TUTORIALS IN
OPERATIONS RESEARCH
INFORMS—New Orleans 2005

 | ISBN 0000-0000
DOI 10.1287/educ.1053.0000

Branch and Tree Decomposition Techniques for Discrete Optimization

Illya V. Hicks
Department of Industrial Engineering, Texas A & M University, College Station, Texas 77843-3131, USA, ivhicks@tamu.edu

Arie M. C. A. Koster
Zuse Institute Berlin (ZIB), Takustraße 7, D-14195 Berlin, Germany, koster@zib.de

Elif Kolotoğlu
Department of Industrial Engineering, Texas A & M University, College Station, Texas 77843-3131, USA, elif@tamu.edu

Abstract This chapter gives a general overview of two emerging techniques for discrete optimization that have footholds in mathematics, computer science, and operations research: branch decompositions and tree decompositions. Branch decompositions and tree decompositions along with their respective connectivity invariants, branchwidth and treewidth, were first introduced to aid in proving the Graph Minors Theorem, a well-known conjecture (Wagner's conjecture) in graph theory. The algorithmic importance of branch decompositions and tree decompositions for solving $\mathcal{NP}$-hard problems modelled on graphs was first realized by computer scientists in relation to formulating graph problems in monadic second order logic. The dynamic programming techniques utilizing branch decompositions and tree decompositions, called branch decomposition and tree decomposition based algorithms, fall into a class of algorithms known as fixed-parameter tractable algorithms and have been shown to be effective in a practical setting for $\mathcal{NP}$-hard problems such as minimum domination, the travelling salesman problem, general minor containment, and frequency assignment problems.

Keywords Branchwidth, Treewidth, Graph algorithms, Combinatorial Optimization

1. Introduction

The notions of branch decompositions and tree decompositions and their respective connectivity invariants, branchwidth and treewidth, are two emerging techniques for discrete optimization that also encompass the fields of graph theory, computer science, and operations research. The origins of branchwidth and treewidth are deeply rooted in the proof of the Graph Minors Theorem, formally known as Wagner's conjecture. Briefly, the Graph Minors Theorem states that in an infinite list of graphs there would exist two graphs H and G such that H is a minor of G. The algorithmic importance of the branch decomposition and tree decomposition was not realized until Courcelle [50] and Arnborg et al. [13] showed that several $\mathcal{NP}$-hard problems posed in monadic second-order logic can be solved in polynomial time using dynamic programming techniques on input graphs with bounded treewidth or branchwidth. A problem which is $\mathcal{NP}$-hard implies that as long as it is not proven that $\mathcal{P} = \mathcal{NP}$ we cannot expect to have a polynomial time algorithm for the problem. These techniques are referred to as tree decomposition based algorithms and branch decomposition

based algorithms, respectively. Branch decomposition and tree decomposition based algorithms are important in discrete optimization because they have been shown to be effective for combinatorial optimization problems like the ring-routing problem [47], the travelling salesman problem [48], frequency assignment [87], general minor containment [73], and the optimal branch decomposition problem [74].

The procedure to solve an optimization problem with bounded branchwidth or treewidth involves two steps: (i) computation of a (good) branch/tree decomposition, and (ii) application of an algorithm that solves instances of bounded branchwidth/treewidth in polynomial time. Since the branchwidth or treewidth is considered to be a constant, not part of the input, this value may occur in the exponent of the complexity of both running time and space requirements. Hence, it is important to have a decomposition of width as small as possible. The problem of minimizing this quantity is however $\mathcal{NP}$-hard in itself.

Note that not every combinatorial problem defined on a graph of bounded branchwidth or treewidth can be solved in polynomial time. An example is the bandwidth minimization problem which is $\mathcal{NP}$-hard even on ternary trees (every vertex has degree one or three) [59, 95]. Even if the problem is polynomial on trees, the problem need not to be polynomial on graphs of bounded treewidth: $L(2,1)$-coloring is $\mathcal{NP}$-complete for graphs with treewidth 2 [54]. For more information on $L(2,1)$-colorings, one is referred to the work of Chang and Kuo [42] and the work of Bodlaender and Fomin [30].

Besides using the theory of monadic second-order logic, whether or not the problem can be solved in polynomial time on graphs of bounded branchwidth or treewidth can be discovered by investigating characteristics of the solution. Given a vertex cut set, one has to answer the question what impact the solution on one side of the cut set has on the solution on the other side. If the solutions only depend on the solution in the vertex cut, the problem likely can be solved with a dynamic programming algorithm specialized for the problem.

This chapter gives a general overview of branchwidth and treewidth along with their connections to structural graph theory, computer science, and operations research. Section 2 offers preliminary and relevant definitions in the subject area. Section 3 offers some interesting background on the Graph Minors Theorem and its relation to branchwidth and treewidth. Section 4 describes algorithms to construct branch decompositions as well as a blueprint for branch decomposition based algorithms. Section 5 offers similar results for treewidth with the addition of algorithms for computing relevant lower bounds to treewidth. Section 6 describes the extension of branchwidth and treewidth to matroids and Section 7 describes relevant open problems in the area. It is our hope that this chapter will spark interest into this fascinating area of research.

2. Definitions

2.1. Graph definitions

In this section we give basic definitions. The reader may skip this section and refer to it when necessary.

A *graph* is an ordered pair (V, E) where V is a nonempty set, called the set of *vertices* or *nodes*; E, the set of *edges*, is an unordered binary relation on V. A graph is called *complete* if all possible edges between the nodes of the graph are present in the graph. A *hypergraph* is an ordered pair (V, E) of nodes and edges, and an incidence relationship between them that is not restricted to two ends for each edge. Thus, edges of hypergraphs, also called *hyperedges*, can have any number of ends.

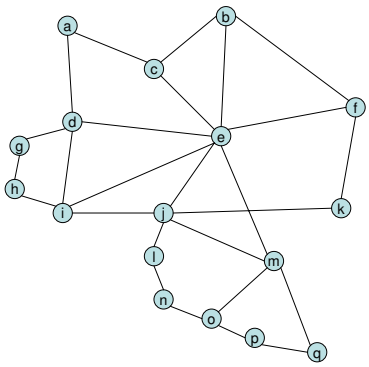


FIGURE 1. Example graph.

A graph $\bar{G} = (\bar{V}, \bar{E})$ is a *subgraph* of the graph $G = (V, E)$ if $\bar{V} \subseteq V$ and $\bar{E} \subseteq E$. For a subset $V' \subseteq V$, $G[V']$ denotes the graph induced by V', i.e., $G[V'] = (V', E \cap (V' \times V'))$. For a subset $E' \subseteq E$, the graph induced by these edges is denoted by $G[E']$. *Contraction* of an edge e means deleting that edge and identifying the ends of e into one node. Parallel edges are identified as well. A graph H is a *minor* of a graph G if H can be obtained from a subgraph of G by a series of contractions. A *subdivision* of a graph G is a graph obtained from G by replacing its edges by internally vertex disjoint paths.

The *degree* of a vertex is the number of edges incident with that vertex. A graph is *connected* if every pair of vertices can be joined by a path. The *connectivity* of a graph is the smallest number of vertices which can be removed to disconnect the graph. A graph that does not contain any cycles (*acyclic*) is called a *forest*. A connected forest is called a *tree*. The *leaves* of a tree are the vertices of degree one.

A graph $G = (V, E)$ is *bipartite* if V admits a partition into two classes such that every edge has its ends in different classes: vertices in the same partition class must not be adjacent. A bipartite graph is *complete* if all possible edges between the nodes of the graph, while maintaining the restriction of the bipartition, are present in the graph. A graph is *planar* if it can be embedded in a plane such that no two edges cross. The *incidence graph* $I(G)$ of a hypergraph G is the simple bipartite graph with vertex set $V(G) \cup E(G)$ such that $v \in V(G)$ is adjacent to $e \in E(G)$ if and only if v is an end of e in G. Seymour and Thomas [116] define a hypergraph H as *planar* if and only if $I(H)$ is planar. Also, a hypergraph G is called *connected* if $I(G)$ is connected. For an edge e, $\eta(e)$ is the number of nodes incident with e. The largest value $\eta(e)$ over all $e \in E$ is denoted by $\eta(G)$.

2.2. Branch decompositions

Let $G = (V, E)$ be a hypergraph and T be a ternary tree (a tree where every non-leaf node has degree 3) with $|E(G)|$ leaves. Let ν be a bijection (one-to-one and onto function) from the edges of G to the leaves of T. Then the pair (T, ν) is called a *branch decomposition* of G [106].

A *partial branch decomposition* is a branch decomposition without the restriction of every non-leaf node having degree 3. A *separation* of a graph G is a pair (G_1, G_2) of subgraphs with $G_1 \cup G_2 = G$ and $E(G_1 \cap G_2) = \emptyset$, and the *order* of this separation is defined as $|V(G_1 \cap G_2)|$.

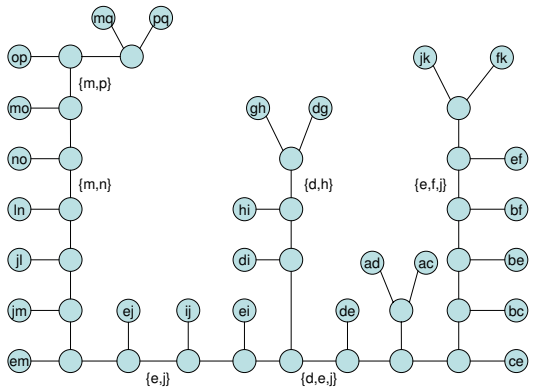


FIGURE 2. Branch decomposition of width 3 for the graph of Figure 1.

Let (T, ν) be a branch decomposition. Then removing an edge, say e, from T partitions the edges of G into two subsets A_e and B_e. The *middle set* of e, denoted $mid(e)$, is the set of vertices of G that are incident to the edges in A_e and the edges in B_e and the *width* of an edge e, denoted $|mid(e)|$, is the order of the separation $(G[A_e], G[B_e])$. The *width* of a branch decomposition (T, ν) is the maximum width among all edges of the decomposition. The *branchwidth* of G, denoted by $\beta(G)$, is the minimum width over all branch decompositions of G. A branch decomposition of G with width equal to the branchwidth is an *optimal branch decomposition* of G. Figure 2 illustrates an optimal branch decomposition of the graph given in Figure 1.

Robertson and Seymour [106] characterized the graphs which have branchwidth ≤ 2 and showed that $n \times n$-grid graphs have branchwidth n. Other known classes of graphs with known branchwidth are cliques whose branchwidth is $\lceil (2/3)|V(G)| \rceil$. For chordal graphs, the branchwidth of this class of graphs is characterized by $\lceil (2/3)\omega(G) \rceil \leq \beta(G) \leq \omega(G)$ where $\omega(G)$ is the maximum clique number of G [71, 106]. A *triangulated* or *chordal* graph is a graph in which every cycle of length at least four has a chord. Related to chordal graphs, another connectivity invariant related to branchwidth called *strong branchwidth* was developed by Tuza [123].

2.3. Tangles

A *tangle* in G of *order* k is a set $\mathcal{T}$ of separations of G, each of order $< k$, such that:

(T1) for every separation (A, B) of G of order $< k$, one of (A, B), (B, A) is an element of $\mathcal{T}$;
(T2) if (A_1, B_1), (A_2, B_2), $(A_3, B_3) \in \mathcal{T}$ then $A_1 \cup A_2 \cup A_3 \neq G$; and
(T3) if $(A, B) \in \mathcal{T}$ then $V(A) \neq V(G)$.

These are called the first, second and third tangle axioms. The *tangle number* of G, denoted by $\theta(G)$, is the maximum order of tangles of G. Figure 3 shows the tangle of order 3 for the graph in Figure 1.

The relationship between tangle number and branchwidth is as follows:

Theorem 1 (Robertson and Seymour [106]). *For any hypergraph G, $max(\beta(G), \eta(G)) = \theta(G)$ unless $\eta(G) = 0$ and $V(G) \neq \emptyset$.*

Separation of order 0
$(\emptyset,\, G)$
Separations of order 1
$(v,\, G)\ \forall\ v\ \in\ V(G)$
Separations of order 2
$(\{v,\ w\},\, G)\ \forall\ v,\ w\ \in\ V(G)$
$(G[e],\, G[E(G)\setminus e])\ \forall\ e\ \in\ E(G)$
$(G[\{ac, ad\}],\, G[E(G)\setminus\{ac, ad\}])$
$(G[\{dg, gh\}],\, G[E(G)\setminus\{dg, gh\}])$
$(G[\{fk, jk\}],\, G[E(G)\setminus\{fk, jk\}])$
$(G[\{gh, hi\}],\, G[E(G)\setminus\{gh, hi\}])$
$(G[\{jl, lm\}],\, G[E(G)\setminus\{jl, lm\}])$
$(G[\{ln, no\}],\, G[E(G)\setminus\{ln, no\}])$
$(G[\{mq, pq\}],\, G[E(G)\setminus\{mq, pq\}])$
$(G[\{dg, gh, hi\}],\, G[E(G)\setminus\{dg, gh, hi\}])$
$(G[\{jl, ln, no\}],\, G[E(G)\setminus\{jl, ln, no\}])$
$(G[\{mq, op, pq\}],\, G[E(G)\setminus\{mq, op, pq\}])$
$(G[\{dg, di, gh, hi\}],\, G[E(G)\setminus\{dg, di, gh, hi\}])$
$(G[\{mo, mq, op, pq\}],\, G[E(G)\setminus\{mo, mq, op, pq\}])$
$(G[\{mo, mq, no, op, pq\}],\, G[E(G)\setminus\{mo, mq, no, op, pq\}])$
$(G[\{ln, mo, mq, no, op, pq\}],\, G[E(G)\setminus\{ln, mo, mq, no, op, pq\}])$
$(G[\{jl, ln, mo, mq, no, op, pq\}],\, G[E(G)\setminus\{jl, ln, mo, mq, no, op, pq\}])$
$(G[\{jl, jm, ln, mo, mq, no, op, pq\}],\, G[E(G)\setminus\{jl, jm, ln, mo, mq, no, op, pq\}])$
$(G[\{em, jl, jm, ln, mo, mq, no, op, pq\}],\, G[E(G)\setminus\{em, jl, jm, ln, mo, mq, no, op, pq\}])$
$(G[\{em, je, jl, jm, ln, mo, mq, no, op, pq\}],\, G[E(G)\setminus\{em, je, jl, jm, ln, mo, mq, no, op, pq\}])$
FIGURE 3. Tangle of order 3 for the graph G of Figure 1.

