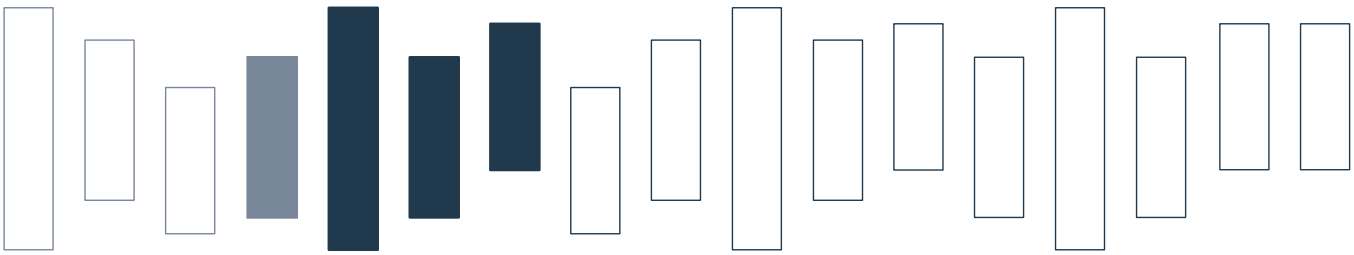
Technical Report AC-TR-16-004

April 2016

A SAT Approach to Branchwidth

Neha Lodha, Sebastian Ordyniak, and Stefan Szeider



This is the authors' copy of a paper that appears in the Proceedings of SAT, 2016.

www.ac.tuwien.ac.at/tr

A SAT Approach to Branchwidth*

Neha Lodha, Sebastian Ordyniak, and Stefan Szeider

Algorithms and Complexity Group, TU Wien, Vienna, Austria
[neha,ordyniak,sz]@ac.tuwien.ac.at

Abstract. Branch decomposition is a prominent method for structurally decomposing a graph, hypergraph or CNF formula. The width of a branch decomposition provides a measure of how well the object is decomposed. For many applications it is crucial to compute a branch decomposition whose width is as small as possible. We propose a SAT approach to finding branch decompositions of small width. The core of our approach is an efficient SAT encoding which determines with a single SAT-call whether a given hypergraph admits a branch decomposition of certain width. For our encoding we developed a novel partition-based characterization of branch decomposition. The encoding size imposes a limit on the size of the given hypergraph. In order to break through this barrier and to scale the SAT approach to larger instances, we developed a new heuristic approach where the SAT encoding is used to locally improve a given candidate decomposition until a fixed-point is reached. This new method scales now to instances with several thousands of vertices and edges.

Technical Report AC-TR-16-004

1 Introduction

Background Branch decomposition is a prominent method for structurally decomposing a graph or hypergraph. This decomposition method was originally introduced by Robertson and Seymour [17] in their Graph Minors Project and has become a key notion in discrete mathematics and combinatorial optimization. Branch decompositions can be used to decompose other combinatorial objects such as matroids, integer-valued symmetric submodular functions, and propositional CNF formulas (after dropping of negations, clauses can be considered as (hyper-)edges). The width of a branch decomposition provides a measure of how well it decomposes the given object; the smallest width over its branch decompositions denotes the *branchwidth* of an object. Many hard computational problems can be solved efficiently by means of dynamic programming along a branch decomposition of small width. Prominent examples include the traveling salesman problem [6], the #P-complete problem of propositional model counting [3], and the generation of resolution refutations for unsatisfiable CNF formulas [2]. In fact, all decision problems on graphs that can be expressed in monadic second order logic can be solved in linear time on graphs that admit a branch decomposition of bounded width [10].

A bottleneck for all these algorithmic applications is the space requirement of dynamic programming, which is typically single or double exponential in the width of

* Dedicated to the memory of Helmuth Veith (1971–2016)

the given branch decomposition. Hence it is crucial to compute first a branch decomposition whose width is as small as possible. This is very similar to the situation in the context of treewidth, where the following was noted about inference on probabilistic networks of bounded treewidth [15]:

> *[. . .] since inference is exponential in the tree-width, a small reduction in tree-width (say by even by* 1 *or* 2*) can amount to one or two orders of magnitude reduction in inference time.*

Hence small improvements in the width can change a dynamic programming approach from unfeasible to feasible. The boundary between unfeasible and feasible width values strongly depends on the considered problem and the currently available hardware. For instance, Cook and Seymour [6] mention a threshold of 20 for the Traveling Salesman Problem in 2003. Today one might consider a higher threshold. Computing an optimal branch decomposition is NP-hard [19].

Contribution In this paper we propose a practical SAT-based approach to finding a branch decompositions of small width. At the core of our approach is an efficient SAT encoding which takes a hypergraph H and an integer w as input and produces a propositional CNF formula which is satisfiable if and only if H admits a branch decomposition of width w. By multiple calls of the solver with various values of w we can determine the smallest w for which the formula is satisfiable (i.e., the branchwidth of H), and we can transform the satisfying assignment into an optimal branch decomposition. Our encoding is based on a novel *partition-based* characterization of branch decompositions in terms of certain sequences of partitions of the set of edges. This characterization together with clauses that express cardinality constraints allow for an efficient SAT encoding that scales up to instances with about hundred edges. The computationally most expensive part in this procedure is to determine the optimality of w by checking that the formula corresponding to a width of $w - 1$ is unsatisfiable. If we do not insist on optimality and aim at good upper bounds, we can scale the approach to larger hypergraphs with over two hundred edges.

The number of clauses in the formula is polynomial in the size of the hypergraph and the given width w, but the order of the polynomial can be quintic, hence there is a firm barrier to the scalability of the approach to larger hypergraphs. In order to break through this barrier, we developed a new *SAT-based local improvement* approach where the encoding is not applied to the entire hypergraph but to certain smaller hypergraphs that represent local parts of a current candidate branch decomposition. The overall procedure thus starts with a branch decomposition obtained by a heuristic method and then tries to improve it locally by multiple SAT-calls until a fixed-point (or timeout) is reached. This method scales now to instances with several thousands of vertices and edges and branchwidth upper bounds well over hundred. We believe that a similar approach using a SAT-based local improvement could also be developed for other (hyper)graph width measures.

Related Work Previously, SAT techniques have been proposed for other graph width measures: Samer and Veith [18] proposed a SAT encoding for *treewidth*, based on a characterization of treewidth in terms of elimination orderings (that is, the encoding

Technical Report AC-TR-16-004

entails variables whose truth values determine a permutation of the vertices, and the width of a corresponding decomposition is then bounded by cardinality constraints). This approach was later improved by Berg and Järvisalo [4] who empirically evaluated various SAT and MaxSAT strategies for treewidth encodings based on elimination orderings. Heule and Szeider [11] developed a SAT approach for computing the *clique-width* of graphs. For this purpose they developed a novel partition-based characterization of clique-width. Our encoding of branchwidth was inspired by this. However, the two encodings are different as clique-width and branchwidth are entirly different notions.

