The PACE 2019 Parameterized Algorithms and Computational Experiments Challenge: The Fourth Iteration

M. Ayaz Dzulfikar[1]
University of Indonesia, Kota Depok, Jawa Barat 16424, Indonesia
muhammad.ayaz@ui.ac.id

Johannes K. Fichte
Faculty of Computer Science, TU Dresden, 01062 Dresden, Germany
johannes.fichte@tu-dresden.de

Markus Hecher
Institute of Logic and Computation, TU Wien, Favoritenstraße 9-11, 1040 Wien, Austria
hecher@dbai.tuwien.ac.at

Abstract

The organizers of the 4th Parameterized Algorithms and Computational Experiments challenge (PACE 2019) report on the 4th iteration of the PACE challenge. This year, the first track featured the MinVertexCover problem, which asks given an undirected graph $G = (V, E)$ to output a set $S \subseteq V$ of vertices such that for every edge $vw \in E$ at least one endpoint belongs to S. The exact decision version of this problem is one of the most discussed problem if not even the prototypical problem in parameterized complexity theory. Another two tracks were dedicated to computing the hypertree width of a given hypergraph, which is a certain generalization of tree decompositions to hypergraphs that has widely been applied to problems in databases, constraint programming, and artificial intelligence. On one track we asked for submissions that compute hypertree decompositions of minimum width (MinHypertreeWidth) and on the other track we asked to heuristically compute hypertree decompositions of small width quickly (HeurHypertreeWidth). We received 28 implementations from 26 teams. This year we asked participants to submit solver descriptions in order to count as a submission for the challenge. We received those from 16 teams with overall 33 participants from 10 countries. One team submitted successful solutions to all three tracks.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Complexity theory and logic; Mathematics of computing → Solvers; Mathematics of computing → Graph algorithms; Mathematics of computing → Hypergraphs

Keywords and phrases Parameterized Algorithms; Vertex Cover Problem; Hypertree Decompositions; Implementation Challenge; FPT

Digital Object Identifier 10.4230/LIPIcs.IPEC.2019.2

Funding The work has been supported by the Austrian Science Fund (FWF), Grants Y698 and P26696, and the German Science Fund (DFG), Grant HO 1294/11-1 and Erasmus+ KA107. The third author is also affiliated with the University of Potsdam, Germany.

Acknowledgements The PACE challenge was supported by *Networks* [5], an NWO Gravitation project of the University of Amsterdam, Eindhoven University of Technology, Leiden University and the Center for Mathematics and Computer Science (CWI), by the *Centre for Information and High Performance Computing (ZIH) of TU Dresden* [3], and by *data-experts* [1]. The prize money (4,000 EUR) was given through the generosity of Networks and data experts. We are grateful to Szymon Wasik and Jan Badura for the fruitful collaboration and for hosting the challenge at optil.io [154]. We like to acknowledge the generous support by the High Performance Computing Center at TU

[1] The author contributed significantly to the realization of PACE 2019 during his internship at TU Dresden.

14th International Symposium on Parameterized and Exact Computation (IPEC 2019).
Editor: Bart M.P. Jansen and Jan Arne Telle; Article No. 2; pp. 2:1–2:23
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Dresden, who gave us access to the HRSK-II and 80.000 CPU hours from February on to select instances and validate the results [3].

1 Introduction

The Parameterized Algorithms and Computational Experiments Challenge (PACE) was conceived in Fall 2015 to deepen the relationship between parameterized algorithms and practice. It aims to:

1. Bridge the divide between the theory of algorithm design and analysis, and the practice of algorithm engineering.
2. Inspire new theoretical developments.
3. Investigate in how far theoretical algorithms from parameterized complexity and related fields are competitive in practice.
4. Produce universally accessible libraries of implementations and repositories of benchmark instances.
5. Encourage the dissemination of these findings in scientific papers.

The first iteration of PACE was held at IPEC 2016 [40]. There programmers were asked for submissions on two tracks, namely, (a) a treewidth track allowing for exact sequential, exact parallel, heuristic sequential, and heuristic parallel submissions and (b) a feedback vertex set track. PACE 2017 [41] featured (a) a treewidth track allowing for sequential exact or heuristic submissions and (b) a minimum fill-in track. PACE 2018 [26] then asked for submissions that solve the Steiner tree problem. The line of past challenges has inspired a long list of works on the proposed problems [8, 17, 20, 57, 62, 76, 81, 90, 97, 98, 107, 123, 142, 153, 143, 144, 151, 152, 99]. Benchmarks from the PACE challenges have been used for other competitions and evaluations [47, 129, 134]. Various applications have built on top of results from PACE or were inspired by the success of solvers produced for PACE [16, 21, 22, 30, 32, 33, 58, 59, 60, 61, 64, 65, 66, 101, 103, 105, 110, 109, 111, 122, 127, 150, 158]. Finally, PACE challenges have been mentioned in research works [38, 115]. Among the various papers have also been papers that received a best paper award [16, 143].

In this article, we report on the 4th iteration of PACE. The PACE 2019 challenge was announced on November 16, 2018. Format descriptions were posted on November 20, 2018 and the public instances were released on December 5, 2019 (hypertree decompositions) January 4, 2019 (vertex cover) and updated on March 6, 2019 (vertex cover), since the initial instances were not challenging enough for the participants. The final version of the submissions was due on May 6, 2019. We informed the participants of the results on July 14. We released the private instances on July 29 [50, 56] and announced the final results to the public on September 11, during the award ceremony at the International Symposium on Parameterized and Exact Computation (IPEC 2019) in Munich. For the first time in the history of PACE, we had a poster session after the award ceremony, where 4 posters were presented, namely WeGotYouCovered [92], bogdan [156], asc [139] as well as TULongo [124, 125].

PACE 2019 consists of three tracks. The first track featured the MinVertexCover problem, which asks given an undirected graph $G = (V, E)$ to output a set $S \subseteq V$ of vertices such that for every edge $vw \in E$ at least one endpoint belongs to S. The exact decision version of this problem is one of the most discussed problem if not even the prototypical problem in parameterized complexity theory. Another two tracks were dedicated to compute the hypertree width of a given hypergraph, which is a certain generalization of tree decompositions to hypergraphs that has widely been applied to problems in databases [82], constraint programming [35, 82, 87], and artificial intelligence [106, 108]. On one track we asked for submissions that compute hypertree decompositions of minimum width (MinHypertreeWidth) and on the other track we asked to heuristically compute hypertree decompositions of small width fast (HeurHypertreeWidth). This year's PACE had quite relaxed solver requirements and we even allowed solvers to use external dependencies such as ILP, SAT, and SMT solvers if the external solvers were available under an open source license. We received 28 implementations from 26 teams. This year we asked participants to hand in solver descriptions and received those from 16 teams. If we count only submissions that handed in a solver and a description according to the submission requirements, we had overall 33 participants from 10 countries. One outstanding team submitted successful solutions to all three tracks.

2 The PACE 2019 Challenge Problems

In this section, we give an overview on the PACE 2019 problems. We organized the section by problem and present the well-known vertex cover problem first and then a generalization of treewidth to hypergraphs. For each problem, we start with a quick definition, introduce the tracks and selected instances. We finish with the submission requirements. In the following, we assume that the reader is familiar with basic graph terminology and we refer to standard texts [25, 45] otherwise.

2.1 Vertex Cover (Track 1a)

Computing minimum vertex covers was among the original 21 NP-complete problems by Karp [104] and is probably one of the most famous graph problems. In fact, there are over 537,000 results (queried on 30.07.2019) on Google scholar, dealing with problems related to finding vertex covers and variants thereof. Besides, vertex covers are particularly well-studied in parameterized complexity [37], ranging from studies involving different parameters [31, 126] and related problems [121, 137], over kernelization [52, 112], and also concern concrete applications [28, 46, 51, 96, 131]. We use the following definition.

▶ **Definition 1** (Vertex Cover). *Given an undirected graph $G = (V, E)$. A set $S \subseteq V$ is a* vertex cover *for G, if for every edge $uv \in E$, we have $\{u, v\} \cap S \neq \emptyset$. A vertex cover S is a minimum vertex cover for G if there is no vertex cover S' for G such that $|S'| < |S|$.*

This definition motivates the problem of Track 1a.

Problem:	MinVertexCover (Exact)
Input:	Undirected graph G
Task:	Output a minimum vertex cover for G.

Data Format

The input format for providing a graph (*.gr*) was taken from the PACE 2017 format for graphs [41]. The output format for specifying a vertex cover (*.vc*) was an adaption of the input format in the same style. More details on the format can be found at `pacechallenge.org/2019/vc/vc_format`. There is also a simple checker available at `github.com/hmarkus/vc_validate` in Python (folder: `vc_validate`) and C++ (folder: `cpp`).