2.4. Tree decompositions

The notions of a tree decomposition and treewidth were introduced by Robertson and Seymour [104] and measure the *tree-likeness* of a graph. A tree decomposition of a graph $G = (V, E)$ is a pair $(\{X_i, i \in I\}, T = (I, F))$ with $X_i \subseteq V$, $i \in I$ and $T = (I, F)$ a tree, such that:

(TD1) $\bigcup_{i\in I} X_i = V$;
(TD2) for all $vw \in E$, there is an $i \in I$ with $v, w \in X_i$; and
(TD3) for all $v \in V$, $\{i \in I : v \in X_i\}$ forms a connected subtree 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* $\tau(G)$ of G is the minimum width over all tree decompositions of G. The '-1' in the definition of the width of a tree decomposition has cosmetic reasons only: in this way the treewidth of a connected graph equals one if and only if it is a tree (or forest if the graph is unconnected). Figure 4 shows a tree decomposition of the graph in Figure 1. Property *(TD3)* is sometimes called the interpolation property. Furthermore, note that *(TD1)* is obsolete if G has no isolated vertices.

The notions of branchwidth and treewidth are closely related to each other, as expressed in the following theorem.

Theorem 2 (Robertson and Seymour [106]). *Let $G = (V, E)$ be a graph with $E \neq \emptyset$. Then* $\max(\beta(G), 2) \leq \tau(G) + 1 \leq \max(\lfloor \frac{3}{2}\beta(G) \rfloor, 2)$.

Several equivalent notions for treewidth have been studied over time, such as partial k trees, dimension, and k-decomposability. A graph has treewidth at most k if and only if it is a partial k-tree if and only if the dimension of G is at most k if and only if G is k-decomposable. See Bodlaender [27] for further details.

If T is restricted to be a path, we refer to $(\{X_i, i \in I\}, T = (I, F))$ as a *path decomposition* and the best width as the *pathwidth* of G. The pathwidth of a tree can be arbitrarily large.

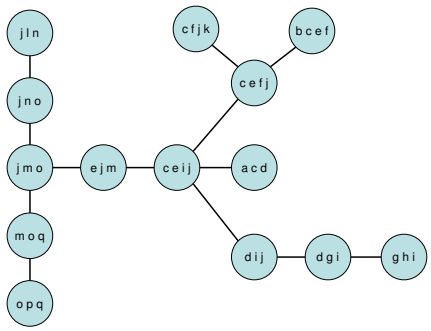


FIGURE 4. Tree decomposition with width 3 for the graph of Figure 1.

One is referred to survey papers by Bodlaender [27] and Bienstock [19] for a more thorough exposition of pathwidth.

2.5. Monadic Second Order Logic

Every class of graphs, when interpreted as a logical object, can be defined by a logical statement. Furthermore, graph properties such as the existence of disjoint paths can be stated as a logical condition. The following theorem from Courcelle [49] showed that graph problems that can be stated in monadic second-order logic can be solved in polynomial time for graphs with bounded branchwidth or treewidth.

Theorem 3 (Courcelle [49]). *Let ϕ be a monadic second-order logic problem and let $\mathcal{K}$ be a class of graphs with branchwidth or treewidth bounded above by k. For a graph G in $\mathcal{K}$, it can be determined in polynomial time if G satisfies ϕ. If G is given with a branch decomposition or tree decompositions with width less than or equal to k, then a linear time algorithm exists.*

Monadic second-order logic (MSOL) is an extension of first-order logic that includes vertex and edge sets and quantification (universal and existential) over these sets. In the context of graph theory, first-order logic consists of the logical connectives, $\wedge$, $\vee$, $\neg$, $\Rightarrow$ and $=$ (traditionally interpreted as 'and', 'or', 'not', 'implies', and 'equals', respectively), variables (e.g., x_1; y_1; a; b; z), universal ($\forall$) and existential ($\exists$) quantifiers, and a symbol defined to represent the existence of edges between vertices. An example of a first-order formula ϕ is

$$\forall x \; \exists y \; [(edg(x, y) \vee edg(y, x)) \wedge \neg(x = y)]$$

which is read "for every x there exists y such that the graph in question contains an edge from x to y or the graph in question contains an edge from y to x and it is not true that x equals y" or simply the graph contains no isolated vertices. First-order logic is limited in its expressiveness: it can only be used for local properties. Monadic second-order logic's set variables and set quantification allow for larger, more complex expressions.

For problems that involve optimization of a numerical evaluation over the sets of vertices or edges that are introduced in MSOL, we turn to the language of extended monadic second-order logic (EMSOL) and the extended monadic second-order extremum problem presented by Arnborg et al. [13]. These problems are also solvable in polynomial or linear time for graphs with bounded branchwidth or treewidth.

3. Graph Minors Theorem

In the 1930's, Kuratowski [90] proved that a graph G is planar if and only if G does not contain a subdivision of K_5, the complete graph on five vertices, or $K_{3,3}$, the complete bipartite graph with three vertices on each side of the bipartition. Later, Wagner [124] proved that a graph G is planar if and only if it does not contain K_5 or $K_{3,3}$ as a minor of G.

Let $\mathcal{F}$ be a class of graphs. $\mathcal{F}$ is *minor closed* if all the minors of any member of $\mathcal{F}$ also belong to $\mathcal{F}$. Given a minor closed class of graphs $\mathcal{F}$, the *obstruction set* of $\mathcal{F}$ is the set of minor minimal graphs that are not elements of $\mathcal{F}$ (i.e. graphs that do not belong to $\mathcal{F}$ but all of their proper minors belong to $\mathcal{F}$). Clearly, any class of graphs embeddable on a given surface is a minor closed class.

Recall, the obstruction set for the class of planar graphs was found to be K_5 and $K_{3,3}$. But what if we had surfaces other than the sphere? In 1979, Glover et al. [63] exhibited a list of 103 graphs in the obstruction set of projective-planar graphs, and then in 1980 Archdeacon [10] proved that this list is complete. In 1989, Archdeacon and Huneke [11] proved that the obstruction set for any non-orientable surface is finite. Finally as a corollary of the Graph Minors Theorem (GMT), formerly known as Wagner's conjecture, Robertson and Seymour [103] proved that every minor closed class of graphs has a finite obstruction set. The notions of branch decompositions, tangles, and tree decompositions were beneficial to the proof of GMT.

Given some surface Σ, an *antichain* for Σ is a list of minor minimal graphs which cannot be embedded in Σ. This means that no member of an antichain is isomorphic to a minor of another. In the early 1960's, Wagner conjectured that *every surface has a finite antichain.* The proof of this conjecture would imply that the obstruction set for any minor closed class is finite [103].

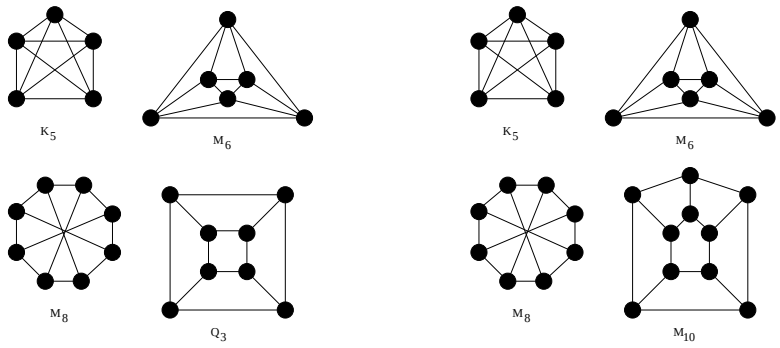
Theorem 4 (Graph Minors Theorem; Robertson and Seymour [109]). *For every infinite sequence of graphs G_1, G_2,..., there exists i, j with $i < j$ such that G_i is isomorphic to a minor of G_j.*

A class with a reflexive and transitive relation is a *quasi-order.* For example, the relation "H is isomorphic to a subgraph of G" defines a quasi-order on the class of all graphs. A quasi-order, denoted by $(Q, \leq)$, is *well-quasi-ordered* if for every countable sequence q_1, q_2,... of members of Q there exist $1 \leq i < j$ such that $q_i \leq q_j$. Wagner's conjecture is equivalent to stating that the "minor" quasi-order, "H is isomorphic to a minor of G", is well-quasi-ordered.

One quasi-order that is not well-quasi-ordered is the "subgraph" quasi-order stated earlier. This is true because a countable set of circuit graphs, one of each size, is an infinite antichain [103]. A graph G *topologically contains* a graph H if G has a subgraph which is isomorphic to a subdivision of H. Topological containment is not a well-quasi-ordering either. The set of graphs formed by taking a circuit graph of each size and replacing each edge by two parallel edges is an infinite antichain [103]; however, Kruskal proved that the class of all trees is well-quasi-ordered under topological containment, one of two famous conjectures of Vázsonyi [89]. Robertson and Seymour [105] used this theorem to prove:

Theorem 5. *For any integer k the class of all graphs with treewidth $\leq k$ is well-quasi-ordered by minors.*

This theorem and many more like it proved beneficial to prove Theorem 3. The other conjecture of Vázsonyi was that the class of all graphs with maximum degree at most three is a well-quasi-ordering under topological containment [103].



(a) Obstruction set for graphs with $\beta(G) \leq 3$

(b) Obstruction set for graphs with $\tau(G) \leq 3$

FIGURE 5. Obstruction set for graphs with branchwidth and treewidth at most 3.

Another quasi-order is *immersion*. A pair of adjacent edges ab and bc is *lifted* if ab and bc are replaced by the edge ac. A graph H is *immersed* in a graph G if H can be obtained from a subgraph of G by lifting pairs of edges. Nash-Williams [96] conjectured that the "immersion" quasi-order is well-quasi-ordered. This would imply both of Vázsonyi's conjectures.

Tree decompositions were introduced in the proof of GMT because large order tangles have a tree-like structure in their association with small order tangles. Branch decompositions were introduced in Robertson and Seymour [106] where they studied the relationship between tree decompositions and tangles.

By a result of Robertson and Seymour [106], for a given integer k, the class of graphs with branchwidth at most k is a minor closed class, meaning that if G is a graph that has branchwidth at most k, any minor of G has branchwidth at most k. Hence, the class of graphs with branchwidth at most k and the class of graph with treewidth at most k have finite obstruction sets. The only completely known obstruction sets are for graphs with branchwidth or treewidth 2 and 3. The obstruction set for both graphs with branchwidth at most 2 and graphs with treewidth at most 2 is K_4. Bodlaender and Thilikos [38] proved that a graph has branchwidth at most 3 if and only if it does not have K_5, Q_3, M_6, and M_8 as minor, see Figure 5(a). Similarly, Arnborg et al. [15] and Satyanarayana and Tung [113] independently proved that the obstruction set for graphs with treewidth at most 3 is K_5, M_6, M_8, and M_{10}, see Figure 5(b). The complete obstruction sets are not known for larger values of either treewidth or branchwidth; however, Hicks [71] did prove that the Petersen graph is a member of the obstruction set for graphs with branchwidth at most 4. Note, the size of an obstruction for graphs with branchwidth or treewidth at most k probably grows exponentially with k; it may prove extremely difficult to characterize these obstructions sets for large k. We refer the reader to the work of Bienstock and Langston [20] for a more thorough survey of GMT.

4. Branch Decompositions and Algorithms

This section details constructing branch decompositions with width as small as possible and the use of branch decomposition based algorithms for solving discrete optimization problems.

4.1. Constructing Branch Decompositions

Since the complexity of branch decomposition based algorithms is typically exponential in the given fixed width of an input branch decomposition (cf. Section 4.2), finding branch decompositions whose associated width is as small as possible is vital to the performance of a branch decomposition based algorithm.

4.1.1. Construction in Theory By a result of Seymour and Thomas [116], computing the branchwidth and finding an optimal branch decomposition of a general graph is $\mathcal{NP}$-hard. However, there is a polynomial time algorithm in Robertson and Seymour [108] to approximate the branchwidth of a graph within a factor of 3. For example, the algorithm decides if a graph has branchwidth at least 10 or finds a branch decomposition with width at most 30. This algorithm and its improvements by Bodlaender [25], Bodlaender and Kloks [32], and Reed [101] are only of theoretical importance. Bodlaender and Thilikos [38] did give an algorithm to compute the optimal branch decomposition for any chordal graph with maximum clique size at most 4 but the algorithm has been only shown practical for a particular type of 3-tree. Bodlaender and Thilikos [37] also developed a tree decomposition based linear time algorithm for finding an optimal branch decomposition but it appears to be computationally impractical as well.

