops win if they can trap the robber at a vertex, while the robber wins if he can escape capture indefinitely.

The tree decomposition of a graph inherently provides a winning strategy. First, the cops station themselves at the vertices in the root of the tree decomposition. Depending on where the robber hides, the cops change their positions to the appropriate subtree. The cops proceed to work down the tree decomposition until they reach a leaf and catch the robber, thereby winning the game.

The other direction (converse implication) is much more involved. For proof of this converse implication, see [10].

4. Tree Decompositions and NP-Hard Problems

In this section, we explore the following process: 1) bounding the treewidth of a graph, 2) finding the corresponding tree decomposition for this bounded treewidth, and 3) using dynamic programming on tree decompositions.

4A. Bounding Treewidth

While determining the exact treewidth of a graph is NP-Hard, there are algorithms that can determine whether the treewidth is at most k in time $O(n^k)$.

Take, for example, the following. As proved above, the treewidth of a graph G is at most k if and only if $k + 1$ cops can win the cops and robbers game on G . Thus, one way to determine whether the treewidth of a graph G is at most k is to attempt all robber hunting strategies for $k + 1$ cops on the graph. Well, the number of cops required to win the cops and robbers game is defined as the “cop number” of a graph G , so this is analogous to finding if the “cop number” of a graph is at most $k + 1$. Berarducci et al. proved in 1993 that this task can be performed in $O(n^k)$ time. [11]

4B. Finding a Tree Decomposition

Many NP-Hard problems can be solved in polynomial time on graphs with constant treewidth. Once treewidth has been bounded, the next important step is to find a usable tree decomposition. The following theorems guarantee that we can do this:

Theorem 3: In a graph on n vertices with treewidth k , there is an algorithm that will return a tree decomposition with width k in time $n^{k+O(1)}$. [3]

For large n , the algorithm provided by *ACP* can take quite a long time to find a tree decomposition with width k . Thus, in these cases, the following can be much more useful:

Theorem 4: In a graph on n vertices with treewidth k , there is an algorithm that will return a tree decomposition with width k in time $2^{\tilde{O}(k^3)n}$. [2]

If one is at liberty to sacrifice larger treewidth for faster running time, then the following becomes useful:

Theorem 5: In a graph on n vertices with treewidth k , there is an algorithm that will return a tree decomposition with width $5k + 4$ in time $2^{O(k)n}$. [4]

This final theorem is important as it removes the dependence in the running time of finding a tree decomposition from k :

Theorem 6: In a graph on n vertices with treewidth k , there is an algorithm that will return a tree decomposition with width $O(k\sqrt{\log k})$ in $\text{poly}(n)$ time. [7]

4C. Dynamic Programming and Tree Decompositions

This is going to be the last section of the paper, outside of the conclusion (still working on it right now). Will be added in for the final version. This section will talk about:

- Which graph properties can be solved in polynomial time for bounded treewidth
- Smooth decompositions
- nice decompositions
- an example of dynamic programming and tree decomposition on coloring

5. Conclusion

This has yet to be finished, but will be a combination of the results shown in this survey paper and the introduction written above.

References

- [1] N. Robertson and P. D. Seymour. *Graph minors. ii. algorithmic aspects of tree-width*. Journ Alg, 7:309322, 1986.
- [2] Hans L. Bodlaender. *A linear-time algorithm for finding tree-decompositions of small treewidth*. SIAM Journal on Computing, 25(6):13051317, 1996.
- [3] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. *Complexity of finding embeddings in a k -tree*. SIAM Journal on Algebraic and Discrete Methods, 8(2):277284, April 1987.
- [4] Hans L. Bodlaender, Pi G. Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Loksharov, and Micha Pilipczuk. *A $o(c^k n)$ 5-approximation algorithm for treewidth*. CoRR, abs/1304.6321, April 2013.
- [5] Bruno Courcelle, J. A. Makowsky, and U. Rotics. *On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic*. Discrete Applied Mathematics, 108(1-2):2352, February 2001.
- [6] Erik D. Demaine, MohammadTaghi Hajiaghayi, and Bojan Mohor. *Approximation algorithms via contraction decomposition*. Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, pages 278287, 2007.
- [7] Uriel Feige, MohammadTaghi Hajiaghayi, and James R. Lee. *Improved approximation algorithms for minimum weight vertex separators*. SIAM Journal on Computing, 38(2):629657, 2008.
- [8] D. Eppstein. *Diameter and treewidth in minor-closed graph families*. Algorithmica, 27(3-4):275291, January 2000.
- [9] D. R. Fulkerson and O. A. Gross. *Incidence matrices and interval graphs*. Pacific Journal of Mathematics, 15(3):835855, 1965.
- [10] P.D. Seymour and R. Thomas. *Graph searching and a min-max theorem for treewidth*. Journal of Combinatorial Theory, Series B, 58(1):2223, May 1993.
- [11] Berarducci, Alessandro; Intrigila, Benedetto; *On the cop number of a graph*. Adv. in Appl. Math. 14 (1993), no. 4, 389–403.
- [12] O’Donnell, Ryan; *Treewidth and Tree Decomposition*. CMU 18-859T. A Theorist’s Toolkit. 2013