	Name	#	Reference (Download Link)	Converter
1	`PACE 2016/Treewidth`	17	`pacechallenge.org:2016`	
2	`TransitGraphs`	23	`github:daajoe/transit_graphs`	[53]
3	`Road-graphs`	5	`github.com:ben-strasser`	
4	`SNAP`	15	`snap.stanford.edu`	
5	`frb`	41	`buaa.edu.cn:kexu/benchmarks`	
6	`ASP Horn backdoors`	1,077	`asparagus.cs.uni-potsdam.de`	[23, 24, 78, 102, 54]
7	`SAT Horn backdoors`	83	`marco.gario.org`	[74, 55]
			`tinyurl.com:countingbenchmarks`	
8	`SAT2VC`	8,329	`satlib.org`	[49]
			`tinyurl.com:countingbenchmarks`	
			`sat2018.tuwien:benchmarks`	

Table 1 Information on the origins of our graphs, including download links and relevant converters.

Instances for Vertex Cover

In order to establish a suitable set of benchmark instances from various areas, we considered in total 9,591 instances comprising of the following 8 origins.

1. 17 graphs from `PACE 2016/Treewidth` [40];
2. 23 graphs from `TransitGraphs` (Denmark, FlixBus, Israel, Luxembourg, Metro Bilbao, Mexico City, NYC Subway, Pace Bus, Praha, Translink, VBB, WienerLinien) [53];
3. 5 graphs from `Road-graphs` [42, 141];
4. 15 graphs from `SNAP` (Stanford Network Analysis Project) [117] including gnutella [120, 135], social circles from facebook [130], bitcoin OTC trust network [113, 114], and wiki vote [118, 119],
5. 41 graphs from `frb` [155];
6. 1,077 graphs from `ASP Horn backdoors` [63], where a vertex cover of the graph gives a Horn backdoor to the considered logic program (ASP) [27, 100], originating from various ASP competitions [10, 29, 43, 79], in more detail, 7 *SCoreDLP-Mutex* [128] logic programs, 282 *SCore-RLP200* logic programs [157], 200 *MinimalDiagnosis* [29, 77] logic programs, and 588 *automotive* [140] logic programs;
7. 83 graphs from `SAT Horn backdoors` [138], where a vertex cover of the graph gives a Horn backdoor to the considered SAT instance, originating from 15 *dfremont projection* formulas [71] and 68 *Gario* formula collection [73, 75];
8. 8,329 graphs from `SAT2VC`, where we took widely used SAT instances, reduced them into 3-SAT instances [133] and then reduced them to k-vertex cover by means of the well-known reductions by Karp [104] (merge the reductions from SAT to clique and then clique to vertex cover into one). Then, a vertex cover of $k = n + 2m$ exists if and only if the given formula of n variables and m clauses is satisfiable. We took 6,956 SAT instances from the *SATlib* collection [95], 1,187 instances from a popular *#SAT* collection [66, 71], and 186 instances from the *SAT 2018 competition* [94].

We implemented a variety of converters, among those tools where the following: asp_horn_backdoors [54], HornVCBuilder [74], lp2normal [23, 24], gringo [78, 102], a SAT to vertex cover converter [49]. Table 1 gives an overview on the sources and the involved graph converters or programs that implement the reductions described above.

For PACE 2019, we were interested in challenging, large instances that are still within reach for the participants in reasonable development time, but require reasonable efforts to

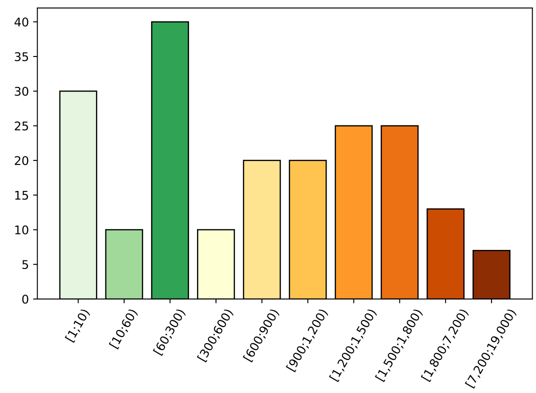

Figure 1 Overview on the number of instances and runtime intervals, which were computed by means of a simple ILP encoding solved by the (M)ILP solver Gurobi, for the selected instances of Track 1a. The x-axis labels the considered intervals, i.e., $[a; b)$ indicates that runtime t was within the interval $a \leq t < b$. The y-axis indicates the number of selected instances.

score one of the medals. In order to get a naive classification of the "practical hardness" of our collected benchmark instances, we encoded the MinVertexCover problem into a simple ILP encoding and ran the solver Gurobi [89] on all instances with a timeout of 6 hours. After obtaining initial runtime results, we assigned to each instance a category (easy$\{1, 2, 3\}$, medium$\{1, 2, 3, 4\}$, and hard$\{1, 2, 3\}$). From the classified instances, we picked 200 instances by sampling uniformly at random among the distribution given in Figure 1. Mainly, we dropped an instance if the runtime was below 1s, and picked 80 instances from category easy (runtime within the interval $[1; 300)$), picked 80 instances from category medium (runtime within the interval [300;1,500), and 40 instances from category hard (runtime in the interval [1,500;19,000)). We dropped instances that could not be solved within 6 hours. We numbered the instances from 1 to 200 with increasing hardness, selected the odd numbered instances as public and even numbered instances as private instances. Table 2 shows statistics on the resulting instances. The 100 public instances were released at `pacechallenge.org/2019/vc/vc_exact`. All the selected instances are publicly available at `Zenodo:3368306` [50]. We released the full collection of instances, instance selection scripts, and mapping of selected instances at `github:daajoe/pace_2019_vc_instances`.

2.2 Hypertree Decompositions (Tracks 2a and 2b)

Hypertree decompositions and the resulting measure hypertree width is a prominent structural restriction in the area of constraint satisfaction problems (CSP) [11, 39, 149] and databases [82], such as the commercial database system LogicBlox [12, 15, 6, 7, 132] that uses hypertree decompositions. In the beginning of the 80s, Freuder [72] showed that the CSP is tractable under structural restrictions imposed in terms of bounded treewidth of the constraint graph. The result has later been generalized by Gottlob, Leone, and Scarcello to hypertree width [82], which still renders CSP polynomial-time tractable. In fact, the polynomial-time solvability for bounded hypertree width instances still holds when one is interested in counting the number of satisfying assignments [48], which is also known as sum-of-products, weighted counting, partition function, or probability of evidence [106]. Thus, this problem is also of high interest in artificial intelligence, e.g., to solve probabilistic reasoning [108]. While there are even more general measures [86, 87], hypertree decompositions allow for computing a hypertree decomposition of width at most k (if one exists) in polynomial time for a given fixed integer k. Still hypertree width was mainly of theoretical

instances	$\lvert V_{\min}\rvert$	$\lvert V_{\max}\rvert$	$\lvert V_{\text{avg}}\rvert$	$\lvert V_{\text{med}}\rvert$	$\lvert E_{\min}\rvert$	$\lvert E_{\max}\rvert$	$\lvert E_{\text{avg}}\rvert$	$\lvert E_{\text{med}}\rvert$	$\text{tw}^{\text{ub}}_{\text{med}}$
public	198	138.14k	16.44k	14.69k	813	227.24k	30.95k	24.66k	105.0
private	153	98.13k	16.30k	13.59k	625	161.36k	30.50k	27.15k	103.5
all	153	138.14k	16.37k	13.59k	625	227.24k	30.73k	24.66k	107.0

Table 2 Basic statistics on the selected PACE 2019 instances for Track 1a (Vertex Cover/Exact). $|V_{\min}|$ and $|E_{\min}|$ refers to the minimum number of vertices and edges, respectively; max refers to the maximum; avg refers to the mean; med refers to the median; $\text{tw}^{\text{ub}}_{\text{med}}$ refers to a heuristically computed upper bound (median over the instances) on the treewidth using the library htd [8].

interest due to few practical implementations and missing efficient implementations of heuristics to compute the associated decompositions. The success of PACE 2016 and 2017 and its resulting decomposers for computing tree decompositions, which facilitated lots of follow-up implementations [32, 33, 66, 61], and the hope that PACE also drives advances to the CSP, reasoning, and database community, motivated us to propose this problem for Track 2.