For finding branch decompositions of smallest width, Robertson and Seymour [17] suggested an exponential-time algorithm which was later implemented by Hicks [12]. Further exponential-time algorithms have been proposed (see, for instance [9, 14]) but there seem to be no implementations. Ulusal [20] proposed an encoding to integer programming (CPLEX). One could also find suboptimal branch decompositions based on the related notion of tree decompositions; however, finding an optimal tree decomposition is again NP-hard, and by transforming it into a branch decomposition one introduces an approximation error factor of up to 50% [17] which makes this approach prohibitive in practice. For practical purposes one therefore mainly resorts to heuristic methods that compute suboptimal branch decompositions [6, 13, 16].

Due to the space restrictions several proofs have been omitted.

Technical Report AC-TR-16-004

2 Preliminaries

Formulas and Satisfiability We consider propositional formulas in Conjunctive Normal Form (*CNF formulas*, for short), which are conjunctions of clauses, where a clause is a disjunction of literals, and a literal is a propositional variable or a negated propositional variables. A CNF formula is *satisfiable* if its variables can be assigned true or false, such that each clause contains either a variable set to true or a negated variable set to false. The satisfiability problem (SAT) asks whether a given formula is satisfiable.

Graphs and Branchwidth We consider finite hypergraphs and undirected graphs. For basic terminology on graphs we refer to a standard text book [8]. For a (hyper-)graph H we denote by $V(H)$ the vertex set of H and by $E(H)$ the edge set of H. If $E \subseteq E(H)$, we denote by $H \setminus E$ the hypergraph with vertices $V(H)$ and edges $E(H) \setminus E$. Let G be a simple undirected graph. The *radius* of G, denoted by $\mathrm{rad}(G)$, is the minimum integer r such that G has a vertex from which all other vertices are reachable via a path of length at most $\mathrm{rad}(G)$. The *center* of G is the set of vertices such that all other vertices of G can be reached via a path of length at most $\mathrm{rad}(G)$. We will often consider various forms of trees, i.e., connected acyclic graphs, as they form the backbone of branch decompositions. Let T be an undirected tree. We will always assume that T is rooted (in some arbitrary vertex r) and hence the parent and child relationships between its vertices are well-defined. We say that T is ternary if every non-leaf vertex of T has degree exactly three. We will write $\mathrm{p}_T(t)$ (or just $\mathrm{p}(t)$ if T is clear from the context) to denote the parent of $t \in V(T)$ in T. We also write T_t to denote the subtree of T rooted in t, i.e., the component of $T \setminus \{\{t, \mathrm{p}_T(t)\}\}$ containing t. For a tree T, we denote by

h(T), the *height* of T, i.e., the length of a longest path between the root and any leaf of T plus one. It is well-known that every tree has at most two center vertices, moreover, if it has two center vertices then they form the endpoints of an edge in the tree.

Let H be a hypergraph. Every subset E of $E(H)$ defines a *cut* of H, i.e., the pair $(E, E(H) \setminus E)$. We denote by $\delta_H(E)$ (or just $\delta(E)$ if H is clear form the context) the set of *cut vertices* of E in H, i.e., $\delta(E)$ contains all vertices incident to both an edge in E and an edge in $E(H) \setminus E$. Note that $\delta(E) = \delta(E(H) \setminus E)$.

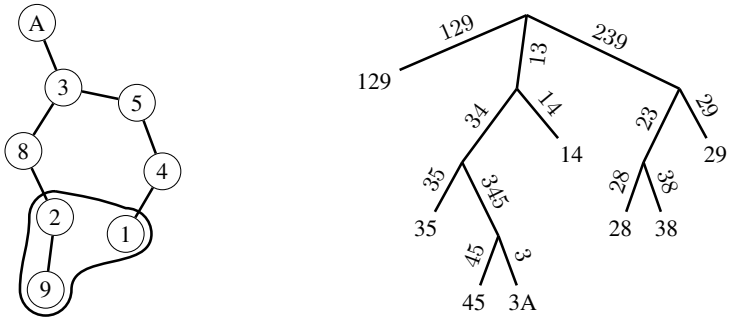


Fig. 1: A hypergraph H (left) and an optimal branch decomposition (T, γ) of H (right). The labels of the leaves of T are the edges assigned to them by γ and the labels of the edges of T are the cut vertices of that edge.

A *branch decomposition* $\mathcal{B}(H)$ of H is a pair (T, γ), where T is a ternary tree and $\gamma : L(T) \to E(H)$ is a bijection between the edges of H and the leaves of T (denoted by $L(T)$). For simplicity, we write $\gamma(L)$ to denote the set $\{\, \gamma(l) \mid l \in L \,\}$ for a set of leaves L of T and we also write $\delta(T')$ instead of $\delta(\gamma(L(T')))$ for a subtree T of T'. For an edge e of T, we denote by $\delta_{\mathcal{B}}(e)$ (or simply $\delta(e)$ if $\mathcal{B}$ is clear from the context), the set of *cut vertices* of e, i.e., the set $\delta(T')$, where T' is any of the two components of $T \setminus \{e\}$. Observe that $\delta_{\mathcal{B}}(e)$ consists of the set of all vertices v such that there are two leaves l_1 and l_2 of T in distinct components of $T \setminus \{e\}$ such that $v \in \gamma(l_1) \cap \gamma(l_2)$. The *width* of an edge e of T is the number of cut vertices of e, i.e., $|\delta_{\mathcal{B}}(e)|$ and the *width* of $\mathcal{B}$ is the maximum width of any edge of T. The *branchwidth* bw(H) of H is the minimum width over all branch decompositions of H (or 0 if $|E(G)| = 0$ and H has no branch decomposition). We also define the *depth* of $\mathcal{B}$ as the radius of T. Fig. 1 illustrates a branch decomposition of a small hypergraph. In the figure and in the remainder of the paper we will often denote a set $\{1, 2, 3, A\}$ of vertices as $123A$. We will use the following property of branch decompositions.

Proposition 1. *Let $\mathcal{B} := (T, \gamma)$ and $\mathcal{B}' := (T', \gamma')$ be two branch decompositions of the same hypergraph H. Then there is bijection $\alpha : V(T) \to V(T')$ between the vertices of T such that $l \in L(T)$ if and only if $\alpha(l) \in L(T')$ and moreover $\gamma(l) = \gamma'(\alpha(l))$ for every $l \in L(T)$. In other words w.l.o.g. one can assume that $\mathcal{B}$ and $\mathcal{B}'$ differ only in terms of the edges of T and T'.*

Technical Report AC-TR-16-004

Partitions As partitions play an important role in our reformulation of branchwidth, we recall some basic terminology. A *partition* of a set S is a set P of nonempty subsets of S such that any two sets in P are disjoint and S is the union of all sets in P. The elements of P are called *equivalence classes*. Let P, P' be partitions of S. Then P' is a *refinement* of P if for any two elements $x, y \in S$ that are in the same equivalence class of P' are also in the same equivalence class of P (this entails the case $P = P'$). Moreover, we say that P' is a *k-ary refinement* of P if additionally it holds that for every $p \in P$ there are $p_1, \ldots, p_k$ in P' such that $p = \bigcup_{1 \leq i \leq k} p_i$.

