A SAT Approach to Branchwidth

NEHA LODHA, SEBASTIAN ORDYNIAK, and STEFAN SZEIDER, Algorithms and Complexity Group, Faculty of Informatics, TU Wien, Vienna, Austria

Branch decomposition is a prominent method for structurally decomposing a graph, a hypergraph, or a propositional formula in conjunctive normal form. The width of a branch decomposition provides a measure of how well the object is decomposed. For many applications, it is crucial to computing a branch decomposition whose width is as small as possible. We propose an approach based on Boolean Satisfiability (SAT) to finding branch decompositions of small width. The core of our approach is an efficient SAT encoding that determines with a single SAT-call whether a given hypergraph admits a branch decomposition of a certain width. For our encoding, we propose a natural partition-based characterization of branch decompositions. The encoding size imposes a limit on the size of the given hypergraph. To break through this barrier and to scale the SAT approach to larger instances, we develop 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 SAT-based local improvement method scales now to instances with several thousands of vertices and edges.

CCS Concepts: • **Mathematics of computing** → *Graph theory*; *Hypergraphs*; *Graph algorithms*; • **Theory of computation** → *Constraint and logic programming*;

Additional Key Words and Phrases: Branchwidth, carving-width, SAT encoding, heuristic search

ACM Reference format:
Neha Lodha, Sebastian Ordyniak, and Stefan Szeider. 2019. A SAT Approach to Branchwidth. *ACM Trans. Comput. Logic* 20, 3, Article 15 (May 2019), 24 pages.
https://doi.org/10.1145/3326159

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 (1991a) 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 formulas in conjunctive normal form (CNF) (Robertson and Seymour 1991b, Oztok and Darwiche 2014). The width of a branch decomposition provides a measure of how well it decomposes the

Dedicated to the memory of Helmut Veith (1971–2016).

A preliminary and shortened version appeared in the proceedings of SAT'16, The 19th International Conference on Theory and Applications of Satisfiability Testing.

We acknowledge support by the Austrian Science Fund (FWF, projects W1255-N23 and P-27721).

Authors' addresses: N. Lodha, S. Ordyniak, and S. Szeider, Algorithms and Complexity Group, Faculty of Informatics, TU Wien, Institute of Logic and Computation, Favoritenstraße 9-11, E192-01, Stiege 2, 4. Stock, A-1040 Wien, Austria; emails: {neha, ordyniak, sz}@ac.tuwien.ac.at.

1529-3785/2019/05-ART15 $15.00
https://doi.org/10.1145/3326159

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 (Cook and Seymour 2003), the #P-complete problem of propositional model counting (Bacchus et al. 2003), and the generation of resolution refutations for unsatisfiable CNF formulas (Alekhnovich and Razborov 2002). Branch decompositions also form the basis of several width-parameters employed in Knowledge Compilation and Reasoning (Darwiche 2009), where they are known as dtrees. 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 (Courcelle 1990; Grohe 2008).

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 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 (Kask et al. 2011):

> *[...] 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 (2003) mention a threshold of 20 for the traveling salesman problem. Today one might consider a higher threshold. Computing an optimal branch decomposition, i.e., a branch decomposition of minimum width, is NP-hard (Seymour and Thomas 1994).

Contribution. In this article, 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 that takes a hypergraph H and an integer w as input and produces a propositional CNF formula that 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 natural *partition-based* characterization of branch decompositions in terms of certain sequences of partitions of the set of edges, similar to the "binary mergings" introduced by Gu and Tamaki (2008). This characterization—together with clauses that express cardinality constraints–gives rise to an efficient SAT encoding that scales up to instances with about a 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, then 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. To break through this barrier, we developed a new *SAT-based local improvement method (SLIM)* 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 method could also be developed for other (hyper) graph width measures.

Encouraged by the good performance of our partition-based encoding for branchwidth, we explored whether a similar encoding can be used for other width parameters. We succeeded to develop a similar encoding for *carving-width*, which is a decompositional parameter closely related to branchwidth (Seymour and Thomas 1994) with applications to graph drawing (Biedl and Vatshelle 2012; Biedl 2014). We will mainly focus on branchwidth and provide a brief description on how similar techniques can be applied to carving-width.

Implementations of our encodings, as well as the local improvement algorithm, are publicly available under https://www.ac.tuwien.ac.at/research/branchlis/.

Related Work. Previously, SAT techniques have been proposed for other graph width measures: Samer and Veith (2009) proposed a SAT encoding for *treewidth*, based on a characterization of treewidth in terms of elimination orderings (that is, the encoding 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 (2014) and Bannach et al. (2017). Heule and Szeider (2015) 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 entirely different notions. Recently our partition-based encoding has inspired similar encodings for special treewidth and pathwidth (Lodha et al. 2017) as well as treecut width and treedepth (Ganian et al. 2019). Moreover, our local improvement approach has been adapted for treewidth (Fichte et al. 2017).

For finding branch decompositions of smallest width, Robertson and Seymour (1991a) suggested an exponential-time algorithm based on the combinatorial notion of *tangles*, which was later implemented by Hicks (2005). Further exponential-time algorithms have been proposed (see, for instance (Fomin et al. 2009; Hliněný and Oum 2008)) but there seem to be no implementations. Ulusal (2008) proposed several encodings to integer linear programming (ILP). In Section 7.1, we provide an experimental comparison of our SAT encoding with the tangles-based algorithm as well as Ulusal's ILP encodings. The evaluation shows that our SAT encoding is superior to Ulusal's ILP encodings, but in most cases inferior to Hicks' tangles-based algorithm. Nevertheless, our SAT encoding provides several advantages over the tangles-based algorithm that are essential for its use within our local improvement procedure:

- The current implementation of the tangles-based algorithm does not support the computation of branchwidth for hypergraphs. Since hypergraphs naturally arise as the local parts in the local improvement procedure, the support for hypergraphs is an essential feature for the use of our encoding within the local improvement procedure. Moreover, even though there is a reduction from hypergraphs to graphs conserving the branchwidth, this reduction increases both the number of vertices and the number edges significantly, which makes our approach more efficient than the tangles-based approach.
- In contrast to the SAT-based approach, the tangles-based approach cannot compute upper bounds for the branchwidth of a (hyper-)graph. Computing upper bounds is however crucial when used inside the local improvement algorithm, as the local hypergraphs are too large to be solved exactly by any known method. In particular, as our experiments show, the local improvement method performs best when the number of hyperedges in the local hypergraphs is around 200.
- As also pointed out by Hicks (2005) the space and time requirements of the tangles-based approach grow exponentially with the branchwidth, hence this approach is not applicable

to (hyper-)graphs with high branchwidth. Such hypergraphs are encountered during local improvement. In contrast, the space requirements of our SAT encoding grows only linearly with the branchwidth.

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% (Robertson and Seymour 1991a), which makes this approach prohibitive in practice. For practical purposes, one therefore mainly resorts to heuristic methods that compute suboptimal branch decompositions (Cook and Seymour 2003; Hicks 2002; Overwijk et al. 2011; Smith et al. 2012).

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 variable. 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 textbook (Diestel 2000). For a hypergraph H, we denote by $V(H)$ the vertex set of H and by $E(H)$ the edge set of H. If $E \subseteq E(H)$, then we denote by $H \setminus E$ the hypergraph with vertices $V(H)$ and edges $E(H) \setminus E$. We denote by $\Delta(H)$ the maximum degree of any vertex in H, i.e., the maximum number of edges containing a particular vertex of H. 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 v such that all other vertices of G can be reached from v 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 $\mathrm{h}(T)$, 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 *separation* 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 from the context) the set of *load vertices* of E in H, i.e., $\delta(E)$ contains all vertices incident to both an edge in E and an edge in $E(H) \setminus E$. The *order* of a separation (given by the edge set E) is the number of load vertices, i.e., $|\delta_H(H)|$. Note that $\delta(E) = \delta(E(H) \setminus E)$.

