A SAT Approach to Branchwidth*

Neha Lodha, Sebastian Ordyniak and **Stefan Szeider**
Algorithms and Complexity Group, TU Wien, 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 develop a novel partition-based characterization of branch decompositions. 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 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 method scales now to instances with several thousands of vertices and edges.

1 Introduction

Branch decomposition is a prominent method for structurally decomposing a graph or hypergraph. This decomposition method was originally introduced by Robertson and Seymour [1991] 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 hyperedges). 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 [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]. 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 [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 [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 is NP-hard [Seymour and Thomas, 1994].

1.1 Contribution

In this paper we propose a practical SAT-based approach to finding a branch decomposition 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 $\leq w$. 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

*This is a shortened version of a paper that appeared in the Proc. of SAT 2016 [Lodha et al., 2016]. This work is dedicated to the memory of Helmut Veith (1971–2016).

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 develop 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.

1.2 Related Work

There exist SAT-based encodings for other width parameters such as treewidth [Samer and Veith, 2009; Berg and Järvisalo, 2014] and clique-width [Heule and Szeider, 2015].

For finding branch decompositions of smallest width, Robertson and Seymour [1991] suggested an exponential-time algorithm 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 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 of up to 50% [Robertson and Seymour, 1991] 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].

2 Preliminaries

2.1 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.

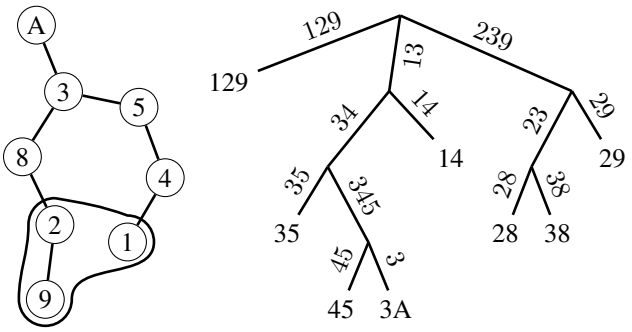

Figure 1: A hypergraph H (left) and an optimal branch decomposition (T,γ) of H (right). The labels of the leaves of T are the edges assigned to them by γ and the labels of the edges of T are the cut vertices of that edge.

2.2 Graphs and Branchwidth

We consider finite hypergraphs and undirected graphs. For basic terminology on graphs we refer to a standard text book [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 $E \subseteq E(H)$, we denote by $H \setminus E$ the hypergraph with vertices $V(H)$ and edges $E(H) \setminus E$.

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 $\delta(E) = \delta(E(H) \setminus E)$.

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 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 $\delta_{\mathcal{B}}(e)$ (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 $T \setminus \{e\}$. 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* of H is the minimum width over all branch decompositions of H (or 0 if $|E(G)| = 0$ and H has no branch decomposition). Fig. 1 illustrates a branch decomposition of a small hypergraph. In figures and in the remainder of the paper we will often denote a set $\{1,2,3,A\}$ of vertices as $123A$.

2.3 Partitions

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 $p = \bigcup_{i=1}^{k} p_i$.

3 An Encoding for 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, 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.

3.1 Partition-based Reformulation of Branchwidth

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 $E \in \bigcup_{i=1}^{l-1} P_i$. We will refer to P_i as the i-th *level* of the derivation $\mathcal{P}$ and we will refer to elements in $\bigcup_{i=1}^{l} P_i$ 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 $\mathcal{P} = (P_1, \ldots, P_5)$, where P_1 contains $\{129\}$, $\{35\}$, $\{45\}$, $\{3A\}$, $\{14\}$, $\{28\}$, $\{38\}$, $\{29\}$, P_2 contains $\{129\}$, $\{35\}$, $\{45, 3A\}$, $\{14\}$, $\{28\}$, $\{38\}$, $\{29\}$, P_3 contains $\{129\}$, $\{35, 45, 3A\}$, $\{14\}$, $\{28, 38\}$, $\{29\}$, P_4 contains $\{129\}$, $\{35, 45, 3A, 14\}$, $\{28, 38, 29\}$, and P_5 contains $\{129, 35, 45, 3A, 14, 28, 38, 29\}$. 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, 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 $\lfloor |E(H)|/2 \rfloor - \lceil w/e \rceil + \lceil \log \lfloor w/e \rfloor \rceil$.*

3.2 The Encoding

The partitioned-based characterization now gives rise to a natural encoding of branchwidth in terms of a CNF formula. Namely, we construct a CNF formula $F(H, w)$ that has $\mathcal{O}(m^3 + m^2n^2)$ variables and $\mathcal{O}(m^4n + m^2n^2)$ clauses, which is satisfiable if and only if the hypergraph H has branchwidth at most w.

4 Local Improvement

The encoding presented in the previous section 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.

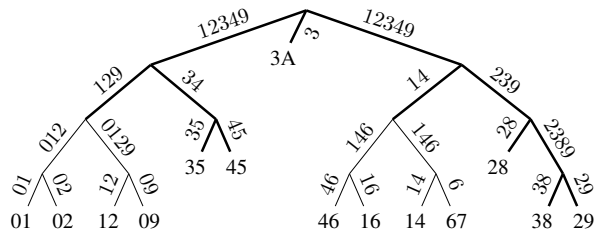

Figure 2: A branch decomposition $\mathcal{B}$ of the graph H given in Fig. 4 together with an example of a local branch decomposition $\mathcal{B}_L$ (bold edges) chosen by our algorithm.