3 Partition-based Reformulation of Branchwidth

One might be tempted to think that the original characterization of branch decompositions as ternary trees leads to a very natural and efficient SAT encoding for the existence of a branch decomposition of a certain width. In particular, in the light of Proposition 1 one could encode the branch decomposition as a formlula by fixing all vertices of the tree (as well as the bijection on the leaves) and then employing variables to guess the children for each inner vertex of the tree. We have tried this approach, however, to our surprise the performance of the encoding based on this characterization of branch decomposition was very poor. We therefore opted to develop a different encoding based on a new partition-based characterization of branch decomposition which we will introduce next. Compared to this, the original encoding was clearly inferior, resulting in an encoding size that was always at least twice as large and overall solving times that where longer by a factor of 3-10, even after several rounds of fine-tuning and experimenting with natural variants.

Let H be a hypergraph. A *derivation* $\mathcal{P}$ of H of *length* l is a sequence $(P_1, \ldots, P_l)$ of partitions of $E(G)$ such that:

D1 $P_1 = \{\, \{e\} \mid e \in E(H) \,\}$ and $P_l = \{E(H)\}$ and
D2 for every $i \in \{1, \ldots, l-1\}$, P_i is a 2-ary refinement of P_{i+1} and
D3 P_{l-1} is a 3-ary refinement of P_l.

The *width* of $\mathcal{P}$ is the maximum size of $\delta_H(E)$ over all sets $E \in \bigcup_{1 \leq i < l} P_i$. We will refer to P_i as the i-th *level* of the derivation $\mathcal{P}$ and we will refer to elements in $\bigcup_{1 \leq i \leq l} P_i$ as *sets* of the derivation. We will show that any branch decomposition can be transformed into a derivation of the same width and also the other way around. The following example illustrates the close connection between branch decompositions and derivations.

Example 1. Consider the branch decomposition $\mathcal{B}$ given in Fig. 1. Then $\mathcal{B}$ can, e.g., be translated into the derivation $\mathcal{P} = (P_1, \ldots, P_5)$ defined by:

$$P_1 = \Big\{\{129\}, \{35\}, \{45\}, \{3A\}, \{14\}, \{28\}, \{38\}, \{29\}\Big\}$$
$$P_2 = \Big\{\{129\}, \{35\}, \{45, 3A\}, \{14\}, \{28\}, \{38\}, \{29\}\Big\}$$
$$P_3 = \Big\{\{129\}, \{35, 45, 3A\}, \{14\}, \{28, 38\}, \{29\}\Big\}$$
$$P_4 = \Big\{\{129\}, \{35, 45, 3A, 14\}, \{28, 38, 29\}\Big\}$$

Technical Report AC-TR-16-004

$$P_5 = \Big\{\big\{129, 35, 45, 3A, 14, 28, 38, 29\big\}\Big\}$$

The width of $\mathcal{B}$ is equal to the width of $\mathcal{P}$.

The following theorem shows that derivations provide an alternative characterization of branch decompositions.

Theorem 1. *Let H be a hypergraph and w and d two integers. H has a branch decomposition of width at most w and depth at most d if and only if H has a derivation of width at most w and length at most d.*

One important parameter influencing the size of the encoding for the existence of a derivation is the length of the derivation. The next theorem shows a tight upper bound on the length of any derivation required to rule out the existence of a branch decomposition. Observe that a simple caterpillar (i.e., a path where each inner vertex has one additional "pending" neighbor) shows that the bound given below is tight. The main observations behind the following theorem are that every branch decomposition has depth at most $\lfloor |E(H)|/2 \rfloor$ and moreover for certain branch decompositions one can further reduce its depth by replacing subtrees at the bottom of the branch decomposition containing at most $\lceil w/e \rceil$ leaves with complete binary subtrees of height at most $\lceil \log \lfloor w/e \rfloor \rceil$.

Theorem 2. *Let H be a hypergraph, e the maximum size over all edges of H, and w an integer. Then the branchwidth of H is at most w if and only if H has a derivation of width at most w and length at most $\lfloor |E(H)|/2 \rfloor - \lceil w/e \rceil + \lceil \log \lfloor w/e \rfloor \rceil$.*

4 Encoding

Let H be a hypergraph with m edges and n vertices, and let w and d be positive integers. We will assume that the vertices of H are represented by the numbers from 1 to n and the edges of H by the numbers from 1 to m. The aim of this section is to construct a formula $F(H, w, d)$ that is satisfiable if and only if H has derivation of width at most w and length at most d. Because of Theorem 2 (after setting d to the value specified in the theorem) it holds that $F(H, w, d)$ is satisfiable if and only if H has branchwidth at most w. To achieve this aim we first construct a formula $F(H, d)$ that is satisfiable if and only if H has a derivation of length at most d and then we extend this formula by adding constrains that restrict the width of the derivation to w.

4.1 Encoding of a Derivation of a Hypergraph

The formula $F(H, d)$ uses the following variables. A *set variable* $\mathrm{s}(e, f, i)$, for every $e, f \in E(H)$ with $e < f$ and every i with $0 \leq i \leq d$. Informally, $\mathrm{s}(e, f, i)$ is true whenever e and f are contained in the same set at level i of the derivation. A *leader variable* $\mathrm{l}(e, i)$, for every $e \in E(H)$ and every i with $0 \leq i \leq d$. Informally, the leader variables will be used to uniquely identify the sets at each level of a derivation, i.e., $\mathrm{l}(e, i)$ is true whenever e is the smallest edge in a set at level i of the derivation.

We now describe the clauses of the formula. The following clauses ensure (D1) and that the derivation is a sequence of refinements.

