Treewidth: Algorithmic Techniques and Results

Hans L. Bodlaender

Department of Computer Science, Utrecht University
P.O. Box 80.089, 3508 TB Utrecht, the Netherlands

Abstract. This paper gives an overview of several results and techniques for graphs algorithms that compute the treewidth of a graph or that solve otherwise intractable problems when restricted graphs with bounded treewidth more efficiently. Also, several results on graph minors are reviewed.

1 Introduction

The notion of treewidth is playing a central role in many recent investigations in algorithmic graph theory. There are several reasons for the interest in this, at first sight perhaps somewhat unnatural notion. One of these reasons is the central role that the notion plays in the theory on graph minors by Robertson and Seymour (see Section 5); another reason is that many problems that are otherwise intractable become polynomial time solvable when restricted to graphs of bounded treewidth (see Section 4).

There are several 'real world' applications of the notion of treewidth, amongst others in expert systems [93], telecommunication network design ([46]), VLSI-design, Choleski factorization, natural language processing [91] (see e.g. [21] for a brief overview.) An interesting recent application has been found by Thorup [131]. He shows that for many well known programming languages (like C, Pascal, Modula-2), the control-flow graph of goto-free programs has treewidth bounded by a small constant (e.g., 3 for Pascal, 6 for C). Thus, certain optimization problems arising in compiling can be solved using techniques relying on small treewidth.

2 Definitions

The notion of treewidth was introduced by Robertson and Seymour in their work on graph minors [104].

Definition 1. A *tree decomposition* of a graph $G = (V, E)$ is a pair $(\mathcal{X}, T)$ with $T = (I, F)$ a tree, and $\mathcal{X} = \{X_i \mid i \in I\}$ a family of subsets of V, one for each node of T, such that

- $\bigcup_{i \in I} X_i = V$,
- for all edges $\{v, w\} \in E$ there exists an $i \in I$ with $v \in X_i$ and $w \in X_i$, and
- for all $i, j, k \in I$: if j is on the path from i to k in T, then $X_i \cap X_k \subseteq X_j$.

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

There are several equivalent notions to treewidth (for an overview, also of classes of graphs that have a uniform upper bound on the treewidth, see [25]); amongst others, graphs of treewidth at most k are also known as *partial k-trees*. A notion related to treewidth is *pathwidth*, defined first in [102]. A tree decomposition $(\mathcal{X},T)$ is a path decomposition if T is a path; the pathwidth of a graph G is the minimum width over all path decompositions of G. A survey giving relations to notions of graph searching has been written by Bienstock [14].

Another notion that is related to treewidth and that might be more suitable in some cases for implementation purposes is *branchwidth* [118].

A tree decomposition can easily be converted (in linear time) in a *nice* tree decomposition of the same width (and with a linear size of T): here the tree T is rooted and binary, and nodes are of four types:

- Leaf nodes i are leaves of T and have $|X_i|=1$.
- Introduce nodes i have one child j with $X_i = X_j \cup \{v\}$ for some vertex $v \in V$.
- Forget nodes i have one child j with $X_i = X_j - \{v\}$ for some vertex $v \in V$.
- Join nodes i have two children j with $X_i = X_{j_1} = X_{j_2}$.

Using nice tree decompositions instead of normal ones does in general not give additional algorithmic possibilities, but it considerably eases the design of algorithms, and one can also expect in several cases to have better constant factors in the running times of algorithms that use nice instead of normal tree decompositions.

3 Determining treewidth

In this section we review a number of results on the problem, to determine the treewidth of a given graph.

The problem to determine, when given a graph G and an integer k, whether the treewidth of G is at most k is NP-complete [5], even for graphs of maximum degree at most 9 [36], bipartite graphs, or cocomparability graphs. For several special graph classes, there exist polynomial time algorithms to determine the treewidth of graphs in the class, e.g., for chordal graphs, permutation graphs [33], circular arc graphs [127], circle graphs [87], and distance hereditary graphs [40]. See also [34, 60, 72, 78, 79, 89]. One of the most interesting open problems here is the complexity of treewidth when restricted to planar graphs. As branchwidth can be solved in polynomial time on planar graphs [126], and branchwidth differs at most a factor 1.5 from treewidth, we have a polynomial time approximation algorithm for treewidth on planar graphs with performance ratio 1.5. For arbitrary graphs, there is a polynomial time approximation algorithm for treewidth with performance ration $O(\log n)$ [30]; it is an interesting (but probably hard) open problem whether treewidth can be approximated with constant performance ratio.