In terms of planar graphs, Fomin and Thilikos [56] proved that the branchwidth of any planar graph is asymptotically bounded by the square root of the graph's number of nodes $(2.122\sqrt{n})$ based upon work on planar separators by Alon et al. [7]. This work also offered a complexity bound for the minimum dominating set problem on planar graphs smaller than any known complexity bound for the problem, including the work of Alber and Niedermeier using tree decompositions [4]. The authors have produced similar results for $\mathcal{NP}$-hard problems like the independent set problem, the longest cycle problem and the bisection problem for planar graphs [57]. Kloks, Kratochvil and Müller [80, 81] gave a polynomial time algorithm to compute the branchwidth of interval graphs but this algorithm has not been shown to be practical.

4.1.2. Construction in Practice

Tree Building To construct a branch decomposition, start with a partial branch decomposition and refine this decomposition until the tree is ternary. The underlying structure used in constructing a branch decomposition is the separation, see Section 2.2. Without loss of generality, we only use separations (G_1, G_2) such that $E(G_1)$ and $E(G_2)$ are nonempty. Separations are vital to the construction of a branch decomposition because finding separations will help refine partial branch decompositions into branch decompositions. In this section, we may assume that the input graph G is biconnected since one can derive an optimal branch decomposition for a disconnected graph G from the optimal branch decompositions of G's connected components; one can also derive an optimal branch decomposition for a connected graph H from the optimal branch decompositions of the biconnected components of H.

Given a partial branch decomposition, the studied refinements are *one splits* and *two splits*. Let G represent our input graph and let (T_1, ν) be a partial branch decomposition of G. Let v be a non-leaf node of T_1 with degree greater than three and denote D_v as the set of edges incident with v. For a set $S \subseteq V(G)$, let $he(S)$ denote a hyperedge where the ends of the hyperedge are the elements in S. Define H^v as the hypergraph constructed from the union of hyperedges $he(mid(e))$ for all $e \in D_v$. So if T_1 is a star, then H^v would correspond to G since G is assumed to be biconnected. Let (X, Y) be a separation of H^v. Create the tree T_2 by replacing v with nodes x and y and the edge xy where x would be incident with the edges that correspond to $E(X)$ and y would be incident with the edges that correspond to $E(Y)$.

This procedure is called a *one split*. The middle set for the edge xy would be $V(X) \cap V(Y)$; Figure 6 offers an illustration of a one split.

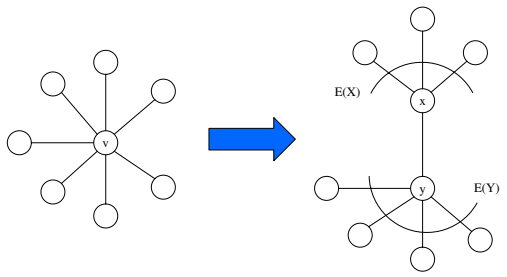


FIGURE 6. A One Split

Let G, (T_1, ν), and v be defined as in the previous paragraph. Let e be an edge incident with v and let $he(e)$ denote the hyperedge of H^v that corresponds to e. Let (X, Y) be a separation of the hypergraph $H^v \setminus \{he(e)\}$. Without loss of generality, assume that the cardinality of $E(X)$ is at most the cardinality of $E(Y)$. If the cardinality of $E(X)$ is greater than one, create T_2 by adding new nodes x and y and edges vx and vy to T_1 with x incident with the edges corresponding to $E(X)$ and y incident with the edges corresponding to $E(Y)$. Otherwise, create T_2 by inserting a new node y and edge vy with y incident with the edges corresponding to $E(Y)$. The middle sets of the new edges in either case would be:

$$mid(vx) = (V(Y) \cup mid(e)) \cap V(X) \tag{1}$$

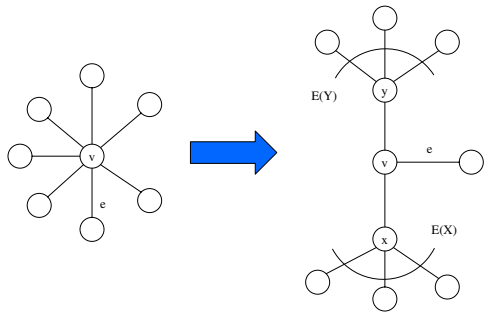
$$mid(vy) = (V(X) \cup mid(e)) \cap V(Y) \tag{2}$$

This procedure is called a *two split*. Figure 7 offers the two examples of a two split. Notice that a two split when $|E(X)| = 1$ is equivalent to an one split with $|E(X)| = 2$; otherwise, the two procedures do not yield the same results.

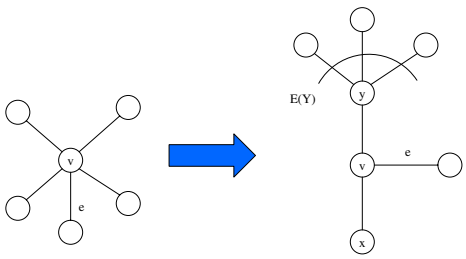
In order to build a branch decomposition, start with a partial branch decomposition whose tree is a star and conduct a sequence of one and two splits to achieve a branch decomposition. The tree-building aspect of using only one splits is equivalent to the tree-building aspect developed by Cook and Seymour [47, 48] and the tree-building aspect of using only two splits is equivalent to the tree-building aspect developed by Robertson and Seymour [108].

A partial branch decomposition (T, ν) of a graph G is called *extendible* given that $\beta(H^v) \leq \beta(G)$ for every non-leaf node $v \in V(T)$. This follows from the fact that if every H^v had branchwidth at most some number k, then one could use the optimal branch decompositions of the hypergraphs to build a branch decomposition of G whose width is at most k. Even though a partial branch decomposition whose tree is a star is extendible, it is $\mathcal{NP}$-hard to check whether an arbitrary partial branch decomposition is extendible for general graphs. In contrast, this is not the case for planar graphs, discussed later.

A separation is called *greedy* or *safe* [47, 48] if the next partial branch decomposition created by the use of the separation in conjunction with a one or two split is extendible if the previous



(a) $|E(X)|>1$



(b) $|E(X)|=1$

FIGURE 7. Two Splits

partial branch decomposition was extendible. In particular, Cook and Seymour [47, 48] describe three types of safe separations; the first and more general type is called a *push*. For a hypergraph H and F, a subset of nodes or edges, let $H[F]$ denote the subhypergraph of H induced by F. The push separation is described in the following lemma.

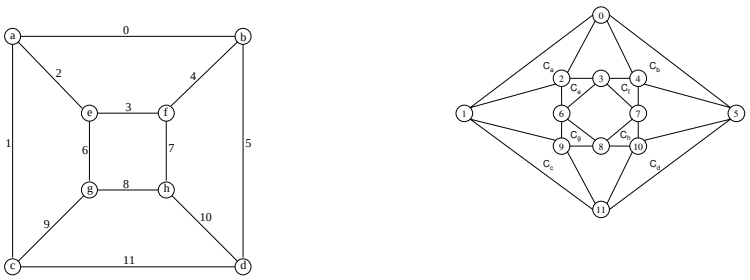
Lemma 1 (Cook and Seymour [47, 48]). *Let G be a graph with a partial branch decomposition (T,ν). Let $v \in V(T)$ have degree greater than three and let $D_v \subseteq E(T)$ be the set of edges incident with v. Also, let H^v be the corresponding hypergraph for v. Suppose there exist $e_1, e_2 \in E(T)$ incident with v such that $|(mid(e_1) \cup mid(e_2)) \cap \bigcup\{mid(f) : f \in D_v \setminus \{e_1, e_2\}\}| \leq \max\{|mid(e_1)|, |mid(e_2)|\}$. Let $h_{e_1}, h_{e_2} \in E(H^v)$ be the corresponding hyperedges for e_1 and e_2, respectively. Then the resulting partial branch decomposition after taking*

a one split using the separation $(H^v[\{h_{e_1}, h_{e_2}\}], H^v[E(H^v) \setminus \{h_{e_1}, h_{e_2}\}])$ *is extendible if* T *was extendible.*

The other types of safe separations utilize 2-separations and 3-separations that satisfy some simple conditions. First, given a partial branch decomposition of a biconnected graph, if a separation (X, Y) is found such that $|V(X) \cap V(Y)| = 2$ then (X, Y) is safe. This is due to the fact that any 2-separation is *titanic* in a biconnected graph [106]. All 3-separations (X, Y) are safe unless $V(X) \cap V(Y)$ corresponds to an independent set in G and either $V(X) \setminus V(Y)$ or $V(Y) \setminus V(X)$ has cardinality one; this is another result derived by Robertson and Seymour [106].

Planar Graphs For planar (hyper)graphs, there exists a polynomial time algorithm called the *ratcatcher method* [116] to compute the branchwidth. We briefly comment on the background behind the method and related results for computing the branchwidth of planar graphs.

Let G be a graph with node set $V(G)$ and edge set $E(G)$. Let T be a tree having $|V(G)|$ leaves in which every non-leaf node has degree three. Let μ be a bijection between the nodes of G and the leaves of T. The pair (T, μ) is called a *carving decomposition* of G. Notice that removing an edge e of T partitions the nodes of G into two subsets A_e and B_e. The *cut set* of e is the set of edges that are incident with nodes in both A_e and B_e (also denoted $\delta(A_e)$ or $\delta(B_e)$). The *width* of a carving decomposition (T, μ) is the maximum cardinality of the cut sets for all edges in T. The *carvingwidth* for G, $\kappa(G)$, is the minimum width over all carving decompositions of G. A carving decomposition is also known as a *minimum-congestion routing trees* and one is referred to Alvarez et al. [8] for a link between carvingwidth and network design. The ratcatcher method is really an algorithm to compute the carvingwidth for planar graphs. In order to show the relation between carvingwidth and branchwidth we need another definition.



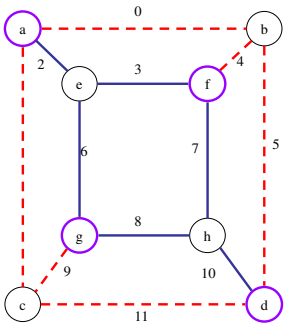
(a) Q_3 (b) $M(Q_3)$

FIGURE 8. Q_3 and its Medial Graph

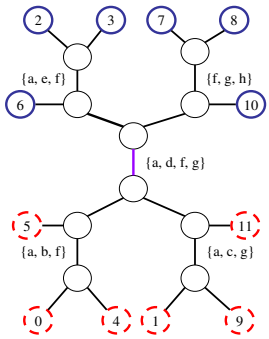
Let G be a planar (hyper)graph and let G also denote a particular planar embedding of the graph on the sphere. For every node v of G, the edges incident with v can be ordered in a clockwise or counter-clockwise order. This ordering of edges incident with v is the *cyclic order* of v. Let $M(G)$ be a graph with the vertex set $E(G)$. For a node $v \in V(G)$, define the cycle C_v in $M(G)$ as the cycle through the nodes of $M(G)$ that correspond to the edges incident with v according to v's cyclic order in G; the edges of $M(G)$ is the union of cycles C_v for all $v \in V(G)$. $M(G)$ is called a *medial graph* of G, see Figure 8. Notice that every connected

planar hypergraph G with $E(G) \neq \emptyset$ has a medial graph and every medial graph is planar. In addition, notice that there is a bijection between the regions of $M(G)$ and the nodes and regions of G. Hence, one can derive, using the theory of Robertson and Seymour [107], that if a planar graph and its dual are both loopless then they have the same branchwidth, see Hicks [71]. Figure 9 illustrates this result by presenting one branch decomposition for both Q_3 and M_6. For the relationship between branchwidth and carvingwidth, Seymour and Thomas [116] proved:

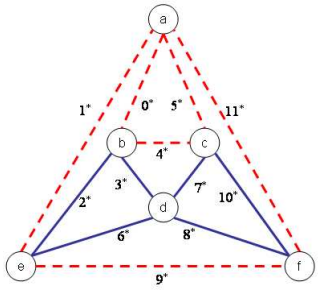
Theorem 6 (Seymour and Thomas [116]). *Let G be a connected planar graph with $|E(G)| \geq 2$, and let $M(G)$ be the medial graph of G. Then the branchwidth of G is half the carvingwidth of $M(G)$.*



(a) Q_3



(b) (T, ν)



(c) dual of Q_3: M_6

FIGURE 9. Q_3 and M_6 have branchwidth four

So, computing the carvingwidth of $M(G)$ gives us the branchwidth of G. Also, having a carving decomposition of $M(G)$, (T, μ), gives us a branch decomposition of G, (T, ν), such that the width of (T, ν) is exactly half the width of (T, μ). The ratcatcher method actually computes the carvingwidth of planar graphs. In addition, the ratcatcher method does not search for low cut sets in the medial graph but for objects that prohibit the existence of low cut sets. These objects are called *antipodalities*, see Seymour and Thomas [116] for more details. The ratcatcher method has time-complexity $O(n^2)$ but requires a considerable amount of memory for practical purposes. A slight variation which is more memory-friendly was offered by Hicks [75] at the expense of the time-complexity going up to $O(n^3)$.