Technical Report AC-TR-16-004

$$(\neg \mathrm{s}(e,f,0)) \wedge (\mathrm{s}(e,f,d)) \wedge (\neg \mathrm{s}(e,f,i) \vee \mathrm{s}(e,f,i+1)) \quad \text{for } e,f \in E(H), e < f, 1 \leq i < d$$

The following clauses ensure that the relation of being in the same set is transitive.

$$\begin{aligned}&(\neg \mathrm{s}(e,f,i) \vee \neg \mathrm{s}(e,g,i) \vee \mathrm{s}(f,g,i))\\ &\wedge(\neg \mathrm{s}(e,f,i) \vee \neg \mathrm{s}(f,g,i) \vee \mathrm{s}(e,g,i))\\ &\wedge(\neg \mathrm{s}(e,g,i) \vee \neg \mathrm{s}(f,g,i) \vee \mathrm{s}(e,f,i)) \quad \text{for } e,f,g \in E(H), e < f < g, 1 \leq i \leq d\end{aligned}$$

The following clauses ensure that $\mathrm{l}(e,i)$ is true if and only if e is the smallest edge contained in some set at level i of a derivation.

$$\underbrace{(\mathrm{l}(e,i) \vee \bigvee_{f \in E(H), f<e} \mathrm{s}(f,e,i))}_{A} \wedge \underbrace{\bigwedge_{f \in E(H), f<e} (\neg \mathrm{l}(e,i) \vee \neg \mathrm{s}(f,e,i))}_{B} \quad \text{for } e \in E(H), 1 \leq i \leq d$$

Part A ensures that e is a leader or it is in a set with an edge which is smaller than e and the part B ensures that if e is not in same set with any smaller edge then it is a leader. The following clauses ensure that at most two sets in the partition at level i can be combined into a set in the partition at level $i+1$, i.e., together with the clauses above it ensures (D2).

$$\neg \mathrm{l}(e,i) \vee \neg \mathrm{l}(f,i) \vee \neg \mathrm{s}(e,f,i+1) \vee \mathrm{l}(e,i+1) \vee \mathrm{l}(f,i+1) \quad \text{for } e,f \in E(H), e < f, 1 \leq i < d-1$$

The following clauses ensure that at most three sets in the partition at level $d-1$ can be combined into a set in the partition at level d , i.e., together with the clauses above it ensures (D3).

$$\begin{aligned}&\neg \mathrm{l}(e,d-1) \vee \neg \mathrm{l}(f,d-1) \vee \neg \mathrm{l}(g,d-1) \vee \neg \mathrm{s}(e,f,d) \vee \neg \mathrm{s}(e,g,d)\\ &\vee \mathrm{l}(e,d) \vee \mathrm{l}(f,d) \vee \mathrm{l}(g,d) \quad \text{for } e,f,g \in E(H), e < f < g\end{aligned}$$

All of the above clauses together ensure (D1), (D2), and (D3). We also add the following redundant clauses.

$$\mathrm{l}(e,i) \vee \neg \mathrm{l}(e,i+1) \quad \text{for } e \in E(H), 1 \leq i < d$$

These clauses use the observation that if an edge is not a leader at level i then it cannot be a leader at level $i+1$. The formula $F(H,d)$ contains at most $\mathcal{O}(m^2 d)$ variables and $\mathcal{O}(m^3 d)$ clauses.

4.2 Encoding of a Derivation of Bounded Width

Next we describe how $F(H,d)$ can be extended to restrict the width of the derivation. The main idea is to first identify the set of cut vertices for the sets in the derivation and then restrict their sizes. To this end we first need to introduce new variables (and later clauses), which allow us to identify cut vertices of edge sets in the derivation. In particular, we introduce a *cut variable* $\mathrm{c}(e,u,i)$ for every $e \in E(H)$, $u \in V(H)$ and i with $1 \leq i \leq d$. Informally, $\mathrm{c}(e,u,i)$ is true if u is a cut vertex of the set containing e at level i of the derivation. In order to restrict the size of the sets of cut vertices later on we do not need the reverse direction of the previous statement. Recall that a vertex u is a cut vertex for some set p of the derivation if there are two distinct edges incident to u such that one of them is contained in p and the other one is not.

Technical Report AC-TR-16-004

Table 1: An illustration of the behavior of the sequential counter for the case that H has six vertices (labeled from 1 to 6) and $w = 4$. The first column identifies the vertex u, the second column gives the value of the variable $\mathsf{c}(e, u, i)$ for a fixed edge e and a fixed level i and the last four columns give the values of the variables $\#(e, u, i, j)$.

u	$\mathsf{c}(e,u,i)$	j = 1	2	3	4
1	0	0	0	0	0
2	1	1	0	0	0
3	1	1	1	0	0
4	0	1	1	0	0
5	1	1	1	1	0
6	0	1	1	1	0

Defining the Cut Vertices In the following we will present an encoding that has turned out to give the best results in our case. The main idea behind the encoding is to only define the variables $\mathsf{c}(e, u, i)$ for the leading edges e in the current derivation.

The following clauses ensure that whenever two edges incident to a vertex are not in the same set at level i of the derivation, then the vertex is a cut vertex for every leading edge of the sets containing the incident edges.

$$\neg\mathsf{l}(e,i) \vee \mathsf{c}(e,u,i) \vee \mathsf{s}(\min\{e,f\},\max\{e,f\},i) \vee \neg\mathsf{s}(\min\{e,g\},\max\{e,g\},i)$$
$$\text{for } e,f,g \in E(H), e \neq f, e \neq g, u \in V(H), u \in f, u \in g, 1 \leq i \leq d$$

$$\neg\mathsf{l}(e,i) \vee \mathsf{s}(\min\{e,f\},\max\{e,f\},i) \vee \mathsf{c}(e,u,i)$$
$$\text{for } e,f \in E(H), e \neq f, u \in V(H), u \in e, u \in f, 1 \leq i \leq d$$

Additionally, we add the following redundant clauses that ensure the "monotonicity" of the cut vertices, i.e., if u is a cut vertex for a set at level i and for the corresponding set at level $i + 2$, then it also has to be a cut vertex at level $i + 1$.

$$\neg\mathsf{l}(e,i) \vee \neg\mathsf{l}(e,i+1) \vee \neg\mathsf{l}(e,i+2) \vee \neg\mathsf{c}(e,u,i) \vee \neg\mathsf{c}(e,u,i+2) \vee \mathsf{c}(e,u,i+1))$$
$$\text{for } e \in E(H), u \in V(H), 1 \leq i \leq d-2$$

The definition of cut vertices adds at most $\mathcal{O}(mnd)$ variables and at most $\mathcal{O}(m^3nd)$ clauses.

Restricting the Size of the Cuts Next we describe how to restrict the size of all sets of cut vertices to w and thereby complete the encoding of $F(H, w, d)$. In particular, our aim is to restrict the number of vertices $u \in V(H)$ for which a variable $\mathsf{c}(e, u, i)$ is true for some $e \in E(H)$ and $1 \leq i \leq d$. In this paper we will only present the *sequential counter* approach [18] since this approach has turned out to provide the best results in our setting. We also considered the *order encoding* [11] with less promising results. For the sequential counter, we will introduce a *counter variable* $\#(e, u, i, j)$ for every $e \in E(H), u \in V(H), 1 \leq i \leq d, 1 \leq j \leq w$.