▶ **Definition 2** (Hypergraphs and Tree Decompositions)**.** *A* hypergraph *is a pair $H = (V, E)$ consisting of a set V of* vertices *and a set E of* hyperedges*, where each hyperedge in E is a subset of V. Let $H = (V, E)$ be a hypergraph. A* tree decomposition *[136] of H is a pair $\mathcal{T} = (T, \chi)$ where $T = (N, A)$ is a rooted tree and χ is a mapping that assigns to each node $t \in N$ a set $\chi(t) \subseteq V$, called* bag*, such that the following conditions hold: (i) $V = \bigcup_{t\in N} \chi(t)$, (ii) $E \subseteq \{2^{\chi(t)} \mid t \in N\}$, and (iii) for each $r, s, t \in A$ where s lies on the path from r to t, we have $\chi(s) \subseteq \chi(r) \cap \chi(t)$.*

We follow the definitions of Gottlob, Leone, and Scarcello [82].

▶ **Definition 3** (Hypertree Decompositions [82])**.** *Let $H = (V, E)$ be a hypergraph and let $S \subseteq V$ be a set of vertices. An* edge cover *$C \subseteq E$ of S is a set of hyperedges, where for every $v \in S$, there is $e \in C$ with $v \in e$. A* hypertree decomposition *of H is a triple $\mathcal{H} = (T, \chi, \lambda)$, where (i) (T, χ) is a tree decomposition of H with $T = (N, A)$, (ii) λ is a mapping that assigns to each node $t \in N$ an edge cover $\lambda(t)$ of $\chi(t)$, and (iii) for every $t \in N$ and every $e \in \lambda(t)$, we have $e \cap \chi_{\leq t} \subseteq \chi(t)$. The set $\chi_{\leq t}$ refers to the set of all vertices occurring in a bag $\chi(t')$ of the subtree $T' = (N', A')$ of T where T' is rooted at t and $t' \in N'$. Then,* $\text{width}(\mathcal{H})$ *is the size of the largest edge cover $\lambda(t)$ over all nodes $t \in N$. The* hypertree width $\text{htw}(H)$ *is the smallest width over all hypertree decompositions of H.*

Based on this generalization of tree decompositions to hypergraphs, we defined the following two problems for Track 2a and 2b.

Problem:	MinHypertreeWidth (Exact)
Input:	Hypergraph H
Task:	Output a hypertree decomposition of H of minimum width.

Problem:	HeurHypertreeWidth (Heuristic)
Input:	Hypergraph H
Task:	Output a hypertree decomposition of H of small width.

Computation of Hypertree Decompositions

Above we mentioned that there are a variety of applications for hypertree decompositions. However, many practical sides are not very well explored. In fact, for tree decompositions

Track	instances	$\|V_{\text{min}}\|$	$\|V_{\text{max}}\|$	$\|V_{\text{med}}\|$	$\|E_{\text{min}}\|$	$\|E_{\text{max}}\|$	$\|E_{\text{med}}\|$
Track 2a (Exact)	public	3	130	24.0	3	100	61.5
Track 2a (Exact)	private	10	351	25.0	5	250	60.0
Track 2a (Exact)	all	3	351	24.0	3	250	60.0
Track 2b (Heuristic)	public	12	694	40.0	5	526	84.0
Track 2b (Heuristic)	private	12	694	40.0	5	495	90.0
Track 2b (Heuristic)	all	12	694	40.0	5	526	84.0

Table 3 Basic statistics on the selected PACE 2019 instances for Track 2a (MinHypertreeWidth) and Track 2b (HeurHypertreeWidth). $|V_{\text{min}}|$ and $|E_{\text{min}}|$ refers to the minimum number of vertices and hyperedges, respectively; max refers to the maximum; med refers to the median.

both exact as well as heuristic-based decomposers are widely available due to recent PACE challenges, this is not the case for hypertree decompositions. There, only very few implementations are available and the exact implementations are highly prototypical. Fortunately, various theoretical results on computing hypertree decompositions [82, 83, 70, 69, 68] and more general measures [57] are available. Some of these approaches are simply combinatorial backtracking based algorithms, others are heuristics based on bucket elimination, and again others are based on encodings into extensions of SAT. A major obstacle for hypertree decompositions is that Condition (iii) of Definition 3 is expensive and in case of encodings into SAT-related formalisms it blows up the size computation considerably.

Data Format

We designed the input and output format by extending the PACE 2016 formats used for graphs and tree decompositions [40, 41], which are similar to the format used by DIMACS challenges [4]. The input format for hypergraphs (*.hgr*) extends the PACE 2016/2017 graph format to edges of arbitrary arity. The output format for hypertree decompositions (*.htd*) allows in addition to the treewidth format to specify a covering function, i.e., mappings for the bags that map hyperedges to 0 or 1. More details on the format can be found at `https://pacechallenge.org/2019/htd/htd_format/`. We provided a simple checker at `https://github.com/daajoe/htd_validate` in Python (folder: `htd_validate`) and C++ (folder: `cpp`). Both tools already implement reading and outputting the formats.

Instances for Hypertree Decompositions

For our benchmark selection, we considered 2,191 instances, which contain hypergraphs that originate from CSP instances and conjunctive database queries from various sources. All of these instances are part of the hyperbench collection and have been collected and published by Fischl et al. [68, 69] together with different hypergraph properties including various notions of width related measures[2]. The selection consists of eight non-disjoint sets. 15 instances from the set `DaimlerChrysler`, 12 instances from the set `Grid2D`, 24 instances on circuits from the set `ISCAS'89` [85]. 31 instances from `MaxSAT` [19]. 1090 instances and 863 instances, respectively on `csp_application` and `csp_random` of instances from the well

2 The hypergraphs together with the properties have been published at `http://hyperbench.dbai.tuwien.ac.at` and the collection of hypergraphs is also available in a public data repository [57].

known XCSP benchmarks [14]. 82 instances from the set `csp_other`, which have been collected for works on hypertree decompositions[3]. 156 instances from the set `CQ` on various conjunctive queries [13, 18, 80, 88, 116, 145].

In order to obtain a basic classification of the instances we heuristically computed hypertree decompositions with htdecomp [44] and computed generalized hypertree decompositions of smallest width [57]. Generalized hypertree decompositions relax Condition (iii) in Definition 3 and allow certain techniques to find a solution faster. The widths of the thereby obtained decompositions are indeed of minimum width, which is guaranteed by comparing the widths with the matching integer lower bounds obtained by fraSMT [57] – a tool for computing fractional hypertree decompositions, which are more general than hypertree decompositions. After obtaining initial runtime results we assigned to each instance one category out of easy, medium, hard, or dropped the instance. An instance was classified as easy if it could be solved within 60s, as medium if it could be solved within 300 and 900s, and hard if it could not be solved within 7,200s. We dropped instances that could not be solved within 7,200s on purpose, since we were interested in many realistic instances and quite challenging instances for the solvers. From the remaining and classified instances we picked 200 by sampling 20 instances uniform at random from category easy, 60 instances from the category medium, and 120 instances from the category hard. Table 3 shows statistics on the resulting selected instances. We released the public instances at `https://pacechallenge.org/2019/htd/htd_exact/` and `https://pacechallenge.org/2019/htd/htd_heur/`, respectively. The pages also contain a document that contains the mapping of the selected PACE 2019 instances and the original instance of hyperbench. We also published the instances in a public data repository [56].

3 Challenge Settings

In the following, we state the submission requirements for this year's PACE and basic information on the system on which we ran the challenge.

Submission Requirements

We invited people to participate in the three proposed tracks. In order to have common setting we however posted the following *submission requirements*.

1. Both submitted solver and external dependencies have to be *open source*.
2. The source code of the solver is maintained by the submitters on a *public repository*.
3. A *dedicated solver description* of at least two pages has to be submitted.

We choose Requirement 1 fairly permissive, in order to obtain valuable information on the actual efficiency of solving the problem. So we did not prescribe the algorithmic paradigm that had to be used. In that way, we also allowed in principle submissions that relied on an encoding into paradigms such as SAT or SMT. We imposed Requirement 3 to enable other researchers to analyze and compare implemented ideas, get insights into correctness of the other solvers, provide theoreticians with basics ideas on the latest implementations and in the hope to improve on the reproducibility of the submitted solver.

In addition, we imposed another main rule for the exact tracks.