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 *guard 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

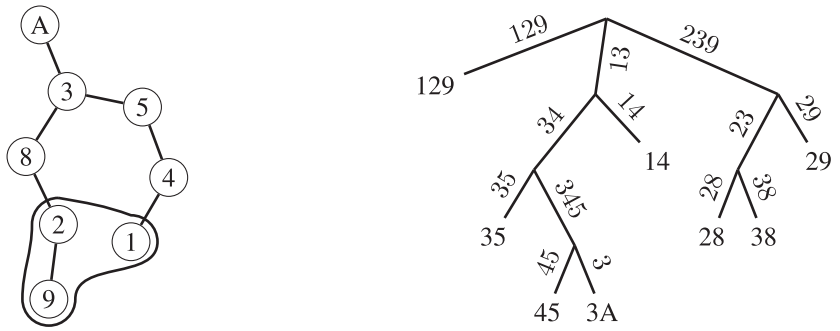

Fig. 1. A hypergraph H (left) and an optimal branch decomposition (T, γ) of H (right). The labels of the leaves of T are the edges assigned to them by γ and the labels of the edges of T are the load vertices of that edge.

edge e of T is the number of load 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* 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. Figure 1 illustrates a branch decomposition of a small hypergraph. In the figure and in the remainder of the article, we will often denote a set $\{1, 2, 3, A\}$ of vertices as $123A$. We will use the following well-known property of branch decompositions (see, e.g., Ulusal (2008)), which informally says that there is a natural one to one correspondence between the nodes of any two branch decompositions of the same hypergraph, which consequently only differ in terms of their edges.

Observation 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 and 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 $l \in L(T)$.*

Proof. Because $\mathcal{B}$ and $\mathcal{B}'$ are branch decompositions of the same hypergraph H, it holds that $|L(T)| = |L(T')| = |E(H)|$. Moreover, because all inner vertices of T and T' are ternary, it holds that $|V(T)| = |V(T')| = 2|E(H)| - 2$. Hence, the bijection α can be obtained by setting $\alpha(l)$ to be the leaf l' of T' with $\gamma(l) = \gamma'(l')$ for every leaf l of T and choosing an arbitrary bijection between the remaining (inner) vertices of T and T'. □

(Weak) Partitions. As (weak) partitions play an important role in our reformulation of width parameters, we recall some basic terminology. A *weak partition* of a set S is a family P of nonempty subsets of S such that any two sets in P are disjoint. We denote by $\mathrm{U}(P)$ the union of all sets in P. If additionally $S = \mathrm{U}(P)$, then P is a *partition*. The elements of P are called *equivalence classes*. Let P, P' be weak partitions of S. Then P' is a *refinement* of P if $\mathrm{U}(P) \subseteq \mathrm{U}(P')$ and any two elements $x, y \in S$ that are in the same equivalence class of P' are not in distinct equivalence classes 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 $p \subseteq \bigcup_{1 \leq i \leq k} p_i$.

3 BRANCHWIDTH

In this section, we introduce our encoding for branchwidth. The encoding is based on a partition-based reformulation of branchwidth in terms of derivations, which will also lead to an efficient encoding for the related notion of carving-width.

3.1 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 Observation 1 one could encode the branch decomposition as a formula 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 were 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 *lengthl* 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 $i \in \{1, \ldots, l-2\}$, 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 $\delta_H(S)$ over all sets $S \in \bigcup_{1 \leq i < l} P_i$. 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. Characterizing branch decompositions in this way is very natural, similar concepts have been considered by Gu and Tamaki (2008) in terms of binary mergings. The main purpose of introducing derivations here is to provide a clean tailor-made characterization of branch decompositions that is best suited to be employed by our encodings.

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

$P_1 = \{\{129\}, \{35\}, \{45\}, \{3A\}, \{14\}, \{28\}, \{38\}, \{29\}\}$,
$P_2 = \{\{129\}, \{35\}, \{45, 3A\}, \{14\}, \{28\}, \{38\}, \{29\}\}$,
$P_3 = \{\{129\}, \{35, 45, 3A\}, \{14\}, \{28, 38\}, \{29\}\}$,
$P_4 = \{\{129\}, \{35, 45, 3A, 14\}, \{28, 38, 29\}\}$,
$P_5 = \{\{129, 35, 45, 3A, 14, 28, 38, 29\}\}$.

The width of $\mathcal{B}$ is equal to the width of $\mathcal{P}$. Note that every set S in $\bigcup_{1 \leq i \leq 5} P_i$ naturally corresponds to a node t of $\mathcal{B}$, and its elements are all edges assigned to leaves below t in $\mathcal{B}$. Moreover, P_i contains all sets corresponding to nodes of $\mathcal{B}$ at the same level, i.e., two nodes of $\mathcal{B}$ are at the same level if their maximum distance to any leaf below them in $\mathcal{B}$ is equal. For instance, P_1 contains all sets corresponding to the leaves of $\mathcal{B}$ and P_2 contains all sets corresponding to inner nodes of $\mathcal{B}$ that have only leaves as children. Conversely, given the derivation $\mathcal{P}$ one can easily construct the branch decomposition $\mathcal{B}$ as follows: First one adds a leaf for every set in P_1, then for every set in P_2 that is the union of two sets S and S' in P_1 one adds a new node whose children are the leaves corresponding to S and S'. This process then continues in the same manner until one reaches P_5 for which one adds the root node of $\mathcal{B}$ whose children are the three nodes corresponding to the sets in P_4.

Using the same ideas as provided in the example, we can now show the following theorem.

Theorem 3.2. *Let H be a hypergraph and w and d 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.*

Proof. Toward showing the forward direction of the theorem, let $\mathcal{B} := (T, \gamma)$ be a branch decomposition of width at most w and depth at most d. Moreover, let r be an arbitrary node of degree three at the center of T and assume in the following that T is rooted in r. Observe that because r is in the center of T, it holds that $\mathrm{h}(T_r) = d$. Let t be a node of T. We define $E(t)$ to be the set of all edges of H represented by the leaves of the subtree T_t, i.e., $E(t) := \gamma(L(T_t))$. We claim that $\mathcal{P} := (P_1, \ldots, P_{\mathrm{h}(T_r)})$, where $P_i := \{\, E(t) \mid t \in V(T) \text{ and } \mathrm{h}(T_t) = i \,\}$, is a derivation of H of width at most w and length at most d. Because in every tree, the set of all subtrees of T of a fixed height partitions the leaves of T, we obtain that P_i is a partition of $E(H)$ for every i with $1 \leq i \leq \mathrm{h}(T_r)$. Because $P_1 = \{\, E(l) \mid l \in L(T) \,\} = \{\, \gamma(l) \mid l \in L(T) \,\}$, $P_{\mathrm{h}(T_r)} = \{E(r)\} = \{E(H)\}$, and $\mathcal{B}$ is a branch decomposition, we obtain that $\mathcal{P}$ satisfies ***(D1)***. Because every node of T apart from r has at most two children and r has exactly three children, we obtain that P_i is a 2-ary refinement of P_{i+1} for every i with $1 \leq i < \mathrm{h}(T_r) - 1$ and $P_{\mathrm{h}(T_r)-1}$ is a 3-ary refinement of $P_{\mathrm{h}(T_r)}$, which shows that $\mathcal{P}$ satisfies ***(D2)*** and ***(D3)***. Hence, $\mathcal{P}$ is a derivation of H, and because $\mathrm{h}(T_r) = d$, as observed in the beginning of the proof, we obtain that the length of $\mathcal{P}$ is at most d.

It remains to show that the width of $\mathcal{P}$ is at most w. To see this let $E \in \bigcup_{1 \leq i < d} P_i$. Then $E = E(t)$ for some $t \in V(T) \setminus \{r\}$. Hence, $\delta_H(E)$ is equal to $\delta_{\mathcal{B}}(\{t, \mathrm{p}(t)\})$, which is at most w, because the width of $\mathcal{B}$ is at most w.