The idea of the sequential counter is illustrated in Table 1. Informally, $\#(e, u, i, j)$ is true if u is the lexicographically j-th cut vertex of the edge edge e. We need the following clauses.

$$(\neg\#(e,u-1,i,j) \vee \#(e,u,i,j)) \wedge (\neg\mathsf{c}(e,u,i) \vee \neg\#(e,u-1,i,j-1))$$
$$\vee\#(e,u,i,j)) \wedge (\neg\mathsf{c}(e,u,i) \vee \neg\#(e,u-1,i,w))$$
$$\text{for } e \in E(H), 2 \leq u \leq |V(H)|, 1 \leq i \leq d, 1 \leq j \leq w$$

Technical Report AC-TR-16-004

$\neg \mathrm{c}(e,u,i) \vee \#(e,u,i,1)$ for $e \in E(H)$, $1 \leq u \leq |V(H)|$, $1 \leq i \leq d$

This completes the construction of the formula $F(H,w,d)$. In total $F(H,w,d)$ has at most $\mathcal{O}(m^2d{+}mndw) \subseteq \mathcal{O}(m^3{+}m^2n^2)$ variables and at most $\mathcal{O}(m^3nd{+}mndw) \subseteq \mathcal{O}(m^4n + m^2n^2)$ clauses. By construction, $F(H,w,d)$ is satisfiable if and only H has a derivation of width at most w and length at most d. Because of Theorem 1, we obtain:

Theorem 3. *The formula $F(H,w,d)$ is satisfiable if and only if H has a branch decomposition of width at most w and depth at most d. Moreover, a corresponding branch decomposition can be constructed from a satisfying assignment of $F(H,w,d)$ in linear time in terms of the number of variables of $F(H,w,d)$.*

5 Local Improvement

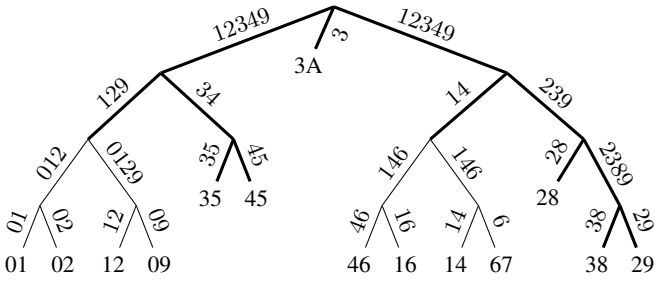


Fig. 2: A branch decomposition $\mathcal{B}$ of the graph H given in Fig. 3 together with an example of a local branch decomposition $\mathcal{B}_L$ (highlighted by thicker edges) chosen by our algorithm.

The encoding presented in the previous section allows us to compute the exact branchwidth of hypergraphs up to a certain size. Due to the instinct difficulty of the problem one can hardly hope to go much further beyond this size barrier with an exact method. In this section we therefore propose a local improvement approach that employs our SAT encoding to improve small parts of an heuristically obtained branch decomposition. Our local improvement procedure can be seen as a kind of local search procedure that at each step tries to replace a part of the branch decomposition with an better one found by means of the SAT encoding and repeats this process until a fixed-point (or timeout) is reached.

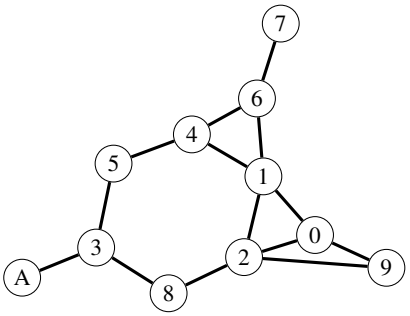


Fig. 3: The graph H used to illustrate the main idea behind our local improvement procedure.

Technical Report AC-TR-16-004

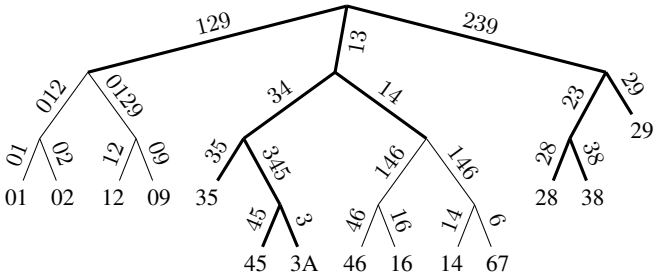


Fig. 4: The improved branch decomposition $\mathcal{B}'$ obtained from $\mathcal{B}$ after replacing the local branch decomposition $\mathcal{B}_L$ of $H(T_L)$ with an optimal branch decomposition $\mathcal{B}'_L$ of $H(T_L)$ obtained from our SAT encoding. See Fig. 2 for an illustration of $\mathcal{B}$ and $\mathcal{B}_L$.

Let H be a hypergraph and $\mathcal{B} := (T, \gamma)$ a branch decomposition of H. For a connected ternary subtree T_L of T we define the *local branch decomposition* $\mathcal{B}_L := (T_L, \gamma_L)$ of $\mathcal{B}$ by setting $\gamma_L(l) = \delta_{\mathcal{B}}(e)$ for every leaf $l \in L(T_L)$, where e is the (unique) edge incident to l in T_L. We also define the hypergraph $H(T_L)$ as the hypergraph that has one (hyper-)edge $\gamma_L(l)$ for every leaf l of T_L and whose vertices are defined as the union of all these edges. We observe that $\mathcal{B}_L$ is a branch decomposition of $H(T_L)$. The main idea behind our approach, which we will formalize below, is that we can obtain a new branch decomposition of H by replacing the part of $\mathcal{B}$ formed by $\mathcal{B}_L$ with any branch decomposition of $H(T_L)$. In particular, by replacing $\mathcal{B}_L$ with a branch decomposition of $H(T_L)$ of lower width, we will potentially improve the branch decomposition $\mathcal{B}$. This idea is illustrated in Fig. 2 and Fig. 4.

input : A hypergraph H
output: A branch decomposition of H

$\mathcal{B} \leftarrow$ `BDHeuristic(`H`)` // ($\mathcal{B} := (T, \gamma)$)
`improved` $\leftarrow$ *true*
while `improved` **do**
  $M \leftarrow$ "the set of edges e of $\mathcal{B}$ whose width ($|\delta_{\mathcal{B}}(e)|$) is maximum"
  $\mathcal{C} \leftarrow$ "the set of components of $T[M]$"
  `improved` $\leftarrow$ *false*
  for $C \in \mathcal{C}$ **do**
    $\mathcal{B}_L \leftarrow$ `LocalBD` ($\mathcal{B}$, C)
    $\mathcal{B}'_L \leftarrow$ `ImproveLD` ($\mathcal{B}_L$)
    if $\mathcal{B}'_L \neq$ NULL **then**
      $\mathcal{B} \leftarrow$ `Replace` ($\mathcal{B}, \mathcal{B}_L, \mathcal{B}'_L$)
      `improved` $\leftarrow$ *true*
    else
      break