E. We expected submissions to be based on a provably optimal algorithm.

[3] `www.dbai.tuwien.ac.at/proj/hypertree/benchmarks.zip`

Region	Country	Teams	Participants	Tracks
Europe	Austria	3	3	1a, 2a, 2b
	Germany	4	7	1a
	Hungary	1	2	1a
	Norway	2	6	1a
	Poland	2	4	1a
	Russia	1	1	1a
	Scotland	1	2	1a, 2a, 2b
Middle East	Lebanon	1	2	1a
North America	USA	1	1	1a
South Asia	India	2	5	1a
	10	18	33	

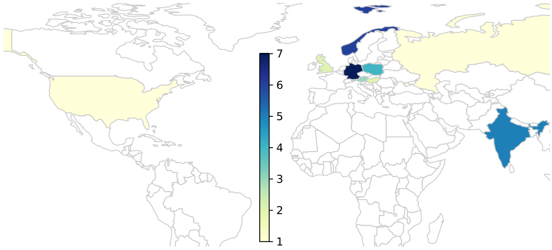

Figure 2 Participation per country based on the easychair registration and submission of both a solver and a description. Details are given by country, tracks, and team (left) and illustrated on the world map (right). Note that more teams and participants uploaded their code on optil.io

While we did not formally check Requirement E, we picked only instances from which we knew the size of a minimum vertex cover or the hypertree width, respectively and checked whether the output was both correct and according to our expected size. If a submission halted on some instance within the allotted time, but produced a solution that was known to be non-optimal, the submission was disqualified.

Limits

Since our evaluation resources were limited and we were interested in the solving behavior on a larger number of instance while allowing the participants to have a “training” phase on public instances, we restricted the runtime to 1,800 seconds and the available main memory to 8GB per instance. Note that in general, a solver is considered to be better than an other solver, if it solves more instances faster than the other solver. For more details about evaluation (criteria), we refer to Section 4.3.

Hardware

Our results were gathered on the cloud evaluation platform optil.io [154] running libc 5.4.0. optil.io evaluates submissions on Intel Xeon CPU E5-2695 v3, which consist of 14 cores running at 2.30GHz. Each submission had access to one core. Since the submissions ran in docker containers and the CPU has only 4 memory channels [2] we repeated the final evaluation 3 times and took the average.

4 Participants and Results

This year, we had 18 teams and 33 participants coming from 10 countries and four regions: Austria, Germany, Hungary, India, Lebanon, Norway, Poland, Russia, Scotland, and USA. Figure 2 provides an overview. The number of teams and participants were little less than half compared to PACE 2018 and at a similar number to PACE 2017. To be precise, the 2019 numbers above correspond to teams and participants who sent a final implementation and a solver description in time whereas in previous challenges registrations were carried out prior to optil.io registration. If we also count people who uploaded some code on the optil.io platform but dropped out of the challenge, the number of teams is 28, which is however also a much smaller number than in the 2018 iteration.

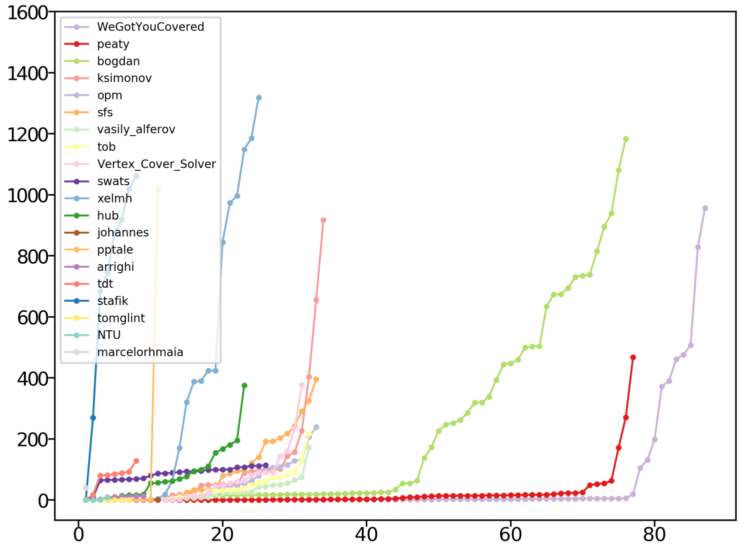

Figure 3 Runtime illustrated as cactus plot for `Track 1a (Vertex Cover/Exact)`. The x-axis labels consecutive integers that identify instances. The y-axis depicts the runtime. The instances are ordered by running time, individually for each solver.

4.1 Track 1a (Vertex Cover/Exact)

Figure 3 illustrates runtime results for all submitted solvers as cactus plot. Table 4 gives a detailed overview on the standings and solvers. We allowed each solver 30 minutes per instance and measured the number of solved instances. If two solvers solved the same number of instances we would in addition also take into account the runtime needed to solve those instances.

Winning Team. Demian Hespe and Sebastian Lamm (both: Karlsruhe University of Technology, Germany), Christian Schulz (University of Vienna, Austria), and Darren Strash (Hamilton College, USA) won Track 1a by solving 87 private instances in overall 1.3 hours and 52.7 seconds solving time on average. Their implementation (WeGotYouCovered) [92, 93] builds on a portfolio of techniques, which include an aggressive kernelization strategy with all known reduction rules, local search, branch-and-reduce, and a state-of-the-art branch-and-bound solver. Surprisingly, they also use several techniques that were not from the literature on the vertex over problem, but originally published to solve the (complementary) maximum independent set and maximum clique problems.

Runner-up. Patrick Prosser and James Trimble from the University of Glasgow, Scotland, scored second with their solver Peaty by solving 77 private instances. In fact, the results looked much closer on the public instances [148]. Interestingly, they also used intensive kernelization, local search, improved branch-and-bound, and a branch-and-bound maximum clique solver, and in addition an exact graph colouring algorithm which can quickly prove the optimality of a solution for some graphs.

Third Place. Sándor Szabó (University of Pecs, Hungary) and Bogdán Zaválnij (Hungarian Academy of Sciences, Hungary) accomplished a safe third place, which was in fact very close to the team that obtained the second place on the private instances. However, on the public instances they solved 9 less than the authors of Peaty. The authors used kernelization and a maximum clique solver by taking the complement graph. The maximum clique solver is based on progressive k-clique search by starting from a heuristically computed maximum clique and increasing until no clique of size k is found.

POS	Solver	#	$\#^{all}$	TLE	RTE	$t_{sum}[h]$	$t_{avg}[s]$	Source	Reference
1	WeGotYouCovered	87	169	13	0	1.3	52.7	H:sebalamm/pace-2019	[92, 93]
2	Peaty	77	157	23	0	0.4	20.6	H:jamestrimble/peaty	[148]
3	bogdan	76	147	24	0	4.5	215.2	H:zbogdan/pace-2019	[156]
4	ksimonov	34	73	64	2	1.0	102.1	L:seemann9/pace-2019-vc	[36]
5	opm	33	75	67	0	0.4	47.8	Z:3236867#.XW_J9S2B3x8	[67]
6	sfs	33	74	65	2	0.8	87.0	L:cg_pace2019/vertex_cover	[34]
7	vasily_alferov	32	70	68	0	0.2	23.2	H:vasalf/cheburashka	[9]
8	hub	23	54	68	9	0.5	79.5	H:hubhegnel/pace-2019	[91]
DSQ	Vertex_Cover_Solver	31	69	68	0	0.4	52.1	H:karamkontar99/Vertex-Cover-Solver	

Table 4 Detailed standings of the submitted solvers that solved at least 15 instances for `Track 1a (Vertex Cover/Exact)`. POS refers to the position of the solver where DSQ refers to a disqualification due to a produced wrong answer. # refers to the number of solved private instances and $\#^{all}$ refers to the number of all instances. TLE refers to the number of instances were the runtime limit was exceeded. RTE contains the number of instances on which we observed a runtime error. Note that if the three sums do not add to 100 we observed a memory overflow on the remaining ones. $t_{sum}[h]$ states the cumulative runtime over all all solved instances in hours, $t_{avg}[s]$ contains the average runtime over all solved instances in seconds. In column source, the character H abbreviates github, character L abbreviates gitlab, and Z refers to Zenodo.

4.2 Track 2a (Hypertree Width/Exact)

Table 5a summarizes runtime results for all submitted solvers. We allowed each solver 30 minutes per instance and measured the number of solved instances. Much to our regret we received only 3 submissions. We guess that hypertree width is just yet not very popular in the parameterized complexity community.

Winning Team. André Schidler and Stefan Szeider from TU Wien, Austria, won this year's Track 2a by solving 69 private instances in overall 1.4 hours at an average of 69.4 seconds when considering the solved instances. Their implementation (asc) [139] uses an incremental SMT-solving approach. There a first-order logic solver (handling arithmetic constraints) interacts with a SAT solver. Hypertree width and a more general parameter (generalized hypertree width) share Conditions (i) and (ii) from Definition 3. The additional Condition (iii) which is present for hypertree width (special condition), however, blows up the encoding size with a cubic number of clauses resulting in extremely large encodings for generalized hypertree decompositions and very long encoding times. For that reason, the authors implement a two-phase approach. They first use an encoding to obtain a generalized hypertree decomposition and try convert it into a hypertree decomposition satisfying the special condition without increasing the width. Only if that fails, they use the full encoding that includes the special condition.