Toward showing the backward direction of the theorem, let $\mathcal{P} := (P_1, \ldots, P_d)$ be a derivation of H of width at most w. We will first show that, w.l.o.g., we can assume that $|P_{d-1}| = 3$. Since we can assume that $\mathcal{P}$ is a minimal derivation, i.e., every subsequence of $\mathcal{P}$ is not a derivation, we obtain that $P_{d-1} \neq P_d$ and hence $|P_{d-1}| \geq 2$. Suppose that $|P_{d-1}| = 2$. Because we can assume that H has at least three edges there is a $p \in P_{d-1}$ and a level i with $1 \leq i < d - 1$ such that p is the union of two elements p' and p'' in P_i and p occurs in every P_j with $i < j \leq d - 1$. Then the derivation obtained from $\mathcal{P}$ after replacing p with p' and p'' in every level j with $i < j \leq d - 1$ satisfies $|P_{d-1}| = 3$. Hence, for the remainder of the proof we will assume that $|P_{d-1}| = 3$.

We claim that $\mathcal{B} := (T, \gamma)$ with T and γ as defined below is a branch decomposition of H of width at most w and depth at most d. The tree T contains one node t_p for every $p \in \bigcup_{1 \leq i \leq d} P_i$ and T has an edge between t_p and $t_{p'}$ if and only if there is an i with $1 \leq i < d$ such that $p \in P_i$, $p' \in P_{i+1}$, and $p \subsetneq p'$. Moreover, the bijection γ is defined by setting $\gamma(t_l) = l$ for every $l \in P_1$. It is straightforward to verify that $\mathcal{B}$ is indeed a branch decomposition of H with width at most w and depth at most d. □

One important parameter influencing the size of the encoding is the length of the derivation. The next theorem shows a tight upper bound on the length of any derivation obtained from some 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 one can further reduce the depth of the branch decomposition by replacing small subtrees at the bottom of the branch decomposition, i.e., subtrees for which no edge has maximum width, with complete binary subtrees of smaller depth.

Theorem 3.3. *Let H be a hypergraph, e the maximum size over all edges of H, and w a positive 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 $\lfloor |E(H)|/2 \rfloor - \lceil w/e \rceil + \lceil \log \lfloor w/e \rfloor \rceil$.*

Proof. The backward direction of the claim follows immediately from Theorem 3.2.

Toward showing the forward direction, we first show that every branch decomposition of width at most w can be transformed into a branch decomposition of the same width and whose depth is at most $\lfloor |E(H)|/2 \rfloor - \lceil w/e \rceil + \lceil \log \lfloor w/e \rfloor \rceil$. The claim then follows from Theorem 3.2.

Let $\mathcal{B} := (T, \gamma)$ be a branch decomposition of H of width at most w. Because T is a ternary tree with exactly $|E(H)|$ leaves, we obtain that its radius is at most $\lfloor |E(H)|/2 \rfloor$. Assume in the following that T is rooted in one of the (at most two) center vertices, say r, of T. The main idea to obtain the exact bound on the radius of T given in the statement of the theorem is now to replace every subtree of T rooted at some node, say t, that contains at most $\lfloor w/e \rfloor$ edges of H and whose height is maximal with respect to this property with a binary tree (containing the same leaf nodes) of height at most $\lceil \log \lfloor w/e \rfloor \rceil$. Because every edge in the obtained binary tree has width at most $(w/e)e = w$, this replacement does not increase the width of $\mathcal{B}$ and it is straightforward to verify that the depth of the obtained branch decomposition is at most $\lfloor |E(H)|/2 \rfloor - \lceil w/e \rceil + \lceil \log \lfloor w/e \rfloor \rceil$. □

3.2 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 integers from 1 to n and the edges of H by the integers from 1 to m. Thus, when we say an edge e is smaller than another edge f ($e < f$), we refer to their representation by integers. The aim of this section is to construct a formula $F(H, w, d)$ that is satisfiable if and only if H has a derivation of width at most w and length at most d. Because of Theorem 3.3 (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 constraints that restrict the width of the derivation to w.

3.2.1 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:

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

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

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

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:

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

Part A ensures that e is a leader or it is in a set with an edge that is smaller than e; part B ensures that if e is not in the 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)***:

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

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 $O(m^2 d)$ variables and $O(m^3 d)$ clauses.

3.2.2 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 load 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 load vertices of edge sets in the derivation. In particular, we introduce a *load 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 load vertex of the set containing e at level i of the derivation. To restrict the size of the sets of load vertices later on, we do not need the backward direction of the previous statement. Recall that a vertex u is a load 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.

Defining the Load 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 $\mathrm{c}(e,u,i)$ 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 load vertex for every leading edge of the sets containing the incident edges:

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

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

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

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

The definition of load vertices adds $O(mnd)$ variables and $O(m^3 nd)$ clauses.

Restricting the Order of the Separations. Next, we describe how to restrict the size of all sets of load vertices to w and thereby complete the encoding of $F(H,w,d)$. In particular, our aim is to restrict

Table 1. Illustration of the Behavior of the Sequential Counter for the Case that H has Six Vertices (Labeled from 1 to 6) and $w = 4$

u	$c(e,u,i)$	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

The first column identifies the vertex u, the second column gives the value of the variable $c(e, u, i)$ for a fixed edge e and a fixed level i and the last four columns give the values of the variables $\#(e, u, i, j)$.

the number of vertices $u \in V(H)$ for which a variable $c(e, u, i)$ is true for some $e \in E(H)$ and $1 \leq i \leq d$. In this article, we will only present the *sequential counter* approach (Samer and Veith 2009), since this approach has turned out to provide the best results in our setting. We also considered the *order encoding* (Heule and Szeider 2015) 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 load vertex of the edge e. We need the following clauses:

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

This completes the construction of the formula $F(H, w, d)$. In total $F(H, w, d)$ has $O(m^2d + mndw) \subseteq O(m^3 + m^2n^2)$ variables and $O(m^3nd + mndw) \subseteq O(m^4n + m^2n^2)$ clauses. By construction, $F(H, w, d)$ is satisfiable if and only if H has a derivation of width at most w and length at most d. Because of Theorem 3.2, we obtain the following.

Theorem 3.4. *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 time that is linear in the number of variables of $F(H, w, d)$.*

4 CARVING-WIDTH

In this section, we introduce our encoding for carving-width. Carving-width is a decompositional parameter that is closely related to branchwidth and has been introduced by Seymour and Thomas (1994). They showed that, given a graph G and an integer k, deciding whether G has carving-width $\leq k$ is NP-complete, but it can be decided in polynomial time if G is planar. If k is a constant and not part of the input, then it can be decided in linear time whether G has carving-width $\leq k$ (Thilikos

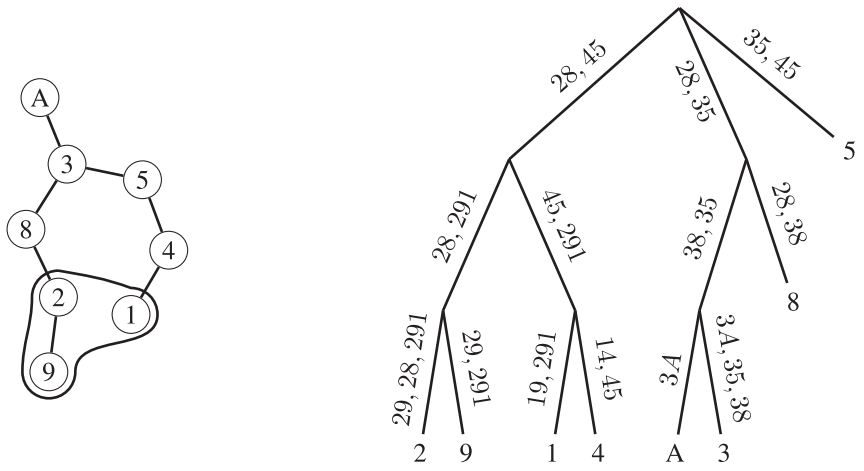

Fig. 2. A hypergraph H (left) and an optimal carving (T, γ) of H (right). The labels of the leaves of T are the vertices assigned to them by γ and the labels of the edges of T are the cut edges of that edge.

et al. 2000). Interestingly, the known polynomial-time algorithm for computing the branchwidth of a planar graph is based on the corresponding algorithm for carving-width and uses the fact that the carving-width of the so-called medial graph of a planar graph is exactly twice the branchwidth of the original graph.