Algorithm 1: Local Improvement

input : A branch decomposition $\mathcal{B} := (T, \gamma)$ of H and a branch decomposition $\mathcal{B}_L := (T_L, \gamma_L)$ of $H(T_L)$
output: An "improved" branch decomposition of $H(T_L)$

if $|T_L| >$`globalbudget` **then**
  return *NULL*
$w \leftarrow$ "the width of $\mathcal{B}_L$"
repeat
  $\mathcal{B}_D \leftarrow$ `SATSolve` ($H(T_L)$, w)
  if $\mathcal{B}_D \neq$ NULL **then**
    $\mathcal{B}'_L \leftarrow \mathcal{B}_D$
  $w \leftarrow w - 1$
until $\mathcal{B}_D ==$ NULL
return $\mathcal{B}'_L$

Algorithm 2: ImproveLD

A general outline of our algorithm is given in Algorithm 1. The algorithm uses two global parameters: `globalbudget` gives an upper bound on the size of the lo-

Technical Report AC-TR-16-004

input : A branch decomposition $\mathcal{B} := (T, \gamma)$ of H and a component C of T
output: A local branch decomposition of $\mathcal{B}$

1 $w \leftarrow$ "the width of $\mathcal{B}$"
2 $T_L \leftarrow C$
3 **for** $c \in V(C)$ *with* $\deg_C(c) = 2$ **do**
4 | "add the unique third neighbor and its edge incident to c to T_L"
5 $Q \leftarrow$ "the set of leaves of T_L"
6 **while** $Q \neq \emptyset$ *and* $|T_L| \leq$`globalbudget` -2 **do**
7 | $l \leftarrow Q$.pop()
8 | **if** *"l is not a leaf of T"* **then**
9 | | $c, c' \leftarrow$ "the two neighbors of l in T which are not neighbors of l in T_L"
10 | | **if** $\delta_{\mathcal{B}}(\{l, c\}) < w$ *and* $\delta_{\mathcal{B}}(\{l, c'\}) < w$ **then**
11 | | | "add c and c' together with their edges incident to l to T_L"
12 | | | Q.push(c)
13 | | | Q.push(c')
14 **return** "the local branch decomposition of $\mathcal{B}$ represented by T_L"

Algorithm 3: Local Selection (LocalBD)

Technical Report AC-TR-16-004

cal branch decomposition and the function `length(H,w)`, which is only used by the function `SATSolve` explained below, provides an upper bound on the length of a derivation which will be considered by our SAT encoding.

Given a hypergraph H, the algorithm first computes a (not necessarily optimal) branch decomposition $\mathcal{B} := (T, \gamma)$ of H using, e.g., the heuristics from [6, 13]. The algorithm then computes the set M of maximum cut edges of T, i.e., the set of edges e of T with $|\delta(e)| = w$, where w is the width of $\mathcal{B}$. It then computes the set $\mathcal{C}$ of components of $T[M]$ and for every component $C \in \mathcal{C}$ it calls the function `LocalBD` to obtain a local branch decomposition $\mathcal{B}_L := (T_L, \gamma_L)$ of $\mathcal{B}$, which contains (at least) all the edges of C. The function `LocalBD` is given in Algorithm 3 and will be described later. Given $\mathcal{B}_L$ the algorithm tries to compute a branch decomposition $\mathcal{B}'_L := (T'_L, \gamma'_L)$ of $H(T_L)$ with smaller width than $\mathcal{B}_L$ using the function `ImproveLD`, which is described later. If successful, the algorithm updates $\mathcal{B}$ by replacing the part of $\mathcal{B}$ represented by T_L with $\mathcal{B}'_L$ according to Theorem 4 and proceeds with line 4. If on the other hand $\mathcal{B}_L$ cannot be improved, the algorithm proceeds with the next component C of $T[M]$. This process is repeated until none of the components C of $T[M]$ lead to an improvement.

The function `LocalBD`, which is given in Algorithm 3, computes a local branch decomposition $\mathcal{B}_L := (T_L, \gamma_L)$ of $\mathcal{B}$ that contains at least all edges in the component C and which should be small enough to ensure solvability found by our SAT encoding as follows. In the beginning T_L is set to the connected ternary subtree of T obtained from $T[C]$ after adding the (unique) third neighbor of any vertex v of C that has degree exactly two in $T[C]$. It then proceeds by processing the (current) leaves of T_L in a breadth first search manner, i.e., in the beginning all the leaves of T_L are put in a first-in first-out queue Q. If l is the current leaf of T_L, which is not a leaf of T, the algorithm adds the two additional neighbors of l in T to T_L and adds them to Q. It proceeds in this manner until the number of edges in T_L does exceed the global budget.

The function `ImproveLD` tries to compute a branch decomposition of $H(T_L)$ with lower width than $\mathcal{B}_L$ using our SAT encoding. In particular, if the size of T_L does not exceed the global budget (in which case it would be highly unlikely that a lower width branch decomposition can be found using our SAT encoding), the function calls the function `SATSolve` with decreasing widths w until `SATSolve` does not return a branch decomposition any more. Here, the function `SATSolve` uses the formula $F(H(T_L), w, d)$ from Theorem 3 with d set to `length(H,w)` to test whether $H(T_L)$ has a branch decomposition of width at most w and depth at most d. If so (and if the SAT-solver solves the formula within a predefined timeout) `SATSolve` returns the corresponding branch decomposition; otherwise it returns NULL.

Last but not least the function `Replace` replaces the part of $\mathcal{B}$ represented by $\mathcal{B}_L$ with the new branch decomposition $\mathcal{B}'_L$ according to Theorem 4.

Let H be a hypergraph, $\mathcal{B} := (T, \gamma)$ a branch decomposition of H, T_L a connected ternary subtree of T, $\mathcal{B}_L := (T_L, \gamma_L)$ be the local branch decomposition of $\mathcal{B}$ corresponding to T_L, and let $\mathcal{B}'_L := (T'_L, \gamma')$ be any branch decomposition of $H(T_L)$. Note that because $\mathcal{B}_L$ and $\mathcal{B}'_L$ are branch decompositions of the same hypergraph $H(T_L)$, we obtain from Proposition 1 that we can assume that $V(T_L) = V(T'_L)$ and $\gamma = \gamma'$. We define the *locally improved* branch decomposition, denoted by $\mathcal{B}(\frac{\mathcal{B}_L}{\mathcal{B}'_L})$, to be the branch decomposition obtained from $\mathcal{B}$ by replacing the part corresponding to $\mathcal{B}_L$ with $\mathcal{B}'_L$, i.e., the tree of $\mathcal{B}'$ is obtained from T by removing all edges of T_L from T and replacing them with the edges of T'_L and the bijection of $\mathcal{B}'$ is equal to γ.