Runner-up. Davide Mario Longo from TU Wien, Austria, scored second with his solver (TULongo/HdSolveE) by solving 31 private instances in 0.8 hours at an average runtime of 95.9 seconds over the solved instances. He used a combination of algorithms to compute lower bounds by obtaining generalized hypertree decompositions and then running a backtracking-based algorithm to determine a hypertree decomposition. Surprisingly, he solved much less public than private instances.

Third Place. Patrick Prosser and James Trimble from the University of Glasgow, Scotland, received a surprising third place by solving one private instance correctly. They made the most of the situation that we had only three submission on this track. Their solver (heidi) implements an incomplete approach, where they heuristically compute a hypertree decomposition and used simple rules to check whether the width is equal to two.

POS	#	$\#^{all}$	Solver	t_{avg}	t_{sum}	Source (github)	Ref.
1	69	144	asc	69.38s	1.32h	`ASchidler/frasmt_pace`	[139]
2	31	48	TULongo	95.96s	0.83h	`TULongo/pace-2019-HD-exact`	[124]
3	1	6	heidi	0.14s	0.00h	`jamestrimble/heidi`	[146]

(a) Track 2a (Hypertree Width/Exact).

POS	Score	Solver	#	PAR1	Δ_w	Source (github)	Ref.
–	na	htdecomp	100	na	na		
1	5.0	hypebeast	100	0.1h	501	`jamestrimble/hypebeast`	[147]
2	14.1	TULongo	98	2.3h	20	`TULongo/pace-2019-HD-Heuristic`	[125]
3	128.9	asc	30	27.5	11	`ASchidler/frasmt_pace`	[139]

(b) Track 2b (Hypertree Width/Heuristic).

Table 5 Detailed overview on the results of the Hypertree Width tracks. # refers to the number of solved private instances. $\#^{all}$ comprises the number of solved public and private instances. t_{avg} and t_{sum} refer to average and cumulated runtime of solved private instances, respectively. PAR1 refers to the runtime where all unsolved instances are accounted by 1,800 seonds. Δ_w refers to the sum of the width difference to the resulted output by the judge, i.e., the sum over $w_{\text{solver}} - w_{\text{judge}}$ for each instance I.

4.3 Track 2b (Hypertree Width/Heuristic)

Table 5b summarizes runtime results for all submitted solvers. As for Track 2a, we received only 3 submissions, while we were hoping to attract more researchers from the community to this topic. We allowed each solver 30 minutes wall clock time per instance, while ensuring that not more than one core was used. Our aim for the track was to have more decomposers available to foster algorithms that employ decompositions for constraint programming and database applications in the near future. In the past, we observed at multiple occasions that heuristics are often only well applicable if there is a fairly good balance between running time of the computation of the decomposer and width of the decomposition [32, 33, 59, 61, 66]. Improving the width by one pays off if the improvement runs fast and the width is fairly large, however, on instances with small width there is no practical point in spending additional runtime to improve while it might even exceed the running time of the later algorithm that exploits the decomposition. In consequence, we decided to favor submissions that produce a result fairly quickly while still penalizing decompositions that are far from the virtual best results. Since the widths are fairly small, we decided for a very simple score (avoiding exponentially increasing penalties) and compute it per instance I by $(50,000 + t + 50 \cdot (w_{\text{solver}} - w_{\text{judge}}))/1,000,000$ where t refers to the wall clock and w_{solver} refers to the resulting width of the considered solver for I and w_{judge} consists of the width htdecomp [44, 84] produced for I, which we used as judge. The way we selected the score also depended on the situation that the runtime environment optil.io has certain technical restrictions in case of unknown optimal results.

Winning Team. Patrick Prosser and James Trimble from the University of Glasgow, Scotland, obtained the first place by obtaining a score of 5.0 by solving 100 instances in 0.1 hours with a cumulated width of 1104 and an overall difference to the judge by 501. Their solver (hypebeast) implements a very simple bucket elimination strategy by starting from a single tree node containing all hyperedges and then trying to move edges to deeper tree nodes.

Runner-up. Davide Mario Longo from TU Wien, Austria, scored second with 14.1 points by solving 98 private instances in 2.3 hours with a total width difference to the judge of 20. His solver (TULongo/HdSolveH) used a variant of det-k-decomp that prunes the search tree heuristically [85]. While the det-k-decomp algorithm performs well on yes instances, it is slow on no instances. So, Longo decided not to perform a complete search, but to prune the search space by looking only at certain separators. The results were very close to the winning team and only the running time cost him the first position. It is notable that, however, the width is much closer to the result by the judge. More precisely, TULongo outputted better results on 86 instances.

Third Place. André Schidler and Stefan Szeider from TU Wien, Austria, obtained the third position at a score of 128.9 by solving 30 instances using the same technique as above.

5 PACE organization

The composition of the steering committee and program committee during PACE 2019 was as follows.

	Édouard Bonnet	LIP, ENS de Lyon
	Holger Dell	IT University of Copenhagen
	Bart M. P. Jansen (chair)	Eindhoven University of Technology
Steering committee:	Thore Husfeldt	IT Univ. of Copenhagen & Lund Univ.
	Petteri Kaski	Aalto University
	Christian Komusiewicz	Philipps-Universität Marburg
	Frances A. Rosamond	University of Bergen
	Florian Sikora	LAMSADE, Université Paris Dauphine
Program Committee	Johannes Fichte	TU Dresden
(Tracks 1a, 2a, 2b):	Markus Hecher	TU Vienna & University of Potsdam

The Program Committee of **PACE 2020** be chaired by Łukasz Kowalik (Univ. of Warsaw).

6 Conclusion and Future Editions of PACE

We thank all the participants for their enthusiasm, strong and interesting contributions. Special thanks go to the participants who also presented at IPEC 2019. We are very happy that this edition attracted many people and that also people from the SAT community showed interest in the latest standings on the vertex cover solvers. While we were hoping to attract more people to the hypertree width measure, which is related to constraint programming and to the database community. We are still happy about strong contributions and hope that this will continue for future editions by considering popular problems to the community or even by repeating previously posted problems.

This year we changed the requirements for submissions by allowing external libraries, but enforcing that these are open source. Further, we asked participants to provide a solver description and to place the source code on a public data library, which is hosted long-term by a public body, e.g., Zenodo. In line with this, we provided the complete pool of instances from which we selected instances, the instance selection process including references to the original source, as well as the evaluation in a public data library.

We welcome anyone who is interested to add their name to the mailing list on the PACE website to receive updates and join the discussion. We look forward to the next edition. Detailed information will be posted on the website at `pacechallenge.org`.

References

1 data experts gmbh. https://www.data-experts.de/.

2 Intel® Xeon® processor E5-2695 v3. https://ark.intel.com/content/www/us/en/ark/products/81057/intel-xeon-processor-e5-2695-v3-35m-cache-2-30-ghz.html, 2014.

3 Centre for information services and high performance computing. https://tu-dresden.de/zih/hochleistungsrechnen/, 2019. Project: pacechallage2019.

4 Dimacs challenge. http://dimacs.rutgers.edu/programs/challenge/, 2019.

5 Networks project. https://www.thenetworkcenter.nl, 2019.

6 Mahmoud Abo Khamis, Hung Q. Ngo, Christopher Ré, and Atri Rudra. Joins via geometric resolutions: Worst-case and beyond. In *Proceedings of the 34th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS'15)*, pages 213–228, Melbourne, Victoria, Australia, 2015. Assoc. Comput. Mach., New York. doi:10.1145/2745754.2745776.

7 Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. Faq: Questions asked frequently. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS '16)*, pages 13–28, San Francisco, California, USA, 2016. Assoc. Comput. Mach., New York. doi:10.1145/2902251.2902280.

8 Michael Abseher, Nysret Musliu, and Stefan Woltran. htd – a free, open-source framework for (customized) tree decompositions and beyond. In Domenico Salvagnin and Michele Lombardi, editors, *Proceedings of the 14th International Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming (CPAIOR'17)*, volume 10335 of *Lecture Notes in Computer Science*, pages 376–386, Padova, Italy, June 2017. Springer Verlag. doi:10.1007/978-3-319-59776-8_30.

9 Vasily Alferov. Cheburashka vertex cover solver. Zenodo, June 2019. doi:10.5281/zenodo.3236897.

10 Mario Alviano, Francesco Calimeri, Günther Charwat, Minh Dao-Tran, Carmine Dodaro, Giovambattista Ianni, Thomas Krennwallner, Martin Kronegger, Johannes Oetsch, Andreas Pfandler, Jörg Pührer, Christoph Redl, Francesco Ricca, Patrik Schneider, Martin Schwengerer, LaraKatharina Spendier, Johannes Peter Wallner, and Guohui Xiao. The fourth answer set programming competition: Preliminary report. In Pedro Cabalar and TranCao Son, editors, *Proceedings of the 12th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'13)*, volume 8148 of *Lecture Notes in Computer Science*, pages 42–53. Springer Verlag, Corunna, Spain, September 2013. doi:10.1007/978-3-642-40564-8_5.