The original algorithm developed by Seymour and Thomas [116] to construct optimal branch decompositions had complexity $O(n^4)$ and used the ratcatcher method to find extendible separations. A practical improvement upon this algorithm using a more thorough divide-and-conquer approach was offered by Hicks [76]. Recently, Gu and Tamaki [66] have found a $O(n^3)$ time algorithm utilizing the ratcatcher method by bounding the number of calls to the ratcatcher method by $O(n)$. In addition, Tamaki [119] offered a linear time heuristic for constructing branch decompositions of planar graphs; the heuristic could find a branch decomposition of a 2,000 node planar graph in about 117 milliseconds on a 900MHz Ultra SPARC-III. The heuristic uses the medial-axis tree of $M(G)$ derived from a breadth-first search tree of $M(G)^*$. Thus, the computed width is bounded below by the height of breadth-first search tree; the difference between this parameter (bounded below by the radius of the dual of the medial graph) and the branchwidth could be huge using a similar construction as in Figure 10. Figure 10 raises an interesting question: What characterizes a planar graph G such that G has the property that $\beta(G)$ is equal to the radius of $M(G)^*$?

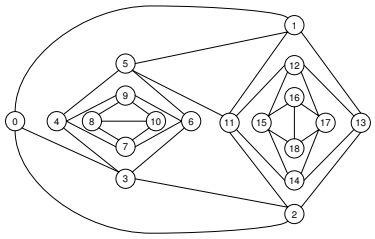


FIGURE 10. Tamaki's heuristic [119] gives a width bounded below by 6; the branchwidth is 3.

General Graphs For general graphs, most work has been done utilizing heuristics to actually construct branch decompositions. Cook and Seymour [47, 48] gave a heuristic algorithm to produce branch decompositions. Their heuristic is based on spectral graph theory and the work of Alon [6]. Moreover, Hicks [72] also found another branchwidth heuristic that was comparable to the algorithm of Cook and Seymour. This heuristic finds separations by minimal vertex separators between diameter pairs.

In addition, Hicks [74] has developed a branch decomposition based algorithm for constructing an optimal branch decomposition based on the notion of a *tangle basis*. For an integer k and hypergraph G, a *tangle basis* $\mathcal{B}$ of order k is a set of separations of G with order $< k$ such that:

(B1) $(G[e], G[E(G) \setminus e]) \in \mathcal{B}$, $\forall\ e \in E(G)$ if $\eta(e) < k$;
(B2) $(C, D) \in \mathcal{B}$ and $\nexists\ e \in E(G)$ such that $G[e] = C$ if and only if $\exists\ (A_1, B_1),\ (A_2, B_2) \in \mathcal{B}$ such that $A_1 \cup A_2 = C$ and $B_1 \cap B_2 = D$; and

(B3) $\mathcal{B}$ obeys the tangle axioms T2 and T3.

We will refer to B1, B2, and B3 as the tangle basis axioms. A tangle basis $\mathcal{B}$ in G of order k is *connected* if every separation (A, B) of $\mathcal{B}$ has A connected. Define the *connected tangle basis number* of G, denoted by $\theta'(G)$, as the maximum order of any connected tangle basis of G. The relationship between connected tangle bases and tangles is given in the following theorem:

Theorem 7 (Hicks [74]). *If hypergraph G is connected such that $\beta(G) \geq \eta(G)$, then $\beta(G) = \theta'(G)$.*

The relationship shown in this theorem was used by Hicks [74] to construct a branch decomposition based algorithm that tests whether a given connected graph has branchwidth at most $k-1$ (where $k \geq 3$) to be used in a practical setting to compute the branchwidth and the optimal branch decomposition of a graph.

In the same vein of connected tangle bases, Fomin et al. [55] have developed the notion of connected branch decompositions for 2-edge-connected graphs. Given a graph G and a branch decomposition (T, ν), (T, ν) is called *connected* if for every edge $e \in E(T)$, the two edge induced subgraphs of G corresponding to the two components of $T \setminus \{e\}$ are both connected. Fomin et al. [55] showed that given a 2-edge-connected graph G and a branch decomposition of G of width k there also exists a connected branch decomposition of G of width $k' \leq k$. The authors used this result to approximate in polynomial time the connected search strategy for a graph.

4.2. Branch Decomposition Based Algorithms

The blueprint of a branch decomposition based algorithm typically consists of two steps: transforming the tree of the branch decomposition of a graph G into a rooted binary tree; and visiting the nodes of the tree in post-depth-first search order in order to generate a solution for the problem of interest. The tree T is transformed into a rooted binary tree by selecting an edge ab and replacing ab by a node r, the *root*, and edges ar and rb. Since each leaf of T corresponds to a particular edge of G, then each rooted subtree of T corresponds to a particular subgraph of G. Also, every node v of T that is not a leaf node and not the root of the tree is adjacent to three nodes: a parent node p; a left child lc; and a right child rc. For each tree node v, let G_v denote the subgraph of G induced by the edges corresponding to the leaves of the tree that are descendants of v. In addition, let C_v denote the middle set of the parent edge pv. The algorithm then visits the tree nodes in post-depth-first order and builds a set of partial solutions for each tree node. By visiting the tree in post-depth-first order, all descendants of tree node v are visited before v is visited.

To illustrate further, some details for the Steiner tree problem in graphs are as follows. A partial solution for a tree node v would be a forest F of G_v such that every connected component D of F has the property that $V(D) \cap C_v \neq \emptyset$ unless F is a Steiner tree of G. For the case that the tree node v is a leaf, there is one partial solution: $\nu(v)$ is an edge of the desired Steiner tree. For tree node v that is not a leaf, the set of partial solutions of v is contained in the union of the partial solutions of v's children, lc and rc, and the merger of partial solutions from both sets. Thus, every partial solution of lc is merged with every partial solution of rc on the node set $C_{lc} \cap C_{rc}$. Figure 11 illustrates this procedure where $C_{lc} = \{a, b, c, d, e\}$, $C_{rc} = \{c, d, e, f, g\}$, $C_v = \{a, b, c, f, g\}$, and the terminals are in black. The number of partial solutions for a tree node v can grow very rapidly because the number of partial solutions for any tree node v is exponential in the cardinality of C_v. Thus, the time complexity of a branch decomposition based algorithm is exponential in the width of

the input branch decomposition, which explains the importance of having low width. In contrast, some partial solutions can be pruned away if they can never be transformed into a feasible Steiner tree. For example, if a partial solution containing a cycle is obtained by merging two partial solutions at a tree node, then that partial solution can be pruned away. Another example is if a partial solution of a tree node v had a connected component D that did not intersect with C_v and all the terminals were not contained in D, then that partial solution can also be pruned.

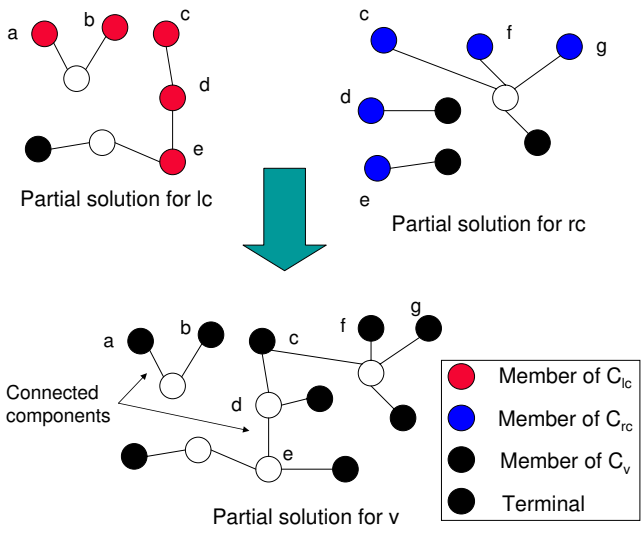


FIGURE 11. Joining Partial Solutions for lc and rc to Obtain a Partial Solution for v, for the Steiner Tree Problem

A branch decomposition based algorithm was offered by Cook and Seymour [47] for the ring-routing-problem, which arises in the design of reliable cost effective synchronous optical networks (SONET networks) and was incorporated into commercial software for Telcordia Technologies (formerly Bellcore). Cook and Seymour [48] also used a branch decomposition based algorithm on a sparse graph generated by fractional solutions of linear programming (LP) relaxations of a TSP test instance to generate an optimal solution for the sparse graph and an upper bound for the test instance. This technique produced the best known solutions for the 12 unsolved problems in TSPLIB95, a library of standard test instances for the TSP [102]. Hicks has also developed computationally efficient branch decomposition based algorithms for minor containment [73] and optimal branch decompositions [74]. One is also referred to the work of Christian [43] for other practical algorithms. Some examples of branch decomposition based algorithms proposed in theory are the notable work of Fomin and Thilikos [56] and the work of Alekhnovich and Razborov [5] who used the branch-width of hypergraphs to design a branch decomposition based algorithm in theory to solve satisfiability problems. Despite the aforementioned examples of research in branch decompositions offering glimpses of potentially rewarding research in the field, overall research in the area has been relatively ignored and vastly unexplored compared to research in tree decompositions.

5. Tree Decompositions and Algorithms

Besides branch decompositions, tree decompositions have also been used to solve discrete optimization problems. In this section, we consecutively discuss the construction of a tree decomposition with width as small as possible, lower bounds on the treewidth of a graph, and finally algorithms for optimization problems that are based on tree decompositions.

5.1. Constructing Tree Decompositions

The construction of an arbitrary tree decomposition is trivial: just take a single node tree and assign to this node a bag containing all vertices in the graph. This pair of a tree and a bag satisfies all three conditions of a tree decomposition and thus qualifies as a tree decomposition. Despite its formal status, it does not deserve the name tree *decomposition* from a practical perspective: the tree decomposition does not decompose anything and is not useful for solving combinatorial problems.

As with branch decompositions, the worth of a tree decomposition depends on its width. The smaller the size of the largest bag, the more promising a tree decomposition is for solving combinatorial optimization problems. Therefore, the construction of a tree decomposition with a small width is desirable. We first discuss the theoretical results. Afterwards, we review how tree decompositions are constructed in practice.

5.1.1. Construction in Theory We are confronted with a difficult problem related to treewidth: Computing the treewidth of a graph is $\mathcal{NP}$-hard. Many theoretical assessments have been made concerning the difficulty of computing treewidth. The formal decision problem of treewidth asks whether there exists a tree decomposition with width at most k for some integer $k > 0$. If k is part of the input, $\mathcal{NP}$-completeness was proved by Arnborg et al. [12]. If k may be considered as a constant, not part of the input, the best algorithm has been given by Bodlaender [26] and checks in linear time whether or not a tree decomposition with width at most k exists. The $O(n)$ notation for this algorithm however hides a huge constant coefficient that obstructs its practical computational value. An experimental evaluation by Röhrig [110] revealed that the algorithm is computationally intractable, even for k as small as four.

Graphs with treewidth at most 4 can be characterized either directly or indirectly. As already pointed out in Section 2, $\tau(G) = 1$ if and only if G is a forest. A graph G has $\tau(G) \leq 2$ if and only if its biconnected components are series-parallel graphs [39]. Arnborg and Proskurowski [14] gave six reduction rules that reduce G to the empty graph if and only if $\tau(G) \leq 3$. Sanders [112] provided a linear time algorithm for testing $\tau(G) \leq 4$.

Besides forests and series-parallel graphs, the complexity of treewidth for some special classes of graphs are known (by presenting either a polynomial time algorithm or an $\mathcal{NP}$-completeness proof). We refer the interested reader to two surveys on the topic by Bodlaender [24, 29]. Most remarkable in this context is that, so far, the complexity of treewidth for planar graphs is unknown, whereas for branchwidth a polynomial time algorithm exists, see Section 4. Lapoire [91] and Bouchitté et al. [41] proved that the treewidth of a planar graph and of its geometric dual differ by at most one.

As it is $\mathcal{NP}$-complete to decide whether the treewidth of a graph is at most k, a natural way to proceed is to consider polynomial time approximation algorithms for the problem. Given a graph G with $\tau(G) = k$, the best algorithms are given by Bouchitté et al. [40] and Amir [9], both providing a tree decomposition of width at most $O(k \log k)$ (i.e., a $O(\log k)$ approximation). So far, neither is a constant approximation algorithm known nor is it proven that no such algorithm exists.

If we insist on computing the treewidth exactly, unless $\mathcal{P} = \mathcal{NP}$, the only way to go is the development of an exponential time algorithm, see Woeginger [125] for a survey in this recent branch of algorithm theory. For treewidth, Arnborg et al. [12] gave an algorithm with running time $O(2^n poly(n))$, where $poly(n)$ is a polynomial in n. Fomin et al. [58] presented a $O(1.9601^n poly(n))$ algorithm. Whether these algorithms are of practical usefulness for computing treewidth is a topic of further research.

5.1.2. Construction in Practice Most results presented in the previous subsection are of theoretical interest only: the computational complexity hides huge constant coefficients that make the algorithms impractical for actually computing treewidth. So far, only the reduction rules for treewidth at most three have been proved to be of practical use in preprocessing the input graph. However, in all those cases where the treewidth is larger than three, we have to turn to heuristics without any performance guarantee. Many of the results reviewed here have been tested on graphs of different origin, see TreewidthLIB [122] for a compendium.

Preprocessing The reduction rules of Arnborg and Proskurowski [14] not only reduce graphs of treewidth at most three to the empty graph, but can also be used as a preprocessing technique to reduce the size of general graphs. In Bodlaender et al. [34], the rules have been adapted and extended such as to preprocess general graphs. Given an input graph G, a value low is maintained during the preprocessing such that $\max\{low, \tau(G')\} = \tau(G)$, where G' is the (partly) preprocessed graph. If at any point, no further preprocessing rules can be applied anymore, a tree decomposition of the preprocessed graph G' is computed (see below). Finally, given a tree decomposition for G', a tree decomposition for the input graph can be obtained by reversal of the preprocessing steps and adapting the tree decomposition appropriately. Computational experiments have shown that significant reductions in the graph size can be achieved by these rules.