Carving decompositions (or simply *carvings*) are defined similarly to branch decompositions with two important differences: (i) the leaves of a carving are in correspondence to the vertices instead of the edges of the hypergraph, and (ii) the width of an edge (and in consequence the width of a carving) is measured in terms of the number of edges of the hypergraph with at least one endpoint in both components of the carving decomposition obtained after deleting the edge.

Let $H = (V, E)$ be a hypergraph and $V' \subseteq V$. We denote by $\delta(V')$ the set of edges $e \in E$ that have at least one endpoint in V' and outside of V', i.e., $e \cap V' \neq \emptyset$ and $e \setminus V' \neq \emptyset$. A *carving* $C(H)$ of a hypergraph $H = (V, E)$ is a pair (T, γ), where T is a ternary tree and $\gamma : L(T) \to V$ is a bijection between the vertices 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 L of leaves 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 $\delta_C(e)$ (or simply $\delta(e)$ if C is clear from the context) the set of *cut edges* of e, i.e., the set $\delta(T')$, where T' is any of the two components of $T \setminus \{e\}$. The *width* of an edge e of T is the number of cut edges of e, i.e., $|\delta_C(e)|$ and the *width* of C is the maximum width of any edge of T. The *carving-width* of H is the minimum width over all carvings of H (or 0 if $|V(H)| = 1$ and H has no carving). We also define the *depth* of C as the radius of T. Figure 2 illustrates a carving of a small hypergraph.

4.1 Partition-based Reformulation of Carving-width

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

(D1) $P_1 = \{ \{v\} \mid v \in V \}$ and $P_l = \{V\}$ and
(D2) for every $i \in \{1, \ldots, l-2\}$, 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 $\delta_H(V)$ over all sets $V \in \bigcup_{1 \leq i < l} P_i$. We will refer to P_i as the ith *level* of the carving derivation $\mathcal{P}$, and we will refer to elements in $\bigcup_{1 \leq i \leq l} P_i$ as *sets* of the

carving derivation. We will show that any carving can be transformed into a carving derivation of the same width and also the other way around. The following example illustrates the close connection between carvings and carving derivations.

Example 4.1. Consider the carving decomposition C given in Figure 2. Then C can, e.g., be translated into the derivation $\mathcal{P} = (P_1, \ldots, P_5)$ defined by

$P_1 = \{\{2\}, \{9\}, \{1\}, \{4\}, \{A\}, \{3\}, \{8\}, \{5\}\}$,
$P_2 = \{\{2, 9\}, \{1, 4\}, \{A, 3\}, \{8\}, \{5\}\}$,
$P_3 = \{\{2, 9, 1, 4\}, \{A, 3, 8\}, \{5\}\}$,
$P_4 = \{\{2, 9, 1, 4, A, 3, 8, 5\}\}$.

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

The following theorem shows that derivations provide an alternative characterization of carving decompositions. Since the proof uses the same construction and is also otherwise very similar to the proof of Theorem 3.2, we will not repeat it here.

Theorem 4.2. *Let H be a hypergraph and w and d integers. H has a carving of width at most w and depth at most d if and only if H has a carving derivation of width at most w and length at most d.*

As in the case of branchwidth, it will be beneficial for our encoding to obtain a tight bound on the length of a carving derivation. The next theorem shows a tight upper bound on the length of any carving derivation obtained from some carving decomposition. The main ideas are similar to the ideas used for branch decompositions (Theorem 3.3), however, there are some subtle differences. 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.

Theorem 4.3. *Let H be a hypergraph with maximum degree Δ and w an integer. Then the carving-width of H is at most w if and only if H has a carving derivation of width at most w and length at most $\lfloor |V(H)|/2 \rfloor - \lceil w/\Delta \rceil + \lceil \log \lfloor w/\Delta \rfloor \rceil$.*

Proof. The backward direction of the claim follows immediately from Theorem 4.2.

Toward showing the forward direction, we first show that every carving of width at most w can be transformed into a carving of the same width and whose depth is at most $\lfloor |V(H)|/2 \rfloor - \lceil w/\Delta \rceil + \lceil \log \lfloor w/\Delta \rfloor \rceil$. The claim then follows from Theorem 4.2.

Let $C := (T, \gamma)$ be a carving of H of width at most w. Because T is a ternary tree with exactly $|V(H)|$ leaves, we obtain that its radius is at most $\lfloor |V(H)|/2 \rfloor$. Assume in the following that T is rooted in one of the (at most two) center vertices, say r, of T. The main idea to obtain the exact bound on the radius of T given in the statement of the theorem is now to replace every subtree of T rooted at some node, say t, that contains at most $\lfloor w/\Delta \rfloor$ vertices of H and whose height is maximal with respect to this property with a binary tree (containing the same leaf nodes) of height at most $\lceil \log \lfloor w/\Delta \rfloor \rceil$. Because every edge in the obtained binary tree has width at most $(w/\Delta)\Delta = w$, this replacement does not increase the width of C and it is straightforward to verify that the depth of the obtained carving decomposition is at most $\lfloor |V(H)|/2 \rfloor - \lceil w/\Delta \rceil + \lceil \log \lfloor w/\Delta \rfloor \rceil$. □

4.2 Encoding

The encoding for carving derivations is very similar (actually almost identical) to the encoding we presented for branch decompositions in Section 3.2. In particular, one can use the exact same encoding for the formulas $F(H, w, d)$ and $F(H, w, d)$ as for derivations after switching the role that the vertices and edges of the hypergraph play in the encoding. For instance, the set variables

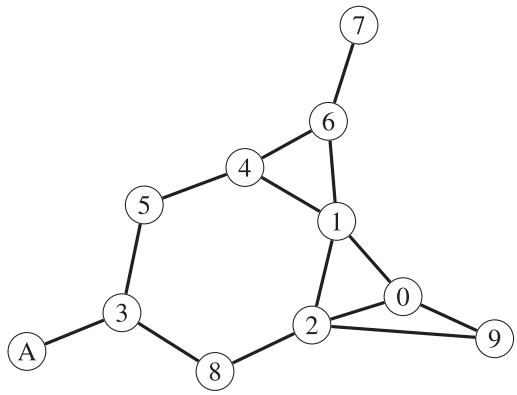

Fig. 3. The graph H used to illustrate the main idea behind our local improvement procedure.

$\mathrm{s}(e, f, i)$ that were defined for all edges $e, f \in E(H)$ with $e < f$ in the encoding for branchwidth, will now be defined for all vertices $e, f \in V(H)$ with $e < f$. Similarly, the cut variables $\mathrm{c}(e, u, i)$ that were defined for all edges $e \in E(H)$ and vertices $u \in V(H)$, will now be defined for all vertices $e \in V(H)$ and all edges $u \in E(H)$.

5 LOCAL IMPROVEMENT FOR BRANCH DECOMPOSITIONS

The encoding presented in Section 3.2 allows us to compute the exact branchwidth of hypergraphs up to a certain size. Due to the intrinsic 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 a 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 a better one found by means of the SAT encoding and repeats this process until a fixed-point (or timeout) is reached.

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 hyperedge $\gamma_L(l)$ 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 Figures 4 and 5.

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 local 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 Cook and Seymour (2003) and Hicks (2002). 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 $T[M]$, where $T[M]$ is the forest with vertex set $V(T)$ and edge set M, and for every component $C \in \mathcal{C}$

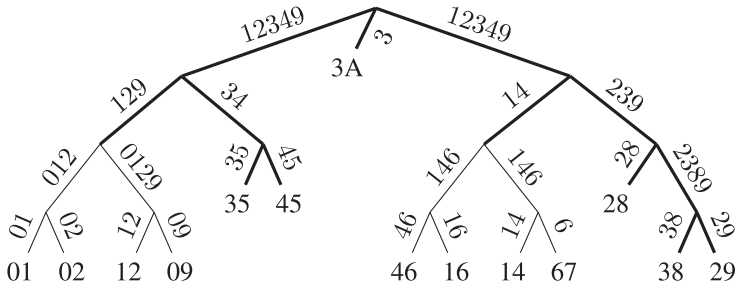

Fig. 4. A branch decomposition $\mathcal{B}$ of the graph H given in Figure 3 together with an example of a local branch decomposition $\mathcal{B}_L$ (highlighted by thicker edges) chosen by our algorithm.