Theorem 4. *$\mathcal{B}(\frac{\mathcal{B}_L}{\mathcal{B}'_L})$ is a branch decomposition of H, whose width is the maximum the width of $\mathcal{B}'_L$ and maximum width over all edges $e \in E(T) \setminus E(T_L)$ in $\mathcal{B}$.*

6 Experimental Results

We have implemented the single SAT encoding and the SAT-based local improvement method and tested them on various benchmark instances, including famous named graphs from the literature [21], graphs from TreewidthLIB [5] which origin from a broad range of applications, and a series of circular clusters [7] which are hypergraphs denoted C_v^e with v vertices and v edges of size e. Throughout we used the SAT-solver Glucose 4.0 (with standard parameter setting) as it performed best in our initial tests compared to other solvers such as GlueMiniSat 2.2.8, Lingeling, and Riss 4.27. We run the experiments on a 4-core Intel Xeon CPU E5649, 2.35GHz, 72 GB RAM machine with Ubuntu 14.04 with each process having access to at most 8 GB RAM.

6.1 Single SAT Encoding

To determine the branchwidth of a graph or hypergraph with our encoding, one could either start from $w = 1$ and increase w until the formula becomes satisfiable, or by setting w to an upper bound on the branchwidth obtained by a heuristic method, and decrease it until the formula becomes unsatisfiable. For both approaches the solving time at the threshold (i.e., for the largest w for which the formula is unsatisfiable) is, as one would expect, by far the longest. Table 2 shows this behavior on some typical

Technical Report AC-TR-16-004

Table 2: Distribution of solving time in seconds for various values of w for some famous named graphs of branchwidth 6.

w	2	3	4	5	6	7	8	9	10
Graph	unsat	unsat	unsat	unsat	sat	sat	sat	sat	sat
Kittell	19.5	87.6	400.8	204.7	103.3	40.4	22.5	18.5	11.2
Errera	5.7	22.7	79.4	1530.9	12.0	7.3	6.7	5.4	6.1
Folkman	3.4	13.7	98.6	2747.0	6.1	5.3	3.7	3.8	5.2
Poussin	3.3	9.2	68.7	941.2	4.5	3.5	3.9	2.9	3.4

Table 3: Exact branchwidth w of famous named graphs known from the literature. Column d indicates the smallest depth of a branch decomposition of width w for the considered graph.

Graph	$\|V\|$	$\|E\|$	w	d	Graph	$\|V\|$	$\|E\|$	w	d	Graph	$\|V\|$	$\|E\|$	w	d
Watsin	50	75	6	8	McGee	24	36	7	7	Dürer	12	18	4	6
Kittell	23	63	6	8	Nauru	24	36	6	7	Franklin	12	18	4	6
Holt	27	54	9	9	Hoffman	16	32	6	6	Frucht	12	18	3	6
Shrikhande	16	48	8	7	Desargues	20	30	6	6	Herschel	11	18	4	6
Errera	17	45	6	7	Dodecahedron	20	30	6	6	Tietze	12	18	4	6
Brinkmann	21	42	8	7	Flower Snark	20	30	6	6	Petersen	10	15	4	6
Clebsch	16	40	8	7	Goldner-Harary	11	27	4	6	Pmin	9	12	3	5
Folkman	20	40	6	7	Pappus	18	27	6	6	Wagner	8	12	4	5
Paley13	13	39	7	7	Sousselier	16	27	5	6	Moser spindle	7	11	3	6
Poussin	15	39	6	7	Chvátal	12	24	6	6	Prism	6	9	3	5
Robertson	19	38	8	7	Grötzsch	11	20	5	6	Butterfly	5	6	2	3

Technical Report AC-TR-16-004

instances. Hence whether we determine the branchwidth from below or from above does not matter much. A more elaborate binary search strategy could save some time, but overall the expected gain is little compared to the solving time at the threshold. The size of the encoding is manageable for graphs and hypergraphs for up to about 100 edges. The solving time varies and depends on the structure of the (hyper)graph. We could determine the exact branchwidth of many famous graphs known from the literature, see Table 3. For many of the graphs the exact branchwidth has not been known before. We also tested the encoding on circular cluster hypergraphs C^i_{2i-1}. We were able to solve instances up to the hypergraph C^{26}_{51}, for which we established a branchwidth of 42.

6.2 SAT-Based Local Improvement

We tested our local improvement method on graphs with several thousands of vertices and edges and with initial branch decomposition of width over 200. In particular, we tested it on all graphs from TreewidthLIB omitting graphs that are minors from other

Table 4: Results for SAT-based local improvement for instances from TreewidthLIB. Column iw gives the width of the initial branch decomposition, w the width of the branch decomposition obtained by local improvement.

Graph	$\|V\|$	$\|E\|$	iw	w
inithx.i.2-pp	363	8897	55	45
fpsol2.i.2-pp	333	7910	48	39
fpsol2.i.3-pp	333	7907	48	39
graph13	458	1877	141	134
fpsol2.i.3	425	8688	53	48
graph13pp-pp	374	1722	133	128
graph09-pp	405	1525	128	123
bn_31-pp	1148	3317	40	36
bn_4	100	574	42	38
celar08-pp-003	76	421	20	16
celar08pp-pp.dgf-034	76	421	20	16
celar09-pp-002	76	421	20	16

Graph	$\|V\|$	$\|E\|$	iw	w
celar10-pp-002	76	421	20	16
fpsol2.i.2	451	8691	53	49
graph05-wpp	94	397	28	24
graph04-pp	179	678	52	48
nrw1379.tsp	1379	4115	42	38
nrw1379.tsp-pp	1367	4081	42	38
bn_36	1444	4181	45	42
inithx.i.1-pp	317	12720	68	65
u724.tsp	724	2117	29	26
water-wpp	22	96	11	8
celar05-pp	80	426	18	15
mulsol.i.2-pp	116	2468	62	59

Technical Report AC-TR-16-004

graphs as well as small graphs with 80 or fewer edges (small graphs can be solved with the single SAT encoding). These are in total 684 graphs with up to 5934 vertices and 17770 edges. We ran our SAT-based local improvement algorithm on each graph with a timeout of 6 hours, where each SAT-call had a timeout of 600 seconds. We used a global budget of 80 and set the depth to 0.6 times the upper bound provided by Theorem 2. We computed the initial branch decomposition by a greedy heuristic kindly provided to us by Hicks [12].

From the 684 graphs, our SAT-based local improvement algorithm could improve the width of the initial branch decomposition for 290 graphs. In some cases the improvement was significant. Table 4 shows the graphs with the best improvement.

6.3 Discussion

As discussed earlier, we are aware of only two implemented algorithms that determine the exact branchwidth of a graph or hypergraph: Hick's combinatorial algorithm based on tangles [12], and Ulusal's integer programming encoding [20]. Since neither of the two implementations are available to the public, we were not able to provide an up-to-date comparision with their approaches but instead performed the comparision with respect to the results stated in the papers. It should therefore be taken into account that hardware and software improved since the time their results were obtained. As Ulusal [20] reports, the integer programming encoding could not solve hypergraphs with more than 13 edges, for instance, it could only solve the circular clusters C^i_{2i-1} up to $i = 7$, whereas we could go up to $i = 26$.