The above mentioned preprocessing rules emphasize the removal of vertices from the graph. Another way to reduce the complexity of finding a good tree decomposition is the splitting of the input graph into smaller graphs for which we can construct a tree decomposition independently. In Bodlaender and Koster [33], so-called *safe separators* are introduced for this purpose. A separator S is a set of vertices whose removal disconnects a graph G. Let V^i, $i = 1, \ldots, p$ $(p \geq 2)$, induce the connected components of $G - S$. On each of the connected components $G[V^i]$, a graph G^i is defined as $G[V^i \cup S] \cup clique(S)$, where $clique(S)$ denotes a complete graph, or *clique*, on S. If $\tau(G) = \max_{i=1,\ldots,p} \tau(G^i)$, then S is called *safe* for treewidth. In particular, clique separators (i.e., S induces a clique) and almost clique separators (i.e., S contains a $|S| - 1$ clique) are safe. Experiments revealed that, roughly speaking, by applying a safe separator decomposition to a graph, it remains to construct a tree decomposition for the smaller graphs given by the decomposition.

Exact algorithms Although treewidth is $\mathcal{NP}$-hard in general, there have been a couple of attempts to tackle the problem by exact approaches. Shoikhet and Geiger [117] implemented a modified version of the $O(n^{k+2})$ algorithm by Arnborg et al. [12]. A branch and bound algorithm based on vertex-ordering has been proposed by Gogate and Dechter [64].

Upper bound heuristics The operations research toolbox for constructing solutions to combinatorial optimization problems has been opened but not yet fully explored for computing the treewidth of a graph. Most heuristics are of a constructive nature: according to some principle, we construct a tree decomposition from scratch. Improvement heuristics as well as metaheuristics are less frequently exploited.

At first sight, condition *(TD3)* does not simplify the construction of good tree decompositions from scratch. However, an alternative definition of treewidth by means of graph triangulations reveals the key to constructive heuristics. A *triangulated* or *chordal* graph is

a graph in which every cycle of length at least four has a chord. A *triangulation* of a graph $G = (V, E)$ is a chordal graph $H = (V, F)$ with $E \subseteq F$.

Lemma 2. *Let G be a graph, and let $\mathcal{H}$ be the set of all triangulations of G. Then, $\tau(G) = \min_{H \in \mathcal{H}} \omega(H) - 1$, where $\omega(H)$ is the size of the maximum clique in H.*

Thus, if G is triangulated, then $\tau(G) = \omega(G) - 1$, otherwise we have to find a triangulation of H with small maximum clique size. Several algorithms exist to check whether G is triangulated or to construct a triangulation of G. All are based on a special ordering of the vertices. A *perfect elimination scheme* of a graph $G = (V, E)$ is an ordering of the vertices $v_1, \ldots, v_n$ such that for all $v_i \in V$, $\delta_{G[v_i,\ldots,v_n]}(v_i)$ induce a clique.

Lemma 3 ([60, 65]). *A graph G is triangulated if and only if there exists a perfect elimination scheme.*

To check whether a graph is triangulated it is thus enough to construct a perfect elimination scheme or to prove that no such scheme exists. The *lexicographic breadth first search* (LEX) recognition algorithm by Rose et al. [111] constructs in $O(n + m)$ time a perfect elimination scheme if such a scheme exists. The *maximum cardinality search* (MCS) by Tarjan and Yannakakis [120] does the same (with the same complexity in theory, but is faster in practice). Both algorithms can be adapted to find a triangulation H if G is not triangulated itself. With help of Lemma 2 a tree decomposition can be constructed with width equal to the maximum clique size of H minus one. The triangulated graph given by both algorithms is not necessarily minimal in the sense that there may not exist a triangulation $H' = (V, F')$ with $E \subset F' \subset F$. As unnecessarily inserted edges can increase the maximum clique size, it is desirable to find a minimal triangulation. For both algorithms there exist variants that guarantee to find a minimal triangulation H' of G, known as LEX-M [111] and MCS-M [17], respectively. See [84] for some experimental results for LEX-P, MCS, and LEX-M. Recently, Heggernes et al. [69] proposed a new algorithm to find a minimal triangulation. Alternatively, we can add as a post-processing step to MCS and LEX-P an algorithm that turns a triangulation into a minimal triangulation [22, 51, 70]. Note that in case the input graph is chordal, the minimal triangulation is the graph itself, and the treewidth of the graph is computed exactly with all described algorithms.

The *minimal fill-in* problem is another problem that is studied in relation to triangulation of graphs. The minimum fill-in of a graph is the minimum number of edges to be added to a graph such that the resulting graph is chordal/triangulated. This problem is known to be $\mathcal{NP}$-hard [127], but it is not difficult to think of two heuristics. The first one is a greedy algorithm: select repeatedly the vertex for which the fill-in among its neighbors is minimized, turn its neighbors into a clique, and remove that vertex. This algorithm is called *Greedy Fill-In* (GFI) or simply the minimum fill-in algorithm in some articles. The second algorithm does the same except that it selects the vertex according to the minimum degree. See Bachoore and Bodlaender [16] and Clautiaux et al. [44, 45] for computational experiments and fine tuning of these algorithms.

Except for the algorithm that turns a triangulation into a minimal triangulation, all heuristics described so far are constructive. The algorithm described in Koster [83] can be viewed as an improvement heuristic, similar to the tree building idea for branchwidth. Given a tree decomposition, it tries to replace the largest bag(s) by smaller ones, preserving all conditions of a tree decomposition. If the algorithm starts with the trivial tree decomposition consisting of a single node, the algorithm can be viewed as a constructive algorithm; if it starts with a tree decomposition constructed by another method, it can be considered an improvement heuristic as well.

Metaheuristics have been applied to treewidth as well. Clautiaux et al. [45] experimented with a tabu search algorithm. For a problem closely related to treewidth, Kjærulff [79] applies simulated annealing, whereas Larrañaga et al. [92] use a genetic algorithm.

Branchwidth and treewidth As already pointed out in Section 2, the notions branchwidth and treewidth are closely related. Given a branch decomposition with width k, a tree decomposition with width at most $\lfloor \frac{3}{2}k \rfloor$ can be constructed in polynomial time: Let i be an internal node of the branch decomposition and let j_1, j_2, j_3 be its neighbors. Moreover, let U_{j_1}, U_{j_2}, $U_{j_3} \subseteq V$ be the vertex sets induced by edges corresponding to the leafs of the subtrees rooted at j_1, j_2, and j_3 respectively. Thus $mid(ij_1) := U_{j_1} \cap (U_{j_2} \cup U_{j_3})$, $mid(ij_2) := U_{j_2} \cap (U_{j_1} \cup U_{j_3})$, and $mid(ij_3) := U_{j_3} \cap (U_{j_1} \cup U_{j_2})$. Now, associate with node i the bag $X_i := mid(ij_1) \cup (ij_2) \cup mid(ij_3)$. Since the union contains $U_j \cap U_k$, $j, k \in \{j_1, j_2, j_3\}$, $j \neq k$, twice, the size of X_i is at most $\lfloor \frac{3}{2}k \rfloor$. It is left to the reader to verify that $(\{X_i, i \in I\}, T = (I, F))$ satisfies all conditions of a tree decomposition.

5.2. Treewidth Lower Bounds

The heuristics for practical use described above do not generally guarantee a tree decomposition with width close to optimal. To judge the quality of the heuristics, lower bounds on treewidth are of great value. Moreover, obtaining good lower bounds quickly is essential for the performance of branch and bound algorithms (see Gogate and Dechter [64]), and the height of a treewidth lower bound is a good indication for the computational complexity of tree decomposition based algorithms to solve combinatorial optimization problems.

In recent years, substantial progress on treewidth lower bounds has been achieved, both theoretically and practically. The probably widest known lower bound is given by the maximum clique size. This can be seen by Lemma 2: the maximum clique of G will be part of a clique in any triangulation of G.

Scheffler [114] proved that every graph of treewidth at most k contains a vertex of degree at most k. Stated differently, the minimum degree $\delta(G)$ is a lower bound on the treewidth of a graph. Typically this lower bound is of no real interest as the minimum degree can be arbitrarily small. Even if the preprocessing rules of the previous section have been applied before, only $\delta(G) \geq 3$ can be guaranteed.

Ramachandramurthi [99, 100] introduced the parameter

$$\gamma_R(G) = \min(n-1, \min_{v,w \in V, v \neq w, \{v,w\} \notin E} \max(d(v), d(w))$$

and proved that this is a lower bound on the treewidth of G. Note that $\gamma_R(G) = n-1$ if and only if G is a complete graph on n vertices. If G is not complete, then $\gamma_R(G)$ is determined by a pair $\{v, w\} \notin E$ with $\max(d(v), d(w))$ as small as possible. From its definition it is clear that $\gamma_R(G) \geq \delta_2(G) \geq \delta(G)$, where $\delta_2(G)$ is the second smallest degree appearing in G (note $\delta(G) = \delta_2(G)$ if the minimum degree vertex is not unique). So, we have

$$\delta(G) \leq \delta_2(G) \leq \gamma_R(G) \leq \tau(G)$$

and all these three lower bounds can be computed in polynomial time.

One of the heuristics for constructing a (good) tree decomposition is the Maximum Cardinality Search algorithm (MCS), see Section 5.1.2. Lucena [94] proved that with the same algorithm a lower bound on the treewidth can be obtained. The MCS visits the vertices of a graph in some order, such that at each step, an unvisited vertex that has the largest number of visited neighbors becomes visited (note that the algorithm can start with an arbitrary vertex). An MCS-ordering of a graph is an ordering of the vertices that can be generated

by the algorithm. The visited degree of a vertex v in an MCS-ordering is the number of neighbors of v that are before v in the ordering. The visited degree of an MCS-ordering ψ of G is the maximum visited degree over all vertices v in ψ and denoted by $mcslb_\psi(G)$.

Theorem 8 (Lucena [94]). *Let G be a graph and ψ an MCS-ordering. Then, $mcslb_\psi(G) \leq \tau(G)$.*

If we define the *maximum visited degree* $MCSLB(G)$ of G as the maximum visited degree over all MCS-orderings of graph G, then obviously $MCSLB(G) \leq \tau(G)$ as well. Bodlaender and Koster [33] proved that determining whether $MCSLB(G) \leq k$ for some $k \geq 7$ is $\mathcal{NP}$-complete and presented computational results by constructing MCS-orderings using tiebreakers for the decisions within the MCS algorithm.

It is easy to see that every lower bound for treewidth can be extended by taking the maximum of the lower bound over all subgraphs or minors: given an optimal tree decomposition for G and H a subgraph (minor) of G, then we can construct a tree decomposition with equal or better width for H by removing vertices from the bags that are not part of the subgraph (minor) and replacing contracted vertices by their new vertex.

In Koster et al. [84] the minimum degree lower bound has been combined with taking subgraphs. The maximum minimum degree over all subgraphs, denoted by $\delta D(G)$ is known as the *degeneracy* of a graph G and can be computed in polynomial time by repeatedly removing a vertex of minimum degree and recording the maximum encountered. Szekeres and Wilf [118] proved that $\delta D(G) \geq \chi(G) - 1$ and thus $\delta D(G) \geq \omega(G) - 1$. Hence, the degeneracy provides a lower bound no worse than the maximum clique size and in addition it can be computed more efficiently. In Bodlaender and Koster [33] it is shown that $MCSLB(G) \geq \delta D(G)$.

Independently, Bodlaender et al. [35] and Gogate and Dechter [64] combined the minimum degree lower bound with taking minors. The so-called *contraction degeneracy* $\delta C(G)$ is defined as the maximum minimum degree over all minors of G. In [35] it is proven that computing $\delta C(G)$ is $\mathcal{NP}$-hard and computational experiments are presented by applying tiebreakers to the following algorithm: repeatedly contract a vertex of minimum degree to one of its neighbors and record the maximum encountered. Significantly better lower bounds than the degeneracy are obtained this way. In Wolle et al. [126] further results for contraction degeneracy are discussed, showing for example that $\delta C(G) \leq 5 + \gamma(G)$, where $\gamma(G)$ is the *genus* of G.

Also the lower bounds $\delta_2(G)$, $\gamma_R(G)$, and $MCSLB(G)$ can be computed over all subgraphs or minors. In [35] the combination of $MCSLB(G)$ and taking minors has been studied, whereas the combination of $\delta_2(G)$ and $\gamma_R(G)$ with taking subgraphs or minors is the topic of research in [88]. Whereas computing $\delta_2(G)$ over all subgraphs (denoted by $\delta_2 D(G)$) can be computed in polynomial time, surprisingly computing $\gamma_R(G)$ over all subgraphs (denoted by $\gamma_R D(G)$) is already $\mathcal{NP}$-hard. A 2-approximation for $\gamma_R D(G)$ is given by $\delta_2 D(G)$. Furthermore, $\delta_2 D(G) \leq \delta D(G) + 1$ and $\delta_2 C(G) \leq \delta C(G) + 1$, where $\delta_2 C(G)$ is the minor-taking variant of $\delta_2(G)$. Figure 12 shows an overview of the lower bounds for treewidth discussed so far. In practice, $\delta_2 C(G)$ and $\gamma_R C(G)$ are only marginal better (if at all) than the lower bounds computed for the contraction degeneracy.

Another vital idea to improve lower bounds for treewidth is based on the following result.