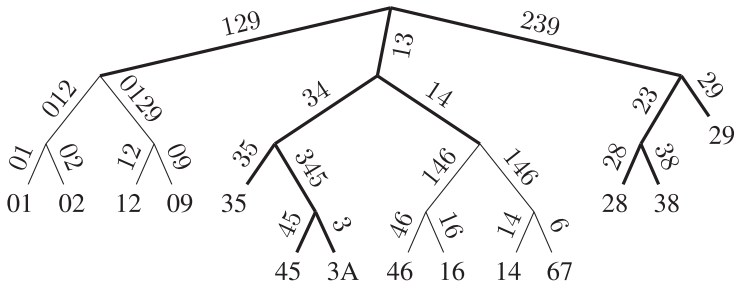

Fig. 5. The improved branch decomposition $\mathcal{B}'$ obtained from $\mathcal{B}$ after replacing the local branch decomposition $\mathcal{B}_L$ of $H(T_L)$ with an optimal branch decomposition $\mathcal{B}'_L$ of $H(T_L)$ obtained from our SAT encoding. See Figure 4 for an illustration of $\mathcal{B}$ and $\mathcal{B}_L$.

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, then the algorithm updates $\mathcal{B}$ by replacing the part of $\mathcal{B}$ represented by T_L with $\mathcal{B}'_L$ according to Theorem 5.1 and proceeds with line 4. If, however, $\mathcal{B}_L$ cannot be improved, then 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 that should be small enough to ensure solvability 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, then 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 reaches 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

ALGORITHM 1: Local Improvement

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"
  $C \leftarrow$ "the set of components of $T[M]$"
  `improved` $\leftarrow$ *false*
  for $C \in C$ **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
return $\mathcal{B}$

ALGORITHM 2: ImproveLD

Input: 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 $\mathcal{B}'_L$

the formula $F(H(T_L), w, d)$ from Theorem 3.4 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), then `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 5.1.

Let H be a hypergraph, $\mathcal{B} := (T, \gamma)$ a branch decomposition of H, T_L a connected ternary subtree of T, $\mathcal{B}_L := (T_L, \gamma_L)$ 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 Observation 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.

ALGORITHM 3: Local Selection (LocalBD).

Input: A branch decomposition $\mathcal{B} := (T, \gamma)$ of H and acomponent C of T
Output: A local branch decomposition of $\mathcal{B}$

```
w ← "the width of B"
T_L ← C
for c ∈ V(C) with deg_C(c) = 2 do
    "add the unique third neighbor and its edge incident to c to T_L"
/* Q is a first-in-first-out queue data-structure definingthe usual functions
   ''push'' and ''pop''.                                                    */
Q ← "the set of leaves of T_L"
while Q ≠ ∅ and |T_L| ≤globalbudget −2 do
    l ← Q.pop()
    if "l is not a leaf of T" then
        c, c' ← "the two neighbors of l in T that are notneighbors of l in T_L"
        if δ_B({l, c}) < w and δ_B({l, c'}) < w then
            "add c and c' together with their edges incident to l to T_L"
            Q.push(c)
            Q.push(c')
return "the local branch decomposition of B represented by T_L"
```

Theorem 5.1. *$\mathcal{B}(\frac{\mathcal{B}_L}{\mathcal{B}'_L})$ is a branch decomposition of H, whose width is the maximum of the width of $\mathcal{B}'_L$ and the maximum width over all edges $e \in E(T) \setminus E(T_L)$ in $\mathcal{B}$.*

Proof. It is easy to verify that $\mathcal{B}(\frac{\mathcal{B}_L}{\mathcal{B}'_L})$ is indeed a branch decomposition of H.

Toward showing that the width of $\mathcal{B}(\frac{\mathcal{B}_L}{\mathcal{B}'_L})$ is equal to the maximum of the width of $\mathcal{B}'_L$ and the maximum width of any edge $e \in E(T) \setminus E(T_L)$ in $\mathcal{B}$, we first give an alternative definition for γ_L.

Let F be the forest obtained from T after deleting all edges of T_L, i.e., F is the forest $T \setminus E(T_L)$. Then every leaf of T_L and also every leaf of T is contained in exactly one component of F. Moreover, because T is a tree every component of F contains at most one leaf of T_L. Let Lcut : $L(T_L) \to L(T)$ be the mapping that assigns to every leaf l of T the set of all leaves of T that are contained in the same component as l in F. Note that Lcut naturally associates every leaf l of T_L to the cut $(\gamma(\text{Lcut}(l)), E(H) \setminus \gamma(\text{Lcut}(l)))$. Note that $\delta_H(\gamma(\text{Lcut}(l))) = \delta_{\mathcal{B}}(e)$ for every $l \in L(T_L)$, where e is the (unique) edge in T_L incident to l. Hence, in the following, we will assume that $\gamma_L(l)$ is equal to $\delta_H(\gamma(\text{Lcut}(l)))$.

We are now ready to prove the statement of the theorem concerning the width of $\mathcal{B}(\frac{\mathcal{B}_L}{\mathcal{B}'_L})$. Observe that it is sufficient to show that for every edge e of T' either $\delta_{\mathcal{B}'}(e) = \delta_{bd}(e)$ if $e \in E(T') \setminus E(T'_L)$ or $\delta_{\mathcal{B}'}(e) = \delta_{\mathcal{B}'_L}(e)$ if $e \in E(T'_L)$. Toward showing the former case, let $e \in E(T') \setminus E(T'_L)$. Because T_L and T'_L are connected the components of $T \setminus \{e\}$ are the same as the components of $T' \setminus \{e\}$ for every such edge e. Hence, $\delta_{\mathcal{B}'}(e) = \delta_{\mathcal{B}}(e)$, as required.

Toward showing the later case, let $e \in E(T'_L)$ and let C_1 and C_2 be the two components of $T' \setminus \{e\}$.

We start by showing that $\delta_{\mathcal{B}'}(e) \subseteq \delta_{\mathcal{B}'_L}(e)$. Because $v \in \delta_{\mathcal{B}'}(e)$, we obtain that there are two leaves l_1 and l_2 of T' with $l_1 \in V(C_1)$ and $l_2 \in V(C_2)$ such that $v \in \gamma'(l_1) \cap \gamma'(l_2)$. Observe that $f(l_1) \in V(C_1)$ and $f(l_2) \in V(C_2)$ and hence $v \in \gamma'_L(f(l_1))$ and $v \in \gamma'_L(f(l_2))$. Consequently, $v \in \delta_{\mathcal{B}'_L}(e)$. This shows that $\delta_{\mathcal{B}'}(e) \subseteq \delta_{\mathcal{B}'_L}(e)$ and it remains to show that $\delta_{\mathcal{B}'_L}(e) \subseteq \delta_{\mathcal{B}'}(e)$. Because $v \in \delta_{\mathcal{B}'_L}(e)$, we obtain that there are two leaves l_1 and l_2 of T'_L with $l_1 \in V(C_1)$ and $l_2 \in V(C_2)$ such that $v \in \gamma'_L(l_1) \cap \gamma'_L(l_2)$. Because $v \in \gamma'_L(l_1)$, we obtain that there is a leaf $l'_1 \in f^{-1}(l_1)$ such

that $v \in \gamma'(l'_1)$. Similarly, because $v \in \gamma'_L(l_2)$, we obtain that there is a leaf $l'_2 \in f^{-1}(l_2)$ such that $v \in \gamma'(l'_2)$. Note that because $l'_1 \in f^{-1}(l_1)$ it holds that $l'_1 \in V(C_1)$ and similarly because $l'_2 \in f^{-1}(l_2)$ it holds that $l'_2 \in V(C_2)$. Hence, $v \in \delta_{\mathcal{B}'}(e)$, which completes the proof of the theorem. □