11 Krzysztof Apt. *Principles of Constraint Programming.* Cambridge University Press, Cambridge, New York, NY, USA, 1st edition, 2009.

12 Molham Aref, Balder ten Cate, Todd J. Green, Benny Kimelfeld, Dan Olteanu, Emir Pasalic, Todd L. Veldhuizen, and Geoffrey Washburn. Design and implementation of the logicblox system. In Susan B. Davidson and Zack Ives, editors, *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD'15)*, pages 1371–1382, Melbourne, Victoria, Australia, 2015. Assoc. Comput. Mach., New York. doi:10.1145/2723372.2742796.

13 Patricia C. Arocena, Boris Glavic, Radu Ciucanu, and Renée J. Miller. The ibench integration metadata generator. In Chen Li and Volker Markl, editors, *Proceedings of Very Large Data Bases (VLDB) Endowment*, volume 9:3, pages 108–119. Very Large Data Base Endowment, November 2015. URL: https://github.com/RJMillerLab/ibench.

14 G. Audemard, F. Boussemart, C. Lecoutre, and C. Piette. XCSP3: an XML-based format designed to represent combinatorial constrained problems. http://xcsp.org, 2016.

15 Nurzhan Bakibayev, Tomáš Kočiský, Dan Olteanu, and Jakub Závodný. Aggregation and ordering in factorised databases. *Proceedings of Very Large Data Bases Endowment (VLDB'13)*, 6(14):1990–2001, September 2013. doi:10.14778/2556549.2556579.

16 Max Bannach and Sebastian Berndt. Practical Access to Dynamic Programming on Tree Decompositions. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *Proceedings of the 26th Annual European Symposium on Algorithms (ESA 2018)*, volume 112 of *Leibniz*

International Proceedings in Informatics (LIPIcs), pages 6:1–6:13, Helsinki, Finland, 2018. Dagstuhl Publishing. doi:10.4230/LIPIcs.ESA.2018.6.

17 Max Bannach, Sebastian Berndt, and Thorsten Ehlers. Jdrasil: A Modular Library for Computing Tree Decompositions. In Costas S. Iliopoulos, Solon P. Pissis, Simon J. Puglisi, and Rajeev Raman, editors, *Proceedings of the 16th International Symposium on Experimental Algorithms (SEA 2017)*, volume 75 of *Dagstuhl Publishing*, pages 28:1–28:21, London, UK, 2017. Leibniz International Proceedings in Informatics (LIPIcs). doi:10.4230/LIPIcs.SEA.2017.28.

18 Michael Benedikt, George Konstantinidis, Giansalvatore Mecca, Boris Motik, Paolo Papotti, Donatello Santoro, and Efthymia Tsamoura. Benchmarking the chase. In Floris Geerts, editor, *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS'17)*, pages 37–52, Chicago, Illinois, USA, 2017. Assoc. Comput. Mach., New York. URL: https://github.com/dbunibas/chasebench.

19 J. Berg, N. Lodha, M. Järvisalo, and S. Szeider. MaxSAT benchmarks based on determining generalized hypertree-width. Technical report, MaxSAT Evaluation 2017, 2017.

20 Sebastian Berndt. Computing tree width: From theory to practice and back. In Florin Manea, Russell G. Miller, and Dirk Nowotka, editors, *Sailing Routes in the World of Computation: Proceedings of the 14th Conference on Computability in Europe (CiE 2018)*, volume 10936 of *Lecture Notes in Computer Science*, pages 81–88, Kiel, Germany, 2018. Springer Verlag. doi:10.1007/978-3-319-94418-0_8.

21 Manuel Bichler, Michael Morak, and Stefan Woltran. Single-shot epistemic logic program solving. In Jeffrey S. Rosenschein and Jérôme Lang, editors, *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI 2018)*, pages 1714–1720. The AAAI Press, 2018.

22 Johannes Blum and Sabine Storandt. Computation and growth of road network dimensions. In Lusheng Wang and Daming Zhu, editors, *Proceedings of the 24th International Computing and Combinatorics Conference (COCOON 2018)*, volume 10976 of *Lecture Notes in Computer Science*, pages 230–241, Qing Dao, China, July 2018. Springer Verlag. doi:10.1007/978-3-319-94776-1_20.

23 Jori Bomanson, Martin Gebser, and Tomi Janhunen. Improving the normalization of weight rules in answer set programs. In Eduardo Fermé and João Leite, editors, *Proceedings of the 14th European Conference on Logics in Artificial Intelligence (JELIA'14)*, pages 166–180, Funchal, Madeira, Portugal, September 2014. Springer Verlag.

24 Jori Bomanson, Martin Gebser, and Tomi Janhunen. Lp2normal and lp2normal2. https://research.ics.aalto.fi/software/asp/lp2normal/, 2016.

25 John A. Bondy and U. Murty. *Graph theory*, volume 244 of *Graduate Texts in Mathematics*. Springer Verlag, 2008.

26 Édouard Bonnet and Florian Sikora. The PACE 2018 Parameterized Algorithms and Computational Experiments Challenge: The Third Iteration. In Christophe Paul and Michal Pilipczuk, editors, *Proceedings of the 13th International Symposium on Parameterized and Exact Computation (IPEC 2018)*, volume 115 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:15, Helsinki, Finland, 2019. Dagstuhl Publishing. doi:10.4230/LIPIcs.IPEC.2018.26.

27 G. Brewka, T. Eiter, and M. Truszczyński. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011. doi:10.1145/2043174.2043195.

28 Shaowei Cai, Wenying Hou, Jinkun Lin, and Yuanjie Li. Improving local search for minimum weight vertex cover by dynamic strategies. In *IJCAI*, pages 1412–1418. ijcai.org, 2018.

29 Francesco Calimeri, Giovambattista Ianni, and Francesco Ricca. The third open answer set programming competition. *Theory Pract. Log. Program.*, 14:117–135, 1 2014. doi:10.1017/S1471068412000105.

30 Francesco Calimeri, Simona Perri, and Jessica Zangari. Optimizing answer set computation via heuristic-based decomposition. *Theory Pract. Log. Program.*, 19(4):603—628, 2019. doi:10.1017/S1471068419000036.

31 Mathieu Chapelle, Mathieu Liedloff, Ioan Todinca, and Yngve Villanger. Treewidth and pathwidth parameterized by the vertex cover number. *Discrete Applied Mathematics*, 216:114–129, 2017.

32 Günther Charwat and Stefan Woltran. Dynamic programming-based QBF solving. In Florian Lonsing and Martina Seidl, editors, *Proceedings of the 4th International Workshop on Quantified Boolean Formulas (QBF'16)*, volume 1719, pages 27–40. CEUR Workshop Proceedings (CEUR-WS.org), 2016. co-located with 19th International Conference on Theory and Applications of Satisfiability Testing (SAT'16).

33 Günther Charwat and Stefan Woltran. Expansion-based QBF solving on tree decompositions. *Fundam. Inform.*, 167(1-2):59–92, 2019.

34 Jiehua Chen and Sven Grottke. A fast solver for minimum vertex cover. Zenodo, June 2019. doi:10.5281/zenodo.3236992.

35 David Cohen, Peter Jeavons, and Marc Gyssens. A unified theory of structural tractability for constraint satisfaction problems. *J. of Computer and System Sciences*, 74(5):721–743, 2008.

36 Christophe Crespelle, Eduard Eiben, and Kirill Simonov. Vertex cover solver for pace 2019. Zenodo, May 2019. doi:10.5281/zenodo.3235533.

37 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

38 Marek Cygan, Łukasz Kowalik, Arkadiusz Socała, and Krzysztof Sornat. Approximation and parameterized complexity of minimax approval voting. *J. Artif. Intell. Res.*, 63:495–513, 2018. doi:10.1613/jair.1.11253.

39 Rina Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

40 Holger Dell, Thore Husfeldt, Bart M. P. Jansen, Petteri Kaski, Christian Komusiewicz, and Frances A. Rosamond. The First Parameterized Algorithms and Computational Experiments Challenge. In Jiong Guo and Danny Hermelin, editors, *11th International Symposium on Parameterized and Exact Computation (IPEC 2016)*, volume 63 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:9, Aarhus, Denmark, 2017. Dagstuhl Publishing. doi:10.4230/LIPIcs.IPEC.2016.30.