Theorem 9 (Bodlaender [28]). *Let $G = (V, E)$ be a graph with $\tau(G) \leq k$ and $\{v, w\} \notin E$. If there exist at least $k + 2$ vertex disjoint paths between v and w, then $\{v, w\} \in F$ for every triangulation H of G with $\omega(H) \leq k$.*

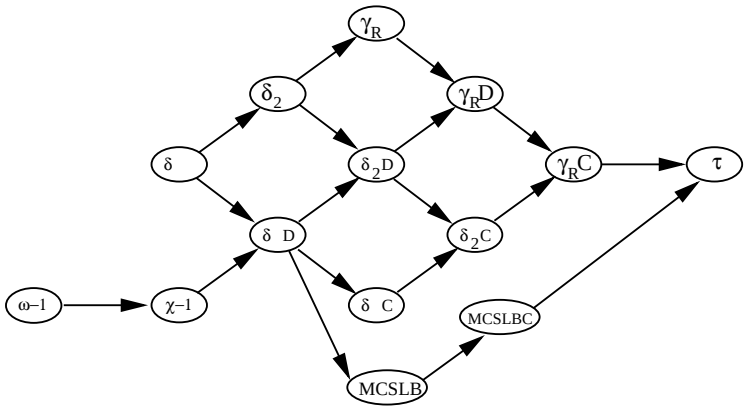


FIGURE 12. Degree-based treewidth lower bounds

Hence, if we know that $\tau(G) \leq k$ and there exist $k+2$ vertex disjoint paths between v and w, adding $\{v, w\}$ to G should not hamper the construction of a tree decomposition with small width. Clautiaux et al. [44] explored this result in a creative way. First, they compute a lower bound ℓ on the treewidth of G by any of the above methods (e.g., $\ell = \delta C(G)$). Next, they assume $\tau(G) \leq \ell$ and add edges $\{v, w\}$ to G for which there exist $\ell + 2$ vertex disjoint paths in G. Let G' be the resulting graph. Now, if it can be shown that $\tau(G') > \ell$ by a lower bound computation on G', our assumption that $\tau(G) \leq \ell$ is false. Hence, $\tau(G) > \ell$ or stated equally $\tau(G) \geq \ell + 1$: an improved lower bound for G is determined. This procedure can be repeated until it is not possible anymore to prove that $\tau(G') > \ell$ (which of course does not imply that $\tau(G') = \ell$).

In Clautiaux et al. [44], $\delta D(G')$ is used to compute the lower bounds for G'. Since computing the existence of at least $\ell + 2$ vertex disjoint paths can be quite time consuming, a simplified version checks whether v and w have at least $\ell + 2$ common neighbors. In Bodlaender et al. [36] the above described approach is nested within a minor-taking algorithm, resulting in the best known lower bounds for most tested graphs, see [122]. In many cases optimality could be proved by combining lower and upper bounds.

For graphs of low genus, in particular for planar graphs, the above described lower bounds are typically far from the real treewidth. For planar graphs, we can once more profit from Theorem 2. Treewidth is bounded from below by branchwidth and branchwidth can be computed in polynomial time on planar graphs. Hence, a polynomial time computable lower bound for treewidth of planar graphs is found. Further research in finding lower bounds (based on the concept of brambles [115]) for (near) planar graphs is underway [31]. One of these bounds is also a lower bound for branchwidth.

5.3. Tree Decomposition Based Algorithms

All efforts to compute good tree decompositions (and lower bounds on treewidth) have two major reasons:

- Several practical problems in various fields of research are equivalent to treewidth on an associated graph.
- For many $\mathcal{NP}$-hard combinatorial problems that contain a graph as part of the input, polynomial time algorithms are known in case the treewidth of the graph is bounded by some constant (as is the case for branchwidth).

For a long time, the second reason has been considered to be of theoretical value only, but (as with branchwidth) more and more practical work has been carried out in this direction.

Examples of the first reason can be found in VLSI design, Cholesky factorization, and evolution theory. We refer to Bodlaender [24] for an overview. In this context we also should mention that the control flow graph of goto-free computer programs written in common imperative programming languages like C or Pascal have treewidth bounded by small constants, see Thorup [121] and Gustedt et al. [67]. Recently, Bienstock and Ozbay [21] connected treewidth with the *Sherali-Adams operator* for 0/1 integer programs.

For many $\mathcal{NP}$-complete problems like Independent Set, Hamiltonian Circuit, Chromatic Index [23], or Steiner Tree [82] it has been shown that they can be solved in polynomial time if defined on a graph of bounded treewidth. Typically there exists a kind of dynamic programming algorithm based on the tree decomposition. Since such algorithms follow a scheme similar to the branch decomposition based algorithms described before, we leave out such a formal description (see e.g., Bodlaender [24] for a description of the algorithm for the independent set problem or Koster [83, 87] for frequency assignment).

The probably first tree decomposition based algorithm that has been shown of practical interest is given by Lauritzen and Spiegelhalter [93]. They solve the inference problem for probabilistic (or Bayesian belief) networks by using tree decompositions. Bayesian belief networks are often used in decision support systems. Applications of Bayesian belief networks can be found in medicine, agriculture, and maritime applications.

For problems where integer linear programming turns out to be troublesome, using a tree decomposition based algorithm could be a good alternative. A demonstrative example in this context is a frequency assignment problem studied by Koster [83] (see also [86, 87]). In the so-called minimum interference frequency assignment problem, we have to assign frequencies to transmitters (base stations) in a wireless network such that the overall interference is minimized. For this purpose, let $G = (V, E)$ be a graph, and for every vertex $v \in V$, a set of radio frequencies F_v is given. For every pair $\{v, w\}$ and every $f \in F_v$, $g \in F_w$, a penalty $p_{vfwg} \geq 0$ is defined. The penalties measure the interference caused by assigning two frequencies to the vertices. For v and w, $\{v, w\} \in E$ if and only if at least one penalty $p_{vfwg} > 0$. In Koster et al. [85] a cutting plane algorithm is shown to be effective only for $|F_v| \leq 6$. In practice however, $|F_v| = 40$ on average. In [83, 87], a tree decomposition based algorithm is developed for the problem. First, a tree decomposition is computed with the improvement heuristic described in Section 5.1.2. Next, the tree decomposition is used to run a dynamic programming algorithm to solve the problem. Several reduction techniques have been developed to keep the number of partial solutions to be maintained during the algorithm small. The algorithm is tested on frequency assignment problems that have been defined in the context of the CALMA project (see [2, 1] for more information on the problems and overview of the results). It was indeed possible to solve 7 out of the 11 instances to optimality by this technique. For the other instances, the computer memory was exhausted before optimality of the best known solution could be proven.

In [86] the algorithm is adapted to an interference lower bound algorithm by considering subsets of the frequencies instead of the single frequencies. Step by step the subsets are refined to improve the lower bound until either the best known solution is proved to be optimal, or computer memory prohibits further computation.

In [87], this tree decomposition based algorithm is discussed in the more general context of partial constraint satisfaction problems with binary relations. It is shown that the maximum satisfiability (MAX SAT) problem can be converted to a partial constraint satisfaction problem and computational results are presented for instances taken from the 2nd DIMACS challenge on cliques, colorings, and satisfiability [53].

Other experimental work has been carried out for vertex covering and vertex coloring. Alber et al. [3] applied a tree decomposition based algorithm for solving the vertex cover problem on planar graphs. Commandeur [46] experimented with an algorithm that solves the vertex coloring by first coloring the heaviest bag of a tree decomposition and the remaining vertices afterwards.

As already pointed out in the frequency assignment application, memory consumption is a major concern for tree decomposition based algorithms. Recently, Betzler et al. [18] have proposed a technique for reducing the memory requirements of these algorithms.

Requests for computational assistance in the construction of tree decompositions for various graphs exemplify that applying treewidth approaches to various other combinatorial problems is gaining more and more interest in fields as different as bioinformatics, artificial intelligence, operations research, and (theoretical) computer science.

6. Branchwidth, Treewidth and Matroids

6.1. Branchwidth of Matroids

It is only natural that branch decompositions can be extended to matroids. In fact, branch decompositions have been used to produce a matroid analogue of the graph minors theorem [62]. A formal definition for the branchwidth of a matroid is given below.

The reader is referred to the book by Oxley [98] if not familiar with matroid theory. Let M be a matroid with finite ground set $S(M)$ and rank function ρ. The rank function of M^*, the dual of M, is denoted ρ^*.

A *separation* (A, B) of a matroid M is a pair of complementary subsets of $S(M)$ and the order of the separation, denoted $\rho(M, A, B)$, is defined to be following:

$$\rho(M, A, B) = \begin{cases} \rho(A) + \rho(B) - \rho(M) + 1 & \text{if } A \neq \emptyset \neq B, \\ 0 & \text{else,} \end{cases}$$

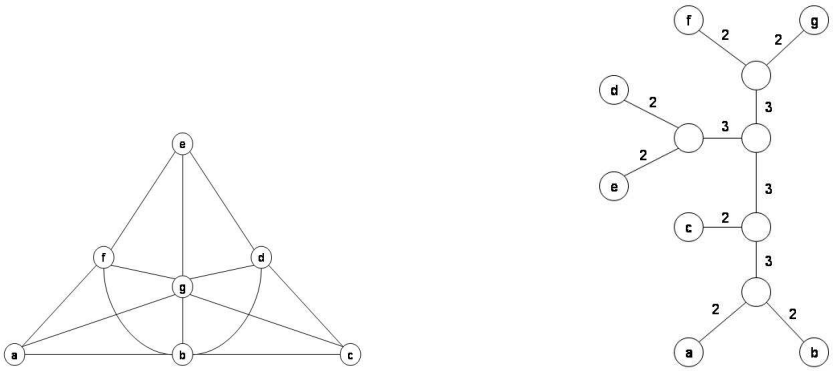
A *branch decomposition* of a matroid M is a pair (T, μ) where T is a tree having $|S(M)|$ leaves in which every non-leaf node has degree 3 and μ is a bijection from the ground set of M to the leaves of T. Notice that removing an edge, say e, of T partitions the leaves of T and the ground set of M into two subsets A_e and B_e. The *order* of e and of (A_e, B_e), denoted $order(e)$ or $order(A_e, B_e)$, is equal to $\rho(M, A_e, B_e)$. The *width* of a branch decomposition (T, μ) is the maximum order of all edges in T. The *branchwidth* of M, denoted by $\beta(M)$, is the minimum width over all branch decompositions of M. A branch decomposition of M is *optimal* if its width is equal to the branchwidth of M. For example, Figure 13 gives a Euclidean representation of a matroid and its optimal branch decomposition where all of the orders for the edges of the branch decomposition are provided.

Some results characterizing the branchwidth of matroids are given in the following lemma.

Lemma 4 (Dharmatilake [52]). *Let M be a matroid. Then, $\beta(M) = \beta(M^*)$, and if M' is a minor of M, then $\beta(M') \leq \beta(M)$.*

Lemma 5 (Dharmatilake [52]). *Let M be a matroid. Then $\beta(M) \leq 1$ if and only if M has no non-loop cycle. Moreover, $\beta(M) \leq 2$ if and only if M is the cycle matroid of a series-parallel graph.*

The cycle matroid of graph G, denoted $M(G)$, has $E(G)$ as its ground set and the cycles of G as the cycles of $M(G)$. For example, Figure 14 gives an optimal branch decomposition of



(a) Euclidean representation of the Fano Matroid

(b) Optimal branch decomposition of the Fano Matroid

FIGURE 13. Fano Matroid F_7 with Optimal Branch Decomposition (T, μ) of width 4

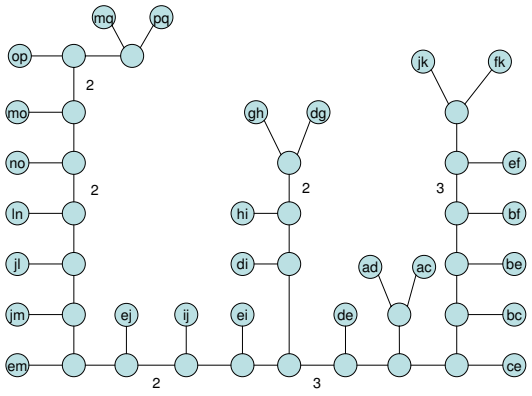


FIGURE 14. Optimal Branch Decomposition (T, μ) of width 3 for the Cycle Matroid of the graph in Figure 1

the cycle matroid of the example graph given in Figure 1 where some of the orders for the edges of the branch decomposition are provided.

In addition, there is also the concept of matroid tangles, first offered by Dharmatilake [52]. Let k be a positive integer and let M be a matroid. A *tangle* of order k in M is a set $\mathcal{T}$ of $< k$ separations of M such that:

- for every separation (A, B) of M such that $\rho(M, A, B) < k$, either $(A, B) \in \mathcal{T}$ or $(B, A) \in \mathcal{T}$;
- if (A_1, B_1), (A_2, B_2), $(A_3, B_3) \in \mathcal{T}$ then $A_1 \cup A_2 \cup A_3 \neq S(M)$; and
- if $(A, B) \in \mathcal{T}$ then $\rho(A) < \rho(M)$.

They are referred to as the first, second and third matroid tangle axioms. Likewise, the *tangle number* of a matroid M, denoted $\theta(M)$, is the largest order of a tangle for M. Notice

that the third matroid tangle axiom offers instances where the tangle number could be infinity. This occurs when the matroid has a coloop because the coloop can always be on the second part of the separations of a tangle. In addition, Dharmatilake gave a min-max relationship between tangles of matroids and the branchwidth of matroids, given below.