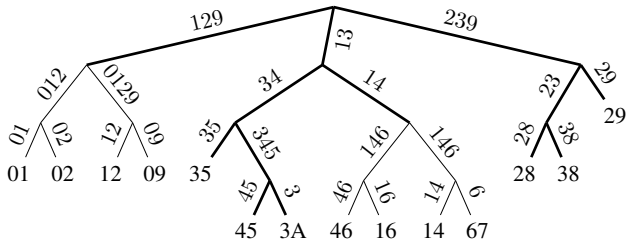

Figure 3: The improved branch decomposition $\mathcal{B}'$ obtained from $\mathcal{B}$ after replacing the local branch decomposition $\mathcal{B}_L$ of $H(T_L)$ with an optimal branch decomposition $\mathcal{B}'_L$ of $H(T_L)$ obtained from our SAT encoding. See Fig. 2 for an illustration of $\mathcal{B}$ 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 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 approachis 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

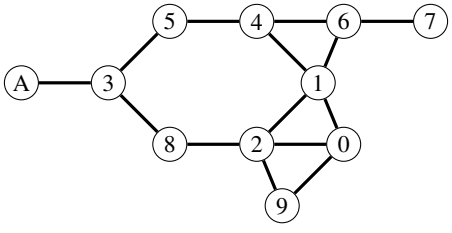

Figure 4: The graph H used to illustrate the main idea behind our local improvement procedure.

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. 3.

5 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 [Weisstein, 2016], graphs from TreewidthLIB [Bodlander, 2016] which origin from a broad range of applications. 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 ran 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.

5.1 Single SAT Encoding

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 almost all of the famous graphs, for some of them the branchwidth was nor known. A precise comparison to the other two known approaches for computing branchwidth, i.e., Hick's [2005] combinatorial algorithm based on tangles, and Ulusal's [2008] integer programming encoding, is difficult since their implementations are not available to us and their experiments were performed on much older hardware. It can be inferred that our encoding scales much better with branchwidth than the other two approaches, which is crucial for its use within the local improvement approach.

5.2 SAT-Based Local Improvement

We tested our local improvement method on graphs with several thousands of vertices and edges and with initial branch decompositions of width up to hundred and more. 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 980 graphs with up to 33810 vertices and 121275 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 computed the initial branch decomposition by a heuristic provided to us by Hicks [2005].

From the 980 graphs, our SAT-based local improvement algorithm could improve the width of the initial branch decomposition for 470 graphs. In some cases the improvement was significant. Table 1 shows the graphs with the best improvement.

In summary, our experiments show that the SAT-based local improvement approach scales well to large graphs with several thousands of vertices and edges, and with an initial branchwidth 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.

Graph	$\lvert V\rvert$	$\lvert E\rvert$	iw	w	d
bn_51	661	2131	95	75	20
bn_77	1020	2616	40	21	19
bn_18-pp	645	2991	66	51	15
bn_52	661	2131	88	74	14
bn_57-pp	387	1330	65	52	13
graph11	340	1425	105	92	13
d493.tsp-pp	488	1452	34	22	12
bn_60-pp	426	1489	80	69	11
bn_63-pp	426	1489	73	62	11
vm1084.tsp-pp	808	2312	29	19	10
inithx.i.2-pp	363	8897	55	45	10
fl3795.tsp	3795	11326	49	39	10

Table 1: 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 and d is the difference between iw and w.

6 Final Remarks

We have presented a first SAT encoding for branchwidth and introduced the new method of SAT-based local improvement 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 for 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 [Berg and Järvisalo, 2014]. 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., a small number of maximum cuts. This could propagate into adjacent local improvement steps and yield an overall branch decomposition of smaller width.

Acknowledgements

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

[Alekhnovich and Razborov, 2002] Michael Alekhnovich and Alexander A. Razborov. Satisfiability, branch-width and tseitin tautologies. In *43rd Symposium on Foundations of Computer Science (FOCS 2002), 16-19 November 2002, Vancouver, BC, Canada, Proceedings*, pages 593–603. IEEE Computer Society, 2002.

[Bacchus *et al.*, 2003] Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi. Algorithms and complexity results for #sat and bayesian inference. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*, pages 340–351. IEEE Computer Society, 2003.

[Berg and Järvisalo, 2014] 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.

[Bodlander, 2016] Hans Bodlander. TreewidthLIB a benchmark for algorithms for treewidth and related graph problems, 2016. http://www.staff.science.uu.nl/~bodla101/treewidthlib/.

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

[Diestel, 2000] Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer Verlag, New York, 2nd edition, 2000.

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

[Grohe, 2008] 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.

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

[Hicks, 2002] I.V. Hicks. Branchwidth heuristics. *Congr. Numer.*, 159:31–50, 2002.

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

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

[Kask *et al.*, 2011] 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.

[Overwijk *et al.*, 2011] 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.

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

[Samer and Veith, 2009] 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.

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

[Ulusal, 2008] Elif Ulusal. *Integer Programming Models for the Branchwidth Problem*. PhD thesis, Texas A&M University, May 2008.

[Weisstein, 2016] Eric Weisstein. MathWorld online mathematics resource, 2016. http://mathworld.wolfram.com.