41 Holger Dell, Christian Komusiewicz, Nimrod Talmon, and Mathias Weller. The PACE 2017 Parameterized Algorithms and Computational Experiments Challenge: The Second Iteration. In Daniel Lokshtanov and Naomi Nishimura, editors, *Proceedings of the 12th International Symposium on Parameterized and Exact Computation (IPEC 2017)*, volume 89 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:12, Vienna, Austria, 2018. Dagstuhl Publishing. doi:10.4230/LIPIcs.IPEC.2017.30.

42 Camil Demetrescu, Andrew V. Goldberg, and David S. Johnson. The shortest path problem: Ninth dimacs implementation challenge. http://users.diag.uniroma1.it/challenge9/download.shtml, 2009.

43 Marc Denecker, Joost Vennekens, Stephen Bond, Martin Gebser, and Mirosław Truszczyński. The second answer set programming competition. In Esra Erdem, Fangzhen Lin, and Torsten Schaub, editors, *Proceedings of the 10th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'09)*, volume 5753 of *Lecture Notes in Computer Science*, pages 637–654. Springer Verlag, Potsdam, Germany, September 2009. doi:10.1007/978-3-642-04238-6_75.

44 Artan Dermaku, Tobias Ganzow, Georg Gottlob, Ben McMahan, Nysret Musliu, and Marko Samer. Heuristic methods for hypertree decomposition. In Alexander Gelbukh and Eduardo F. Morales, editors, *Procedings of the 7th Mexican International Conference on Artificial Intelligence on Advances in Artificial Intelligence (MICAI 2008)*, volume 5317 of *Lecture Notes in Computer Science*, pages 1–11. Springer Verlag, 2008. doi:10.1007/978-3-540-88636-5_1.

45 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate Texts in Mathematics*. Springer Verlag, 2012.

46 Diego Delle Donne and Guido Tagliavini. Star routing: Between vehicle routing and vertex cover. In *COCOA*, volume 11346 of *Lecture Notes in Computer Science*, pages 522–536. Springer, 2018.

47 Eugene F. Dumitrescu, Allison L. Fisher, Timothy D. Goodrich, Travis S. Humble, Blair D. Sullivan, and Andrew L. Wright. Benchmarking treewidth as a practical component of tensor network simulations. *PLOS ONE*, 13(12):1–19, 12 2018. `doi:10.1371/journal.pone.0207827`.

48 Arnaud Durand and Stefan Mengel. Structural tractability of counting of solutions to conjunctive queries. *Theoretical Computer Science*, 57(4):1202–1249, 2015.

49 M. Ayaz Dzulfikar, Johannes K. Fichte, and Markus Hecher. daajoe/sat2vc – a CNF-SAT to VC converter. `https://github.com/daajoe/sat2vc`, 2019.

50 M. Ayaz Dzulfikar, Johannes K. Fichte, and Markus Hecher. PACE2019: Track 1 - vertex cover instances. Zenodo, July 2019. `doi:10.5281/zenodo.3354609`.

51 Leah Epstein, Asaf Levin, and Gerhard J. Woeginger. Vertex cover meets scheduling. *Algorithmica*, 74(3):1148–1173, 2016.

52 Michael R. Fellows, Lars Jaffke, Aliz Izabella Király, Frances A. Rosamond, and Mathias Weller. What is known about vertex cover kernelization? In *Adventures Between Lower Bounds and Higher Altitudes*, volume 11011 of *Lecture Notes in Computer Science*, pages 330–356. Springer, 2018.

53 J. K. Fichte. daajoe/gtfs2graphs – a GTFS transit feed to graph format converter. `https://github.com/daajoe/gtfs2graphs`, 2016.

54 Johannes K. Fichte. Asp horn backdoors. `https://github.com/daajoe/asp_horn_backdoors`, 2014.

55 Johannes K. Fichte. Sat horn backdoors. `https://github.com/daajoe/sat_horn_backdoors`, 2018.

56 Johannes K. Fichte and Markus Hecher. PACE2019: Track 2a+b - hypertree decomposition instances. Zenodo, July 2019. `doi:10.5281/zenodo.3354607`.

57 Johannes K. Fichte, Markus Hecher, Neha Lodha, and Stefan Szeider. An SMT approach to fractional hypertree width. In John Hooker, editor, *Proceedings of the 24th International Conference on Principles and Practice of Constraint Programming (CP 2018)*, volume 11008 of *Lecture Notes in Computer Science*, pages 109–127, Lille, France, 2018. Springer Verlag.

58 Johannes K. Fichte, Markus Hecher, Michael Morak, and Stefan Woltran. Answer set solving with bounded treewidth revisited. In Marcello Balduccini and Tomi Janhunen, editors, *Proceedings of the 14th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'17)*, volume 10377 of *Lecture Notes in Computer Science*, pages 132–145, Espoo, Finland, July 2017. Springer Verlag. `doi:10.1007/978-3-319-61660-5_13`.

59 Johannes K. Fichte, Markus Hecher, Michael Morak, and Stefan Woltran. DynASP2.5: Dynamic programming on tree decompositions in action. In Daniel Lokshtanov and Naomi Nishimura, editors, *Proceedings of the 12th International Symposium on Parameterized and Exact Computation (IPEC'17)*. Dagstuhl Publishing, 2017. `doi:10.4230/LIPIcs.IPEC.2017.17`.

60 Johannes K. Fichte, Markus Hecher, Michael Morak, and Stefan Woltran. Exploiting treewidth for projected model counting and its limits. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *Proceedings on the 21th International Conference on Theory and Applications of Satisfiability Testing (SAT'18)*, volume 10929 of *Lecture Notes in Computer Science*, pages 165–184, Oxford, UK, July 2018. Springer Verlag.

61 Johannes K. Fichte, Markus Hecher, and Markus Zisser. gpusat2 – an improved gpu model counter. In Simon de Givry and Thomas Schiex, editors, *Proceedings of the 25th International Conference on Principles and Practice of Constraint Programming (CP 2018)*, 2019. To appear.

62 Johannes K. Fichte, Neha Lodha, and Stefan Szeider. Sat-based local improvement for finding tree decompositions of small width. In Serge Gaspers and Toby Walsh, editors, *Proceedings of the 20th International Conference on Theory and Applications of Satisfiability Testing (SAT*

2017), volume 10491 of *Lecture Notes in Computer Science*, pages 401–411, Melbourne, VIC, Australia, 2017. Springer Verlag. doi:10.1007/978-3-319-66263-3_25.

63 Johannes K. Fichte and Stefan Szeider. Backdoors to tractable answer-set programming. *Artificial Intelligence*, 220(0):64–103, 2015. doi:10.1016/j.artint.2014.12.001.

64 Johannes Klaus Fichte and Markus Hecher. Treewidth and counting projected answer sets. In Marcello Balduccini, Yuliya Lierler, and Stefan Woltran, editors, *Proceedings of the 15th International Conference on Logic Programming and Nonmonotonic Reasoning LPNMR 2019*, volume 11481 of *Lecture Notes in Computer Science*, pages 105–119, Philadelphia, PA, USA, 2019. Springer Verlag. doi:10.1007/978-3-030-20528-7_9.

65 Johannes Klaus Fichte, Markus Hecher, and Arne Meier. Counting complexity for reasoning in abstract argumentation. In Pascal Van Hentenryck and Zhi-Hua Zhou, editors, *Proceedings of the 33rd AAAI Conference on Artificial Intelligence, AAAI 2019*, pages 2827–2834, Honolulu, Hawaii, USA, 2019. The AAAI Press.

66 Johannes Klaus Fichte, Markus Hecher, Stefan Woltran, and Markus Zisser. Weighted model counting on the GPU by exploiting small treewidth. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *Proceedings of the 26th Annual European Symposium on Algorithms (ESA'18)*, volume 112 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 28:1–28:16. Dagstuhl Publishing, 2018. doi:10.4230/LIPIcs.ESA.2018.28.

67 Aleksander Figiel. An exact vertex cover solver using kernels. Zenodo, June 2019. doi:10.5281/zenodo.3236867.

68 Wolfgang Fischl, Georg Gottlob, Davide M. Longo, and Reinhard Pichler. HyperBench: a benchmark of hypergraphs. http://hyperbench.dbai.tuwien.ac.at, 2017.

69 Wolfgang Fischl, Georg Gottlob, Davide Mario Longo, and Reinhard Pichler. Hyperbench: A benchmark and tool for hypergraphs and empirical findings. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS'19)*, pages 464–480, Amsterdam, Netherlands, 2019. Assoc. Comput. Mach., New York. doi:10.1145/3294052.3319683.