Theorem 10 (Dharmatilake [52]). *Let M be a matroid. Then $\beta(M) = \theta(M)$ if and only if M has no coloop and $\beta(M) \neq 1$.*

It was conjectured by Geelen et al. [61] that the branchwidth of a graph and the branchwidth of the graph's cycle matroid are equal if the graph has a cycle of length at least 2. Similar to the work on characterizing the classes of graphs with bounded branchwidth, there have been some developments in characterizing the classes of matroids with bounded branchwidth. Most notable is the work of Robertson and Seymour [106] characterizing the class of matroids with branchwidth at most 2 and the work of Geelen et al. [62] proving that the size of excluded minors for the class of matroids with bounded branchwidth is finite. In addition, Hall et al. [68] showed that there are at most 14 excluded minors of the class of matroids with branchwidth at most 3. Recently, Oum and Seymour [97] have derived a polynomial time algorithm to approximate the branchwidth of any matroid similar the algorithm offered by Robertson and Seymour [108] for graphs. One is also referred to the work of Hliněný [77] for more detailed discussions on the branchwidth of matroids.

6.2. Treewidth of Matroids

In contrast to branchwidth, it is not straightforward to generalize the notion of treewidth to matroids. Where the width of a branch decomposition is defined on edges of the graph, the width of a tree decompositions is defined on graph vertex sets (the bags). The ground set of the cycle matroid corresponds with the edge set of the associated graph, whereas the vertices do not play a role in the cycle matroid.

Hliněný and Whittle [78], inspired by Geelen, state a definition for treewidth of matroids and show that in case of the cycle matroid of a graph this definition is equivalent to the graph-based definition, by providing an alternative characterization of the treewidth of graphs.

Let M be a matroid on the ground set $S(M)$. A pair (T, μ), where T is a tree and $\mu : S \to V(T)$ is an arbitrary mapping, is called a tree decomposition of M. For a node x of T, denote the connected components of $T - x$ by $T_1, \ldots, T_d$ and set $F_i = \mu^{-1}(V(T_i))$. The *width of* x is

$$\text{(MW)} \quad \sum_{i=1}^{d} \rho_M(S - F_i) - (d-1) \cdot \rho(M),$$

and the *width* of the decomposition (T, μ) is the maximum width over all nodes of T. The *treewidth* $\tau(M)$ of M is the minimum width over all tree decompositions of M.

Theorem 11 (Hliněný and Whittle [78]). *Let G be a graph with at least one edge, and let $M(G)$ be the cycle matroid of G. Then $\tau(G) = \tau(M(G))$.*

Note that the mapping μ in the definition has nothing to do with the bags X_i defined in the graph version of tree decompositions. The mapping μ guarantees that property *(TD2)* is fulfilled, whereas the interpolation property *(TD3)* is hidden in *(MW)*. The width of a node x of T can be rewritten as

$$\rho(M) - \sum_{i=1}^{d} \left(\rho(M) - \rho_M(S - F_i)\right),$$

which can be seen as minimizing the rank defect of the branches of each node (here rank defect is defined as $\rho(M) - \rho_M(S - F)$ for a set $F \subseteq S$).

Besides the equivalence of graph and matroid treewidth, Hliněný and Whittle [78] also prove the relation between matroid branchwidth and matroid treewidth, similar to Theorem 2. They show that $\beta(M) \leq \tau(M) + 1 \leq \max(2\beta(M) - 1, 2)$ holds for all matroids M. Hence all results for matroids of bounded branchwidth carry over to matroids of bounded treewidth and vice versa. Note that the upper bound is not as tight as in the case of graphs.

7. Open Problems

Related to branchwidth and treewidth, the following open questions are worth considering:

- Does there exist a polynomial time algorithm to compute the branchwidth and optimal branch decomposition of a planar graph with complexity smaller than $O(n^3)$?
- Is the branchwidth of a graph and its cycle matroid equal if the graph has a cycle of length at least two?
- Do there exist polynomial time algorithms to compute the branchwidth and optimal branch decomposition of graphs embeddable on orientable surfaces on than the sphere such as the torus or double torus?
- Do there exist (practical) integer programming formulations for finding the branchwidth or treewidth of a graph?
- Does there exist a polynomial time approximation algorithm for treewidth with constant approximation guarantee?
- Does there exist a polynomial time algorithm for computing treewidth of a planar graph, or a proof that this problem is $\mathcal{NP}$-hard?
- Do heuristics other than MCS have a lower bounding counterpart?
- How good can the contraction degeneracy $\delta C(G)$ be in general graphs? Is there a bound (different from treewidth) that limits this parameter?
- Do there exist computational efficient algorithms to compute the treewidth of general graphs? Are the exponential time algorithms useful for practical computations?

Finally, the real application of the methodology to other combinatorial problems is of importance for both the operations research community and the algorithmic graph theory community. For operations researchers, it is good to have alternative methodologies than integer linear programming that indeed solve real-world problems. For the algorithmic graph theorists more applications will drive the research to improved lower and upper bounds for treewidth or related notions.

8. Acknowledgements

The authors would like to acknowledge the support of the National Science Foundation (grant DMI-0217265), the DFG research group "Algorithms, Structure, Randomness" (grant GR 883/9-3, GR 883/9-4), and the Netherlands Organization for Scientific Research (project *Treewidth and Combinatorial Optimization*).

References

[1] K. I. Aardal, C. A. J. Hurkens, J. K. Lenstra, and S. R. Tiourine. Algorithms for radio link frequency assignment: The CALMA project. *Operations Research*, 50(6):968–980, 2003.

[2] K. I. Aardal, C. P. M. van Hoesel, A. M. C. A. Koster, C. Mannino, and A. Sassano. Models and solution techniques for the frequency assignment problem. *4OR*, 1(4):261–317, 2003.

[3] J. Alber, F. Dorn, and R. Niedermeier. Experimental evaluation of a tree decomposition based algorithm for vertex cover on planar graphs. *Disc. Appl. Math.*, 145:210–219, 2004.

[4] J. Alber and R. Niedermeier. Improved tree decomposition based algorithms for domination-like problems. In *Proceedings of the 5th Latin American Theoretical Informatics (LATIN 2002)*, pages 613–627, Heidelberg, Germany, 2002. Springer-Verlag. Lecture Notes in Computer Science 2286.

[5] M. Alekhnovich and A. Razborov. Satisfiability, branch-width and Tseitin tautologies. In *43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 593–603. IEEE Computer Society, 2002.

[6] N. Alon. Eigenvalues and expanders. *Combinatorica*, 2:83–96, 1986.

[7] N. Alon, P. D. Seymour, and R. Thomas. Planar separators. *SIAM Journal on Discrete Mathematics*, 7:184–193, 1994.

[8] C. Alvarez, R. Cases, J. Diaz, J. Petit, and M. Serna. Routing tree problems on random graphs. Technical Report LSI-01-10-R, Software Department, Universitat Politecnica de Catalunya, Barcelona, Spain, 2000.

[9] E. Amir. Efficient approximations for triangulation of minimum treewidth. In *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence*, pages 7–15, 2001.

[10] D. Archdeacon. *A Kuratowski Theorem for the Projective Plane*. PhD thesis, Ohio State University, 1980.

[11] D. Archdeacon and P. Huneke. A Kuratowski theorem for non-orientable surfaces. *Journal of Combinatorial Theory, Series B*, 46(2):173–231, 1989.

[12] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Alg. Disc. Meth.*, 8:277–284, 1987.

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

[14] S. Arnborg and A. Proskurowski. Characterization and recognition of partial 3-trees. *SIAM J. Alg. Disc. Meth.*, 7:305–314, 1986.

[15] S. Arnborg, A. Proskurowski, and D. Corneil. Forbidden minors characterization of partial 3-trees. *Discrete Mathematics*, 80:1–19, 1990.

[16] E. Bachoore and H. L. Bodlaender. New upper bound heuristics for treewidth. In S. Nikoletseas, editor, *WEA 2005: Workshop on Efficient and Experimental Algorithms*. Springer, Lecture Notes in Computer Science, to appear, 2005.

[17] A. Berry, J. Blair, P. Heggernes, and B. Peyton. Maximum cardinality search for computing minimal triangulations of graphs. *Algorithmica*, 39:287–298, 2004.

[18] N. Betzler, R. Niedermeier, and J. Uhlmann. Tree decompositions of graphs: Saving memory in dynamic programming. In *CTW 2004: Cologne-Twente Workshop on Graphs and Combinatorial Optimization*, pages 56–60, Villa Vigoni (CO), Italy, 2004.

[19] D. Bienstock. *Reliability of Computer and Communication Networks*, volume 5 of *DIMACS Ser. In Discrete Math. and Theoret. Comput. Sci.*, chapter Graph Searching, Path-Width, Tree-Width and Related Problems (a Survey), pages 33–49. AMS, Providence, RI, 1991.

[20] D. Bienstock and M. A. Langston. Algorithmic implications of the graph minor theorem. In *Network Models, Handbook of Operations Research and Management Science*, chapter Algorithmic Implications of the Graph Minor Theorem, pages 481–502. North-Holland, Amsterdam, 1995.

[21] D. Bienstock and N. Ozbay. Tree-width and the Sherali-Adams operator. *Discrete Optimization*, 1:13–21, 2004.

[22] J. R. S. Blair, P. Heggernes, and J. Telle. A practical algorithm for making filled graphs minimal. *Theor. Comp. Sc.*, 250:125–141, 2001.

[23] H. L. Bodlaender. Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees. *J. Algorithms*, 11:631–643, 1990.

[24] H. L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11:1–21, 1993.

[25] H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25:1305–1317, 1996.

[26] H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25:1305–1317, 1996.

[27] H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theor. Comp. Sc.*, 209:1–45, 1998.

[28] H. L. Bodlaender. Necessary edges in k-chordalizations of graphs. *Journal of Combinatorial Optimization*, 7:283–290, 2003.

[29] H. L. Bodlaender. Discovering treewidth. In P. Vojtáš, M. Bieliková, B. Charron-Bost, and O. Sýkora, editors, *SOFSEM 2005: Theory and Practice of Computer Science*, pages 1–16. Springer, Lecture Notes in Computer Science, vol. 3381, 2005.

[30] H. L. Bodlaender and F. Fomin. Equitable colorings of bounded treewidth graphs. Technical Report UU-CS-2004-010, Institute of Information and Computing Sciences, Utrecht University, Netherlands, 2004.

[31] H. L. Bodlaender, A. Grigoriev, and A. M. C. A. Koster. Treewidth lower bounds with brambles. Work in progress, 2005.

[32] H. L. Bodlaender and T. Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *Journal of Algorithms*, 21:358–402, 1996.

[33] H. L. Bodlaender and A. M. C. A. Koster. Safe separators for treewidth. In *Proceedings 6th Workshop on Algorithm Engineering and Experiments ALENEX04*, pages 70–78, 2004.

[34] H. L. Bodlaender, A. M. C. A. Koster, F. van den Eijkhof, and L. C. van der Gaag. Preprocessing for triangulation of probabilistic networks. In J. Breese and D. Koller, editors, *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence*, pages 32–39, San Francisco, 2001. Morgan Kaufmann.

[35] H. L. Bodlaender, A. M. C. A. Koster, and T. Wolle. Contraction and treewidth lower bounds. In S. Albers and T. Radzik, editors, *Proceedings 12th Annual European Symposium on Algorithms, ESA2004*, pages 628–639. Springer, Lecture Notes in Computer Science, vol. 3221, 2004.

[36] H. L. Bodlaender, A. M. C. A. Koster, and T. Wolle. Contraction and treewidth lower bounds. Technical Report UU-CS-2004-34, Dept. of Computer Science, Utrecht University, Utrecht, The Netherlands, 2004.

[37] H. L. Bodlaender and D. Thilikos. Constructive linear time algorithms for branchwidth. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *Lecture Notes in Computer Science: Proceedings of the 24th International Colloquium on Automata, Languages, and Programming*, pages 627–637. Springer Verlag, 1997.

[38] H. L. Bodlaender and D. Thilikos. Graphs with branchwidth at most three. *Journal of Algorithms*, 32:167–194, 1999.

[39] H. L. Bodlaender and B. van Antwerpen-de Fluiter. Parallel algorithms for series parallel graphs and graphs with treewidth two. *Algorithmica*, 29:543–559, 2001.

[40] V. Bouchitté, D. Kratsch, H. Müller, and I. Todinca. On treewidth approximations. *Disc. Appl. Math.*, 136:183–196, 2004.

[41] V. Bouchitté, F. Mazoit, and I. Todinca. Chordal embeddings of planar graphs. *Discrete Mathematics*, 273:85–102, 2003.

[42] G. Chang and D. Kuo. The $l(2,1)$-labeling problem on graphs. *SIAM J. Discrete Math.*, 9:309–316, 1996.

[43] W. A. Christian. *Linear-Time Algorithms for Graphs with Bounded Branchwidth*. PhD thesis, Rice University, 2003.

[44] F. Clautiaux, J. Carlier, A. Moukrim, and S. Négre. New lower and upper bounds for graph treewidth. In J. D. P. Rolim, editor, *Proceedings International Workshop on Experimental and Efficient Algorithms, WEA 2003*, pages 70–80. Springer Verlag, Lecture Notes in Computer Science, vol. 2647, 2003.

[45] F. Clautiaux, A. Moukrim, S. Négre, and J. Carlier. Heuristic and meta-heuristic methods for computing graph treewidth. *RAIRO Oper. Res.*, 38:13–26, 2004.

[46] M. Commandeur. Solving vertex coloring using tree decompositions. Master's thesis, Universiteit Maastricht, The Netherlands, 2004.

[47] W. Cook and P. D. Seymour. An algorithm for the ring-routing problem. Bellcore technical memorandum, Bellcore, 1994.