6 LOCAL IMPROVEMENT FOR CARVING DECOMPOSITIONS

The local improvement approach introduced in the previous section for branchwidth can also be employed for carving-width in a very similar manner. Namely, if $C(H) = (T, \gamma)$ is a carving of a hypergraph $H = (V, E)$ and T_L is a connected ternary subtree of T, then we can define the *local carving* $C_L = (T_L, \gamma_L)$ of C by setting $\gamma_L(l) = \gamma(L(T^l))$ for every leaf $l \in L(T_L)$, where T^l is the unique subtree of T' containing l and T' is the subgraph of T obtained after deleting all edges in T_L from T. Note that (T_L, γ_L) is strictly speaking not a carving of H, since its leaves are assigned to subsets of vertices instead of single vertices. Since (T_L, γ_L) only partially decomposes H, i.e., it does not decompose the subsets of vertices assigned to its leaves, we call it a *partial carving*. One can now show, in a very similar manner as for branch decompositions, that any *partial carving* of H, whose leaves correspond to the same subsets of $V(H)$ as the leaves in (T_L, γ_L), can be used to replace (T_L, γ_L) in (T, γ) to obtain a carving of H with potentially smaller width. Moreover, finding a partial carving of smaller width can be achieved by employing almost the same encoding as introduced in Section 4. In particular, one merely needs to adapt Property (D1) of carving derivations (see Section 4.1) to ensure that the initial partition of a derivation is equal to the partition of $V(H)$ given by the leaves of (T_L, γ_L). Hence, the local improvement approach for branchwidth can be easily adapted to carvings with one exception: For the local improvement approach to work, it is crucial that one can obtain an initial carving very efficiently, e.g., via a heuristic method as in the case of branch decompositions. Unfortunately, we are not aware of any suitable heuristic method for the computation of carvings and have therefore refrained from implementing the local improvement approach for carvings.

7 EXPERIMENTAL RESULTS

We have implemented and tested the single SAT encoding for branchwidth and carving-width, as well as the SAT-based local improvement method for branchwidth. We tested them on various benchmark instances, including famous named graphs from the literature (Weisstein 2016), graphs from TreewidthLIB (Bodlander 2016), which originate from a broad range of applications, and a series of circular clusters (Cornuéjols 2001), which are hypergraphs denoted C^i_j with j vertices and j edges of size i. 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, 72GB RAM machine with Ubuntu 14.04 with each process having access to at most 8GB RAM.

7.1 Single SAT Encoding

To determine the branchwidth or carving-width of a graph or hypergraph with our encodings, 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 width 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 instances. Hence, whether we determine the width 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 solving time varies and depends on the structure of the (hyper)graph. We could determine

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
FlowerSnark	1.2	4.4	25.5	889.9	1.6	1.3	1.6	1.3	1.3
Errera	5.7	22.7	79.4	1,530.9	12.0	7.3	6.7	5.4	6.1
Folkman	3.4	13.7	98.6	2,747.0	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

the exact branchwidth and carving-width of many famous graphs known from the literature, see Table 3. For many of the graphs, the exact branchwidth or carving-width were not known before.

We verified the correctness of our encoding for branchwidth by comparing the widths computed by our method with the widths computed by Hicks' (2005) tangles-based algorithm. For carving-width, we are not aware of any other implemented algorithm, but as a sanity check, we used the fact that the carving-width of the medial graph of a planar graph is exactly two times the branchwidth of the graph (Seymour and Thomas 1994), and we tested this for a number of planar graphs.

Table 3 contains a comparison of our SAT encoding for branchwidth with Ulusal's ILP encodings (Ulusal 2008) as well as Hicks' tangles-based algorithm (2005) over all the graphs in the famous graphs benchmark suite. For this comparison, we re-implemented the integer programming encodings and compared our algorithm with this approach and the tangles-based algorithm, for which we obtained the source code from the authors. The table shows the running time in seconds for our encoding (SAT), the best running time for any of the three ILP encodings developed by Ulusal (ILP), and the running time of Hicks' tangles-based algorithm. The table shows that our SAT encoding for branchwidth solved 30, Ulusal's ILP encodings solved 8, Hicks' algorithm solved 36, and our SAT encoding for carving-width solved 38 out of the 41 famous graphs within a timeout of 100min.

To determine the branchwidth of (at least) some of the remaining graphs, we ran additional experiments with an increased timeout of 24h. This allowed us to compute the branchwidth of the Holt graph and the Shrikhande graph. Interestingly, whereas our SAT encoding could solve the Holt graph in less than 24h, the tangles-based algorithm ran out of memory and was not able to solve the instance. Since the Holt graph is also the graph with the highest branchwidth among all the famous graphs for which we know their branchwidth, this suggests that our SAT encoding for branchwidth might have an advantage over the tangles-based approach for instances with high branchwidth. This is in alignment with the fact that the time and space requirements of the tangles-based approach grow exponentially with the branchwidth of the graph (Hicks 2005).

Overall, the experiments show that on the famous graphs, except for the Holt graph, the tangles-based approach is superior to our SAT encoding, which in turn is superior to the ILP encodings developed by Ulusal. However, the results on the Holt graph suggest that our SAT encoding is better suited for the instances with high branchwidth that are encountered within our local improvement procedure.

Finally, to obtain a first indication of the performance of our SAT encodings on hypergraphs, we tested our encodings on the circular cluster hypergraphs C^i_{2i-1} (Cornuéjols 2001), which have also been used by Ulusal to evaluate the performance of ILP encodings. We were able to find the exact branchwidth for instances up to $i = 26$ and the exact carving-width for instances up to $i = 16$ using a timeout of 2,000s. For comparison, the best of Usual's ILP encodings could merely solve these instances up to $i = 6$ whereas Hicks' algorithm does not support hypergraphs.

Table 3. Experimental Results for Our SAT Encodings for Branchwidth and Carving-width on the Famous Graphs

Graph	$\|V\|$	$\|E\|$	Branchwidth				Carving-width
			ω	SAT	ILP	Tangles	ω
Ellingham	78	117	-	-	-	-	-
B10Cage	70	106	-	-	-	-	-
Watsin	50	75	6	-	-	32.14	6
Paley17	17	68	-	-	-	-	-
Kittell	23	63	6	-	-	315.88	12
Holt	27	54	9	★	-	MO	12
Shrikhande	16	48	8	★	-	★	16
Errera	17	45	6	1,384.21	-	3.70	12
Brinkmann	21	42	8	-	-	3,163.38	12
5x5-grid	25	40	5	13.45	-	0.06	6
Folkman	20	40	6	1,518.40	-	3.40	12
Clebsch	16	40	8	-	-	2,017.62	16
Poussin	15	39	6	645.63	-	4.75	11
Paley13	13	39	7	-	-	179.11	16
Robertson	19	38	8	-	-	3,906.94	12
McGee	24	36	7	-	-	177.92	8
Nauru	24	36	6	436.23	-	43.34	8
Hoffman	16	32	6	997.97	-	3.97	10
Desargues	20	30	6	474.72	-	2.10	6
Dodecahedron	20	30	6	152.96	-	2.29	6
Flower Snark	20	30	6	995.93	-	1.87	6
Pappus	18	27	6	241.12	-	1.76	6
Sousselier	16	27	5	18.47	-	0.12	7
Goldner-Harary	11	27	4	9.97	-	0.48	10
4x4-grid	16	24	4	3.77	-	0.10	5
Chvátal	12	24	6	81.20	-	0.82	8
Grötzsch	11	20	5	11.37	-	0.55	7
Dürer	12	18	4	2.45	-	0.01	4
Franklin	12	18	4	4.85	-	0.04	4
Frucht	12	18	3	2.16	475.59	0.41	4
Tietze	12	18	4	1.52	-	0.04	5
Herschel	11	18	4	8.49	-	0.08	6
Petersen	10	15	4	1.62	-	0.09	5
3x3-grid	9	12	3	1.10	1.18	0.02	4
Pmin	9	12	3	1.09	1.07	0.03	4
Wagner	8	12	4	1.54	-	0.02	4
Moser spindle	7	11	3	1.05	2.40	0.014	4
Prism	6	9	3	1.33	1.07	0.03	4
Butterfly	5	6	2	2.11	0.87	0.02	4