Because of the high branchwidth of these hypergraphs, they are also far out of reach for the tangles-based algorithm. On small graphs the tangles-based algorithm and our SAT encoding perform similarly. For very small branchwidth the tangles-based algorithm can deal with larger graphs (according to [12]) whereas our SAT encoding can

deal with graphs with larger branchwidth. These differences in scalability can be explained by the space requirements of the two approaches: The tangles-based algorithm requires space that is exponential in the branchwidth, whereas our SAT encoding requires polynomial space that depends only linearly on the branchwidth.

Our experiments show that the SAT-based local improvement approach scales well to large graphs with several thousands of vertices and edges and branchwidth upper bounds well over hundred. These are instances that are by far out of reach for any known exact method, in particular, for the tangles-based algorithm which cannot handle large branchwidth. The use of our SAT encoding which scales well with the branchwidth is therefore essential for these instances.

Our results on TreewidthLIB instances show that in some cases the obtained improvement can make a difference of whether a dynamic programming algorithm that uses the obtained branch decomposition is feasible or not.

7 Final Remarks

We have presented a first SAT encoding for branchwidth and introduced the new method of SAT-based local improvements for branch decompositions. Both methods are based on a novel partition-based formulation of branch decompositions. Our experiments show that the single encoding outperforms a known integer programming method and performs competitively with the best known combinatorial method. Our SAT-based local improvement method provides the means for scaling the SAT-approach to much larger instances and exhibits a fruitful new application field of SAT solvers.

For both the single SAT encoding and the SAT-based local improvement we see several possibilities for further improvement. For the encoding one can try other ways for stating cardinality constraints and one could apply incremental SAT solving techniques. Further, one could consider alternative encoding techniques based on MaxSAT, which have been shown effective for related problems [4]. For the local improvement we see various directions for further research. For instance, when a local branch decomposition cannot be improved, one could use the SAT solver to obtain an alternative branch decomposition of the same width but where other parameters are optimized, e.g., the number of maximum cuts. This could propagate into adjacent local improvement steps and yield an overall branch decomposition of smaller width.

Finally we would like to mention that branch decompositions are the basis for several other (hyper)graph width measures such as rankwidth and Boolean-width [1], and we leave the investigation on how our approaches can be extended to these width measures for future research.

Acknowledgement We thank Illya Hicks for providing us the code of his branchwidth heuristics and acknowledge support by the Austrian Science Fund (FWF, projects W1255-N23 and P-27721).

References

1. Isolde Adler, Binh-Minh Bui-Xuan, Yuri Rabinovich, Gabriel Renault, Jan Arne Telle, and Martin Vatshelle. On the boolean-width of a graph: Structure and applications. In Dim-

Technical Report AC-TR-16-004

itrios M. Thilikos, editor, *Graph Theoretic Concepts in Computer Science - 36th International Workshop, WG 2010, Zarós, Crete, Greece, June 28-30, 2010 Revised Papers*, volume 6410 of *Lecture Notes in Computer Science*, pages 159–170, 2010.
2. Michael Alekhnovich and Alexander A. Razborov. Satisfiability, branch-width and Tseitin tautologies. In *Proc. of the 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS'02)*, pages 593–603, 2002.
3. Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi. Algorithms and complexity results for #SAT and Bayesian inference. In *44th Annual IEEE Symposium on Foundations of Computer Science (FOCS'03)*, pages 340–351, 2003.
4. Jeremias Berg and Matti Järvisalo. SAT-based approaches to treewidth computation: An evaluation. In *26th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2014, Limassol, Cyprus, November 10-12, 2014*, pages 328–335. IEEE Computer Society, 2014.
5. Hans Bodlander. TreewidthLIB a benchmark for algorithms for treewidth and related graph problems. http://www.staff.science.uu.nl/~bodla101/treewidthlib/.
6. William Cook and Paul Seymour. Tour merging via branch-decomposition. *INFORMS J. Comput.*, 15(3):233–248, 2003.
7. Gérard Cornuéjols. Combinatorial optimization: Packing and covering. Regional Conference Series in Applied Mathematics. Society for Industrial and Applied Mathematics, Carnegie Mellon University, Pittsburgh, Pennsylvania, 2001.
8. Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer Verlag, New York, 2nd edition, 2000.
9. Fedor V. Fomin, Frédéric Mazoit, and Ioan Todinca. Computing branchwidth via efficient triangulations and blocks. *Discr. Appl. Math.*, 157(12):2726–2736, 2009.
10. Martin Grohe. Logic, graphs, and algorithms. In Jörg Flum, Erich Grädel, and Thomas Wilke, editors, *Logic and Automata: History and Perspectives*, volume 2 of *Texts in Logic and Games*, pages 357–422. Amsterdam University Press, 2008.
11. Marijn Heule and Stefan Szeider. A SAT approach to clique-width. *ACM Trans. Comput. Log.*, 16(3):24, 2015.
12. Illya V. Hicks. Graphs, branchwidth, and tangles! Oh my! *Networks*, 45(2):55–60, 2005.
13. I.V. Hicks. Branchwidth heuristics. *Congr. Numer.*, 159:31–50, 2002.
14. Petr Hliněný and Sang-il Oum. Finding branch-decompositions and rank-decompositions. *SIAM J. Comput.*, 38(3):1012–1032, 2008.
15. Kalev Kask, Andrew Gelfand, Lars Otten, and Rina Dechter. Pushing the power of stochastic greedy ordering schemes for inference in graphical models. In Wolfram Burgard and Dan Roth, editors, *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*. AAAI Press, 2011.
16. Arnold Overwijk, Eelko Penninkx, and Hans L. Bodlaender. A local search algorithm for branchwidth. In Ivana Cerná, Tibor Gyimóthy, Juraj Hromkovic, Keith G. Jeffery, Rastislav Královic, Marko Vukolic, and Stefan Wolf, editors, *SOFSEM 2011: Theory and Practice of Computer Science - 37th Conference on Current Trends in Theory and Practice of Computer Science, Nový Smokovec, Slovakia, January 22-28, 2011. Proceedings*, volume 6543 of *Lecture Notes in Computer Science*, pages 444–454. Springer, 2011.
17. Neil Robertson and P. D. Seymour. Graph minors X. Obstructions to tree-decomposition. *J. Combin. Theory Ser. B*, 52(2):153–190, 1991.
18. Marko Samer and Helmut Veith. Encoding treewidth into SAT. In *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, volume 5584 of *Lecture Notes in Computer Science*, pages 45–50. Springer Verlag, 2009.
19. P. D. Seymour and R. Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994.

Technical Report AC-TR-16-004

20. Elif Ulusal. *Integer Programming Models for the Branchwidth Problem*. PhD thesis, Texas A&M University, May 2008.
21. Eric Weisstein. MathWorld online mathematics resource. http://mathworld.wolfram.com.

Technical Report AC-TR-16-004