[48] W. Cook and P. D. Seymour. Tour merging via branch-decomposition. *INFORMS Journal on Computing*, 15(3):233–248, 2003.

[49] B. Courcelle. Graph rewriting: An algebraic and logic approach. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 194–242. Elsevier, 1990.

[50] B. Courcelle. The monadic second-order logic of graphs I: Recognizable sets of finite graphs. *Information and Computation*, 85:12–75, 1990.

[51] E. Dahlhaus. Minimal elimination ordering inside a given chordal graph. In *Proceedings 23rd International Workshop on Graph-Theoretic Concepts in Computer Science WG'97*, pages 132–143. Springer Verlag, Lecture Notes in Computer Science, vol. 1335, 1997.

[52] J. S. Dharmatilake. A min-max theorem using matroid separations. *Contemporary Mathematics*, 197:333–342, 1996.

[53] The second DIMACS implementation challenge: NP-Hard Problems: Maximum Clique, Graph Coloring, and Satisfiability. See http://dimacs.rutgers.edu/Challenges/, 1992–1993.

[54] J. Fiala, P. Glovach, and J. Kratochvil. Distance constrained labelings of graphs of bounded treewidth. In *Proceedings of 32nd International Colloquium on Autmata, Languages and Programming*, 2005. to appear.

[55] F. Fomin, P. Fraigniaud, and D. Thilikos. The price of connectedness in expansions. Technical Report LSI-04-28-R, Departament de Lenguatges i Sistemes Informàtics, Universitat Politécnica de Catalunya, Spain, 2004.

[56] F. Fomin and D. Thilikos. Dominating sets in planar graphs: Branch-width and exponential speed-up. In *Proceedings of the Fourthteenth Annual ACM-SIAM Symposium on Discrete Algorithms (Baltimore, MD 2003)*, pages 168–177, New York, 2003. ACM.

[57] F. Fomin and D. Thilikos. A simple and fast approach for solving problems on planar graphs. In *STACS 2004*, 2004.

[58] F. V. Fomin, D. Kratsch, and I. Todinca. Exact (exponential) algorithms for treewidth and minimum fill-in. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming*, pages 568–580, 2004.

[59] M. R. Garey, R. L. Graham, D. S. Johnson, and D. E. Knuth. Complexity results for bandwidth minimization. *SIAM J. Appl. Math.*, 34:477–495, 1978.

[60] F. Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *J. Comb. Theory Series B*, 16:47–56, 1974.

[61] J. F. Geelen, A. M. H. Gerards, N. Robertson, and G. P. Whittle. On the excluded minors for the matroids of branch-width k. *Journal of Combinatorial Theory Series B*, 88:261–265, 2003.

[62] J. F. Geelen, A. M. H. Gerards, and G. Whittle. Branch width and well-quasi-ordering in matroids and graphs. *Journal of Combinatorial Theory, Series B*, 84:270–290, 2002.

[63] H. Glover, P. Huneke, and C. S. Wang. 103 graphs that are irreducible for the projective plane. *Journal of Combinatorial Theory, Series B*, 27:332–370, 1979.

[64] V. Gogate and R. Dechter. A complete anytime algorithm for treewidth. In proceedings UAI'04, Uncertainty in Artificial Intelligence, 2004.

[65] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York, 1980.

[66] Q.-P. Gu and H. Tamaki. Optimal branch-decomposition of planar graphs in $O(n^3)$ time. preprint, 2005.

[67] J. Gustedt, O. A. Mæhle, and J. A. Telle. The treewidth of Java programs. In David M. Mount and Clifford Stein, editors, *Proceedings 4th International Workshop on Algorithm Engineering and Experiments*, pages 86–97. Springer Verlag, Lecture Notes in Computer Science, vol. 2409, 2002.

[68] R. Hall, J. Oxley, C. Semple, and G. Whittle. On matroids of branch-width three. *Journal of Combinatorial Theory Series B*, 86:148–171, 2002.

[69] P. Heggernes, J. A. Telle, and Y. Villanger. Computing minimal triangulations in time $O(n^\alpha \log n) = o(n^{2.376})$. To appear in proceedings SODA'05, 2005.

[70] P. Heggernes and Y. Villanger. Efficient implementation of a minimal triangulation algorithm. In R. Möhring and R. Raman, editors, *Proceedings of the 10th Annual European Symposium on Algorithms, ESA'2002*, pages 550–561. Springer Verlag, Lecture Notes in Computer Science, vol. 2461, 2002.

[71] I. V. Hicks. *Branch Decompositions and their Applications*. PhD thesis, Rice University, 2000.

[72] I. V. Hicks. Branchwidth heuristics. *Congressus Numerantium*, 159:31–50, 2002.

[73] I. V. Hicks. Branch decompositions and minor containment. *Networks*, 43(1):1–9, 2004.

[74] I. V. Hicks. Graphs, branchwidth, and tangles! oh my! *Networks*, 2005. to appear.

[75] I. V. Hicks. Planar branch decompositions I: The ratcatcher. *INFORMS Journal on Computing*, 2005. to appear.

[76] I. V. Hicks. Planar branch decompositions II: The cycle method. *INFORMS Journal on Computing*, 2005. to appear.

[77] P. Hliněný. On the excluded minors for matroids of branch-width three. preprint, 2002.

[78] P. Hliněný and G. Whittle. Matroid tree-width. Technical report, Technical University Ostrava, Ostrava, Czech Republic, 2003.

[79] U. Kjærulff. Optimal decomposition of probabilistic networks by simulated annealing. *Statistics and Computing*, 2:2–17, 1992.

[80] T. Kloks, J. Kratochvil, and H. Müller. New branchwidth territories. In C. Meinel and S. Tison, editors, *STAC '99, 16th Annual Symposium on Theoretical Aspects of Computer Science, Trier, Germany, March 1999 Proceedings*, pages 173–183, Berlin, 1999. Springer-Verlag.

[81] T. Kloks, J. Kratochvil, and H. Müller. Computing the branchwidth of interval graphs. *Discrete Applied Mathematics*, 145:266–275, 2005.

[82] E. Korach and N. Solel. Linear time algorithm for minimum weight Steiner tree in graphs with bounded treewidth. Manuscript, 1990.

[83] A. M. C. A. Koster. *Frequency Assignment - Models and Algorithms*. PhD thesis, Univ. Maastricht, Maastricht, The Netherlands, 1999.

[84] A. M. C. A. Koster, H. L. Bodlaender, and S. P. M. van Hoesel. Treewidth: Computational experiments. In H. Broersma, U. Faigle, J. Hurink, and S. Pickl, editors, *Electronic Notes in Discrete Mathematics*, volume 8. Elsevier Science Publishers, 2001.

[85] A. M. C. A. Koster, C. P. M. van Hoesel, and A. W. J. Kolen. The partial constraint satisfaction problem: Facets and lifting theorems. *Operations Research Letters*, 23(3–5):89–97, 1998.

[86] A. M. C. A. Koster, C. P. M. van Hoesel, and A. W. J. Kolen. Lower bounds for minimum interference frequency assignment problems. *Ricerca Operativa*, 30(94–95):101–116, 2000.

[87] A. M. C. A. Koster, S. P. M. van Hoesel, and A. W. J. Kolen. Solving partial constraint satisfaction problems with tree decomposition. *Networks*, 40:170–180, 2002.

[88] A. M. C. A. Koster, T. Wolle, and H. L. Bodlaender. Degree-based treewidth lower bounds. In S. Nikoletseas, editor, *WEA 2005: Workshop on Efficient and Experimental Algorithms*. Springer, Lecture Notes in Computer Science, to appear, 2005.

[89] J. Kruskal. Well-quasi-ordering, the tree theorem, and Vazsonyi's conjecture. *Trans. American Mathematical Society*, 95:210–225, 1960.

[90] K. Kuratowski. Sur le probleme des courbes gauches en topologie. *Fundamenta Mathematicae*, 15:271–283, 1930.

[91] D. Lapoire. Treewidth and duality for planar hypergraphs. preprint, 2002.

[92] P. Larrañaga, C. M. H. Kuijpers, M. Poza, and R. H. Murga. Decomposing Bayesian networks: triangulation of the moral graph with genetic algorithms. *Statistics and Computing (UK)*, 7(1):19–34, 1997.

[93] S. J. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *The Journal of the Royal Statistical Society. Series B (Methodological)*, 50:157–224, 1988.

[94] B. Lucena. A new lower bound for tree-width using maximum cardinality search. *SIAM J. Disc. Math.*, 16:345–353, 2003.

[95] B. Monien. The bandwidth minimization problem for caterpillars with hair length 3 is NP-complete. *SIAM J. Alg. Disc. Meth.*, 7:505–512, 1986.

[96] C. Nash-Williams. On well-quasi-ordering infinite trees. *Proceedings of the Cambridge Philosophical Society*, 61:697–720, 1965.

[97] S.-I. Oum and P. D. Seymour. Approximating clique-width and branch-width. Technical report, Mathematics Department, Princeton University, 2005. preprint.

[98] J. G. Oxley. *Matroid Theory*. Oxford University Press, Oxford, UK, 1992.

[99] S. Ramachandramurthi. *Algorithms for VLSI Layout Based on Graph Width Metrics*. PhD thesis, Computer Science Department, University of Tennessee, Knoxville, Tennessee, USA, 1994.

[100] S. Ramachandramurthi. The structure and number of obstructions to treewidth. *SIAM J. Disc. Math.*, 10:146–157, 1997.

[101] B. Reed. Tree width and tangles: A new connectivity measure and some applications. In R. A. Bailey, editor, *Survey in Combinatorics, 1997*, pages 87–162. Cambridge University Press, Cambridge, 1997.

[102] G. Reinelt. TSPLIB - a traveling salesman library. *ORSA Journal on Computing*, 3:376–384, 1991. URL: http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/.

[103] N. Robertson and P. D. Seymour. Graph minors: A survey. In *Surveys in Combinatorics*, London Math Society Lecture Note Series, pages 153–171. Cambridge University Press, Cambridge-New York, 103 edition, 1985.

[104] N. Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *J. Algorithms*, 7:309–322, 1986.

[105] N. Robertson and P. D. Seymour. Graph minors IV: Treewidth and well-quasi-ordering. *Journal of Combinatorial Theory, Series B*, 48:227–254, 1990.

[106] N. Robertson and P. D. Seymour. Graph minors. X. Obstructions to tree-decomposition. *J. Comb. Theory Series B*, 52:153–190, 1991.

[107] N. Robertson and P. D. Seymour. Graph minors XI: Circuits on a surface. *Journal of Combinatorial Theory, Series B*, 60:72–106, 1994.

[108] N. Robertson and P. D. Seymour. Graph minors XIII: The disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63:65–110, 1995.

[109] N. Robertson and P. D. Seymour. Graph minors XX: Wagner's conjecture. *Journal of Combinatorial Theory Series B*, 92(2):325–357, 2004.

[110] H. Röhrig. Tree decomposition: A feasibility study. Master's thesis, Max-Planck-Institut für Informatik, Saarbrücken, Germany, 1998.

[111] D. J. Rose, R. E. Tarjan, and G. S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.*, 5:266–283, 1976.

[112] D. Sanders. On linear recognition for tree-width at most four. *SIAM Journal of Discrete Math*, 9:101–117, 1996.

[113] A. Satyanarayana and L. Tung. A characterization of partial 3-trees. *Networks*, 20:299–322, 1990.

[114] P. Scheffler. *Die Baumweite von Graphen als ein Maß für die Kompliziertheit algorithmischer Probleme*. PhD thesis, Akademie der Wissenschaften der DDR, Berlin, 1989.

[115] P. D. Seymour and R. Thomas. Graph searching and a minmax theorem for tree-width. *J. Comb. Theory Series B*, 58:239–257, 1993.

[116] P. D. Seymour and R. Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994.

[117] K. Shoikhet and D. Geiger. A practical algorithm for finding optimal triangulations. In *Proc. National Conference on Artificial Intelligence (AAAI '97)*, pages 185–190. Morgan Kaufmann, 1997.

[118] G. Szekeres and H. S. Wilf. An inequality for the chromatic number of a graph. *J. Comb. Theory*, 4:1–3, 1968.

[119] H. Tamaki. A linear time heuristic for the branch-decomposition of planar graphs. Technical Report MPI-I-2003-1-010, Max-Planck-Institut Fur Informatick, 2003.

[120] R. E. Tarjan and M. Yannakakis. Simple linear time algorithms to test chordiality of graphs, test acyclicity of graphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, 13:566–579, 1984.

[121] M. Thorup. All structured programs have small tree width and good register allocation. *Information and Computation*, 142:159–181, 1998.

[122] Treewidthlib. http://www.cs.uu.nl/people/hansb/treewidthlib, 2004-03-31.

[123] Z. Tuza. Strong branchwidth and local transversals. *Discrete Applied Mathematics*, 145:291–296, 2005.

[124] K. Wagner. Uber eine eigenschaft der ebenen komplexe. *Mathematische Annalen*, 115:570–590, 1937.

[125] G. J. Woeginger. Exact algorithms for NP-hard problems: A survey. In *Combinatorial Optimization: "Eureka, You Shrink!"*, pages 185–207, Berlin, 2003. Springer Lecture Notes in Computer Science, vol. 2570.

[126] T. Wolle, A. M. C. A. Koster, and H. L. Bodlaender. A note on contraction degeneracy. Technical Report UU-CS-2004-042, Institute of Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands, 2004.

[127] M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM J. Alg. Disc. Meth.*, 2:77–79, 1981.