(Continued)

Table 3. Continued

Graph	$\|V\|$	$\|E\|$	Branchwidth				Carving-width
			ω	SAT	ILP	Tangles	ω
Bull	5	5	2	0.00	1.01	0.00	2
Diamond	4	5	2	2.09	1.10	52.54	3

The columns below "branchwidth" show the branchwidth ω (computed by any of the three methods), and the times in seconds required by our SAT encoding (SAT), the fastest of the three ILP encodings by Ulusal (ILP), and the tangles-based algorithm (Tangles). Finally, the ω column below "carving-width" shows the carving-width computed by our SAT encoding. "-" indicates that the instance could not be solved within a timeout of 24h and $\star$ indicates that the instance could be solved within 24h but not within our initial timeout of 100min. We indicate an out of memory error by "MO."

7.2 SAT-based Local Improvement

We tested our local improvement method on graphs with several thousands of vertices and edges, with initial branch decompositions of width up to above 200. In particular, we tested it on all graphs from TreewidthLIB omitting graphs that are minors of other graphs as well as small graphs with 150 or fewer edges (small graphs can be solved with the single SAT encoding). These are in total 740 graphs with up to 5,934 vertices and 17,770 edges. We ran our SAT-based local improvement algorithm on each graph with a timeout of 6h, where each SAT-call had a timeout of 1,200s and a memory limit of 8GB. We computed the initial branch decomposition by a greedy heuristic described by Hicks (2005) who kindly provided us the implementation. We conducted our experiments using different values for the budget, i.e., the parameter `globalbudget` used in Algorithm 1 bounding the maximum number of edges in the local hypergraph, as well as different values of depth for the derivation of the local hypergraph (the parameter `length(H,w)` used in Algorithm 1). Tables 5, 6, and 7 illustrate our experimental results for *budgets* between 120 and 210 and *depths* ranging between $m/5$ and m as well as the depth given by Theorem 3.3 (here m refers to the number of edges of the local hypergraph). To compare the performance of our approach for different values of these two parameters, we use the following performance indicators:

— the sum of the improvement over all instances (Table 5),
— the maximum improvement for any of the instances (Table 6),
— the total number of instances whose branchwidth could be improved by at least one (Table 7).

For the sum of improvements (Table 5) as well as for the total number of improved instances (Table 7) the best combination turned out to be a budget of 200 and a depth of $m/3$. For this combination, the sum of improvements is 1,483, and we improved the width of 476 out of 740 instances. With regards to the maximum improvement of any instance (Table 6), this combination performed well with a maximum improvement of 20; however, the combination with a budget of 140 and using the optimal depth performed even better, allowing us to improve the width of an instance by 22.

In Table 4, we list some instances that we found particularly notable for various aspects, such as a significant improvement, a large number of vertices and edges, or a remarkably low or high width of the initial branch decomposition.

Our experiments show that the SAT-based local improvement method 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.

Table 4. Results for SAT-based Local Improvement for a Selection of Example Instances from TreewidthLIB

Graph	$\|V\|$	$\|E\|$	Branchwidth		Difference	
			Initial	Improved		
bn_63-pp	426	1,489	73	51	22	significant improvement
bn_51	661	2,131	95	75	20	
bn_77	1,020	2,616	40	23	17	
rl5915.tsp	5,915	17,728	70	64	6	large number of edges
fl3795.tsp	3,795	11,326	49	42	7	
fnl4461.tsp	4,461	13,359	82	72	10	
bn_43-pp	254	725	23	17	6	low initial width
fl1400.tsp-pp	1,390	4,108	23	14	9	
vm1084.tsp-pp	808	2,312	29	19	10	
graph09	458	1,667	125	117	8	large initial width
graph13-wpp	427	1,778	137	129	8	
pignet2-pp	1,024	3,774	175	173	2	

Table 5. Sum of Improvements Over All the Instances for the Various Configurations

budget	$m/1$	$m/2$	optimal	$m/3$	$m/4$	$m/5$
120	809	1,182	1,138	1,200	911	670
130	1,055	1,249	1,216	1,343	1,221	896
140	1,043	1,338	1,299	1,387	1,291	961
150	1,055	1,338	1,318	1,454	1,375	1,028
160	1,004	1,350	1,350	1,453	1,390	1,047
170	962	1,350	1,352	1,460	1,390	1,035
180	913	1,322	1,293	1,454	1,342	1,033
190	934	1,309	1,296	1,478	1,401	1,121
200	780	1,288	1,090	**1,483**	1,156	907
210	891	1,209	1,349	1,395	1,363	1,046

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. Our experiments also show that it can be worth to tune the local improvement approach using parameters such as budget and depth.

8 FINAL REMARKS

We have presented a first SAT encoding for branchwidth based on a novel partition-based formulation of branch decompositions and introduced the new SAT-based local improvement method. We also formulated a SAT-based local improvement method for carving width. Our SAT-based local improvement method provides the means for scaling the SAT-approach to significantly larger instances and exhibits a fruitful new application field of SAT solvers. In many cases, the SAT-based local improvement method could obtain branch decompositions of a width that makes a dynamic

Table 6. Maximum Improvement Over All the Instances for the Various Configurations

budget	$m/1$	$m/2$	optimal	$m/3$	$m/4$	$m/5$
120	14	21	21	20	17	16
130	17	19	20	20	17	14
140	16	20	**22**	20	17	16
150	18	19	20	20	18	17
160	10	19	20	20	17	15
170	10	19	20	20	16	15
180	20	19	20	20	18	19
190	10	19	20	20	19	15
200	10	19	20	20	16	15
210	10	19	20	20	16	17

Table 7. Number of Improved Instance for the Various Configurations

budget	$m/1$	$m/2$	optimal	$m/3$	$m/4$	$m/5$
120	316	395	390	401	322	260
130	394	421	411	426	412	331
140	406	435	428	442	431	353
150	412	448	439	459	447	370
160	417	458	448	465	458	380
170	405	454	450	465	458	380
180	392	457	446	466	444	368
190	403	462	448	470	463	422
200	339	466	378	**476**	393	332
210	379	444	515	462	456	380

programming feasible, which was not possible with the original branch decomposition obtained by a heuristics.

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 (Berg and Järvisalo 2014). Also for the local improvement, we see various directions for further research. For instance, when a local branch decomposition cannot be improved, one could use a SAT solver to obtain an alternative branch decomposition of the same width but where other parameters are optimized, e.g., the number of maximum separations. This could propagate into adjacent local improvement steps and yield an overall branch decomposition of smaller width. Our experiments show the performance of our local improvement approach depends on the exact combination of various parameters such as budget and depths. It would therefore be interesting for future work to investigate the benefit from tools for automated parameter configuration (Falkner et al. 2015).

Finally, we will mention that branch decompositions are the basis for several other (hyper)graph width measures such as rankwidth and Boolean-width (Adler et al. 2010), as well as to width-parameters employed in Knowledge Compilation and Reasoning (Darwiche 2009). Hence,

we think that it might be fruitful to extend our methods to such width measures related to branchwidth and leave this for future research.

ACKNOWLEDGMENT

We thank Illya Hicks for providing us the code of his branchwidth heuristics.

REFERENCES

Isolde Adler, Binh-Minh Bui-Xuan, Yuri Rabinovich, Gabriel Renault, Jan Arne Telle, and Martin Vatshelle. 2010. On the Boolean-width of a graph: Structure and applications. In *Proceedings of the 36th International Workshop on Graph Theoretic Concepts in Computer Science (WG'10) (Lecture Notes in Computer Science)*, Dimitrios M. Thilikos (Ed.), Vol. 6410. 159–170.

Michael Alekhnovich and Alexander A. Razborov. 2002. Satisfiability, branch-width and Tseitin tautologies. In *Proceedings of the 43rd Symposium on Foundations of Computer Science (FOCS'02)*. IEEE Computer Society, 593–603.

Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi. 2003. Algorithms and complexity results for #SAT and Bayesian inference. In *Proceedings of the 44th Symposium on Foundations of Computer Science (FOCS'03)*. IEEE Computer Society, 340–351.

Max Bannach, Sebastian Berndt, and Thorsten Ehlers. 2017. Jdrasil: A modular library for computing tree decompositions. In *Proceedings of the 16th International Symposium on Experimental Algorithms (SEA'17)*, Costas S. Iliopoulos, Solon P. Pissis, Simon J. Puglisi, and Rajeev Raman (Eds.), Vol. 75. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 28:1–28:21.

Jeremias Berg and Matti Järvisalo. 2014. SAT-based approaches to treewidth computation: An evaluation. In *Proceedings of the 26th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'14)*. IEEE Computer Society, 328–335.

Therese C. Biedl. 2014. On area-optimal planar graph drawings. In *Proceedings of the 41st International Colloquium Automata, Languages, and Programming (ICALP'14) (Lecture Notes in Computer Science)*, Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias (Eds.), Vol. 8572. Springer, 198–210.

Therese C. Biedl and Martin Vatshelle. 2012. The point-set embeddability problem for plane graphs. In *Proceedings of the Symposuim on Computational Geometry (SoCG'12)*, Tamal K. Dey and Sue Whitesides (Eds.). ACM, 41–50.

Hans Bodlander. 2016. TreewidthLIB: A benchmark for algorithms for Treewidth and related graph problems. Retrieved from http://www.staff.science.uu.nl/ bodla101/treewidthlib/.

William Cook and Paul Seymour. 2003. Tour merging via branch-decomposition. *INFORMS J. Comput.* 15, 3 (2003), 233–248.

Gérard Cornuéjols. 2001. Combinatorial optimization: Packing and covering. In *Proceedings of the Regional Conference Series in Applied Mathematics*. Society for Industrial and Applied Mathematics.

Bruno Courcelle. 1990. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Info. Comput.* 85, 1 (1990), 12–75.

Adnan Darwiche. 2009. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press. I–XII, 1–548 pages.

Reinhard Diestel. 2000. *Graph Theory* (2nd ed.). Graduate Texts in Mathematics, Vol. 173. Springer Verlag, New York.

Stefan Falkner, Marius Thomas Lindauer, and Frank Hutter. 2015. SpySMAC: Automated configuration and performance analysis of SAT solvers. In *Proceedings of the 18th International Conference on Theory and Applications of Satisfiability Testing (SAT'15) (Lecture Notes in Computer Science)*, Marijn Heule and Sean Weaver (Eds.), Vol. 9340. Springer, 215–222.

Johannes Klaus Fichte, Neha Lodha, and Stefan Szeider. 2017. SAT-based local improvement for finding tree decompositions of small width. In *Proceedings of the 20th International Conference on Theory and Applications of Satisfiability Testing (SAT'17) (Lecture Notes in Computer Science)*, Serge Gaspers and Toby Walsh (Eds.), Vol. 10491. Springer, 401–411.

Fedor V. Fomin, Frédéric Mazoit, and Ioan Todinca. 2009. Computing branchwidth via efficient triangulations and blocks. *Discr. Appl. Math.* 157, 12 (2009), 2726–2736.

Robert Ganian, Neha Lodha, Sebastian Ordyniak, and Stefan Szeider. 2019. SAT-encodings for treecut width and treedepth. In *Proceedings of the 21st Workshop on Algorithm Engineering and Experiments (ALENEX'19)*, Stephen G. Kobourov and Henning Meyerhenke (Eds.). SIAM, 117–129.

Martin Grohe. 2008. Logic, graphs, and algorithms. In *Logic and Automata: History and Perspectives (Texts in Logic and Games)*, Jörg Flum, Erich Grädel, and Thomas Wilke (Eds.), Vol. 2. Amsterdam University Press, 357–422.

Qian-Ping Gu and Hisao Tamaki. 2008. Optimal branch-decomposition of planar graphs in $O(n^3)$ Time. *ACM Trans. Algor.* 4, 3 (2008), 30:1–30:13.

Marijn Heule and Stefan Szeider. 2015. A SAT approach to clique-width. *ACM Trans. Comput. Log.* 16, 3 (2015), 24.

I. V. Hicks. 2002. Branchwidth heuristics. *Congr. Numer.* 159 (2002), 31–50.

Illya V. Hicks. 2005. Graphs, branchwidth, and tangles! Oh my! *Networks* 45, 2 (2005), 55–60.

Petr Hliněný and Sang-il Oum. 2008. Finding branch-decompositions and rank-decompositions. *SIAM J. Comput.* 38, 3 (2008), 1012–1032.

Kalev Kask, Andrew Gelfand, Lars Otten, and Rina Dechter. 2011. Pushing the power of stochastic greedy ordering schemes for inference in graphical models. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI'11)*, Wolfram Burgard and Dan Roth (Eds.). AAAI Press.

Neha Lodha, Sebastian Ordyniak, and Stefan Szeider. 2017. SAT-encodings for special treewidth and pathwidth. In *Proceedings of the 20th International Conference on Theory and Applications of Satisfiability Testing (SAT'17) (Lecture Notes in Computer Science)*, Serge Gaspers and Toby Walsh (Eds.), Vol. 10491. Springer, 429–445.

Arnold Overwijk, Eelko Penninkx, and Hans L. Bodlaender. 2011. A local search algorithm for branchwidth. In *Proceedings of the 37th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'11) (Lecture Notes in Computer Science)*, Ivana Cerná, Tibor Gyimóthy, Juraj Hromkovic, Keith G. Jeffery, Rastislav Královic, Marko Vukolic, and Stefan Wolf (Eds.), Vol. 6543. Springer, 444–454.

Umut Oztok and Adnan Darwiche. 2014. CV-width: A new complexity parameter for CNFs. In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI'14) Including Prestigious Applications of Intelligent Systems (PAIS'14) (Frontiers in Artificial Intelligence and Applications)*, Torsten Schaub, Gerhard Friedrich, and Barry O'Sullivan (Eds.), Vol. 263. IOS Press, 675–680.

Neil Robertson and P. D. Seymour. 1991. Graph minors X. Obstructions to tree-decomposition. *J. Comb. Theory Ser. B* 52, 2 (1991), 153–190.

Neil Robertson and Paul D. Seymour. 1991. Graph minors. X. Obstructions to tree-decomposition. *J. Comb. Theory, Ser. B* 52, 2 (1991), 153–190.

Marko Samer and Helmut Veith. 2009. Encoding treewidth into SAT. In *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing (SAT'09) (Lecture Notes in Computer Science)*, Vol. 5584. Springer Verlag, 45–50.

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

J. Cole Smith, Elif Ulusal, and Illya V. Hicks. 2012. A combinatorial optimization algorithm for solving the branchwidth problem. *Comp. Opt. Appl.* 51, 3 (2012), 1211–1229.

Dimitrios M. Thilikos, Maria J. Serna, and Hans L. Bodlaender. 2000. Constructive linear time algorithms for small cutwidth and carving-width. In *Proceedings of the 11th International Conference on Algorithms and Computation (ISAAC'00) (Lecture Notes in Computer Science)*, D. T. Lee and Shang-Hua Teng (Eds.), Vol. 1969. Springer, 192–203.

Elif Ulusal. 2008. *Integer Programming Models for the Branchwidth Problem*. Ph.D. Dissertation. Texas A&M University.

Eric Weisstein. 2016. MathWorld online Mathematics resource. Retrieved from http://mathworld.wolfram.com.

Received February 2018; revised November 2018; accepted April 2019