70 Wolfgang Fischl, Georg Gottlob, and Reinhard Pichler. General and fractional hypertree decompositions: Hard and easy cases. In Jan Van den Bussche and Marcelo Arenas, editors, *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS'18)*, pages 17–32, Houston, TX, USA, June 2018. Assoc. Comput. Mach., New York.

71 Daniel Fremont. counting-benchmarks. http://tinyurl.com/countingbenchmarks, 2016.

72 Eugene C. Freuder. A sufficient condition for backtrack-bounded search. *J. of the ACM*, 29(1):24–32, 1982.

73 Marco Gario. Backdoors for sat. Master's thesis, TU Dresden, 2011. Program sources at https://marco.gario.org/work/master/.

74 Marco Gario. Hornvcbuilder. https://marco.gario.org/work/master/, 2011.

75 Marco Gario. Horn backdoor detection via vertex cover: Benchmark description. In Adrian Balint, Anton Belov, Daniel Diepold, Simon Gerber, Matti Järvisalo, and Carsten Sinz, editors, *Proceedings of SAT Challenge 2012; Solver and Benchmark Descriptions*, Helsinki, Finland, 2012. University of Helsinki.

76 Serge Gaspers, Joachim Gudmundsson, Mitchell Jones, Julián Mestre, and Stefan Rümmele. Turbocharging Treewidth Heuristics. In Jiong Guo and Danny Hermelin, editors, *Proceedings of the 11th International Symposium on Parameterized and Exact Computation (IPEC'16)*, volume 63 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:13. Dagstuhl Publishing, 2017. doi:10.4230/LIPIcs.IPEC.2016.13.

77 M. Gebser, T. Schaub, S. Thiele, and P. Veber. Detecting inconsistencies in large biological networks with answer set programming. *Theory Pract. Log. Program.*, 11(2-3):323–360, 2011.

78 Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Clingo = ASP + control: Preliminary report. *CoRR*, abs/1405.3694, 2014. arXiv:1405.3694.

79 Martin Gebser, Lengning Liu, Gayathri Namasivayam, André Neumann, Torsten Schaub, and Mirosław Truszczyński. The first answer set programming system competition. In Chitta Baral, Gerhard Brewka, and John Schlipf, editors, *Proceedings of the 9th Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07)*, volume 4483 of *Lecture Notes in Computer Science*, pages 3–17, Tempe, AZ, USA, May 2007. Springer Verlag. doi:10.1007/978-3-540-72200-7_3.

80 F. Geerts, G. Mecca, P. Papotti, and D. Santoro. Mapping and cleaning. In Isabel Cruz, Elena Ferrari, and Yufei Tao, editors, *Proceedings of the IEEE 30th International Conference on Data Engineering (ICDE'14)*, pages 232–243, March 2014.

81 Martin Josef Geiger. Implementation of a metaheuristic for the steiner tree problem in graphs. Medeley Data, 2018. doi:10.17632/yf9vpkgwdr.1.

82 G. Gottlob, N. Leone, and F. Scarcello. Hypertree decompositions and tractable queries. *J. of Computer and System Sciences*, 64(3):579–627, 2002.

83 Georg Gottlob, Gianluigi Greco, and Francesco Scarcello. *Treewidth and Hypertree Width*, pages 3—38. Cambridge University Press, Cambridge, 2014. doi:10.1017/CBO9781139177801.002.

84 Georg Gottlob, Zoltan Miklos, Nysret Musliu, and Marko Samer. Hypertree complementary approaches to constraint satisfaction. https://www.dbai.tuwien.ac.at/proj/hypertree/downloads.html, 2016.

85 Georg Gottlob and Marko Samer. A backtracking-based algorithm for hypertree decomposition. *J. of Experimental Algorithmics*, 13:1:1.1–1:1.19, February 2009.

86 Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. In *Proceedings of the of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2006)*, pages 289–298. ACM Press, 2006.

87 Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. *ACM Transactions on Algorithms*, 11(1):Art. 4, 20, 2014.

88 Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. LUBM: A benchmark for OWL knowledge base systems. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(2):158–182, 2005.

89 LLC Gurobi Optimization. Gurobi optimizer reference manual, 2019. URL: http://www.gurobi.com.

90 Michael Hamann and Ben Strasser. Graph bisection with pareto optimization. *J. of Experimental Algorithmics*, 23:1.2:1–1.2:34, February 2018. doi:10.1145/3173045.

91 Falko Hegerfeld and Florian Nelles. hubhegnel/pace-2019: Release for zenodo. Zenodo, May 2019. doi:10.5281/zenodo.3234674.

92 Demian Hespe, Sebastian Lamm, Christian Schulz, and Darren Strash. WeGotYouCovered, May 2019. doi:10.5281/zenodo.2816116.

93 Demian Hespe, Sebastian Lamm, Christian Schulz, and Darren Strash. WeGotYouCovered: The winning solver from the PACE 2019 implementation challenge, vertex cover track. *CoRR*, 2019. arXiv:1908.06795.

94 Marijn J. H. Heule, Matti Juhani Järvisalo, and Martin Suda, editors. *Proceedings of the SAT Competition 2018: Solver and Benchmark Descriptions*, volume B. Department of Computer Science, University of Helsinki, University of Helsinki, 2018.

95 Holger H. Hoos and Thomas Stützle. SATLIB: An online resource for research on SAT. In Ian P. Gent, Hans van Maaren, and Toby Walsh, editors, *Proceedings of the 3rd Workshop on Satisfiability (SAT'00)*, pages 283–292, Renesse, The Netherlands, 2000. IOS Press. SATLIB is available online at www.satlib.org.

96 Md. Rafiqul Islam, Imran Hossain Arif, and Rifat Hasan Shuvo. Generalized vertex cover using chemical reaction optimization. *Appl. Intell.*, 49(7):2546–2566, 2019.

97 Yoichi Iwata. Linear-Time Kernelization for Feedback Vertex Set. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *Proceedings of the 44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz*

International Proceedings in Informatics (LIPIcs), pages 68:1–68:14. Dagstuhl Publishing, 2017. doi:10.4230/LIPIcs.ICALP.2017.68.

98 Yoichi Iwata and Yusuke Kobayashi. Improved analysis of highest-degree branching for feedback vertex set. *CoRR*, abs/1905.12233, 2019. arXiv:1905.12233.

99 Yoichi Iwata and Takuto Shigemura. Separator-based pruned dynamic programming for steiner tree. In *AAAI*, pages 1520–1527. AAAI Press, 2019.

100 T. Janhunen and I. Niemelä. The answer set programming paradigm. *AI Magazine*, 37(3):13–24, 2016. doi:10.1609/aimag.v37i3.2671.

101 P. Jégou, H. Kanso, and C. Terrioux. On the relevance of optimal tree decompositions for constraint networks. In *Proceedings of the IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI 2018)*, pages 738–743. IEEE Computer Soc., Nov 2018. doi:10.1109/ICTAI.2018.00116.

102 Roland Kaminski. clingo - a grounder and solver for logic programs. https://github.com/potassco/clingo, 2019.

103 Kustaa Kangas, Mikko Koivisto, and Sami Salonen. A Faster Tree-Decomposition Based Algorithm for Counting Linear Extensions. In Christophe Paul and Michal Pilipczuk, editors, *Proceedings of the 13th International Symposium on Parameterized and Exact Computation (IPEC 2018)*, volume 115 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:13. Dagstuhl Publishing, 2019. doi:10.4230/LIPIcs.IPEC.2018.5.

104 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

105 Steven Kelk, Georgios Stamoulis, and Taoyang Wu. Treewidth distance on phylogenetic trees. *Theoretical Computer Science*, 731:99–117, 2018. doi:10.1016/j.tcs.2018.04.004.

106 Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. FAQ: questions asked frequently. *CoRR*, abs/1504.04044, 2017.

107 Krzysztof Kiljan and Marcin Pilipczuk. Experimental Evaluation of Parameterized Algorithms for Feedback Vertex Set. In Gianlorenzo D'Angelo, editor, *Proceedings of the 17th International Symposium on Experimental Algorithms (SEA 2018)*, volume 103 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:12. Dagstuhl Publishing, 2018. doi:10.4230/LIPIcs.SEA.2018.12.

108 Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. MIT Press, 2009.

109 Tuukka Korhonen, Jeremias Berg, and Matti Järvisalo. Enumerating potential maximal cliques via sat and asp. In Thomas Eiter and Sarit Kraus, editors, *Proceedings of the International Joint Conference on Artificial Intelligence IJCAI 2019*, Macao, China, 2019. The AAAI Press.

110 Tuukka Korhonen, Jeremias Berg, and Matti Järvisalo. Solving graph problems via potential maximal cliques: An experimental evaluation of the bouchitté–todinca algorithm. *J. of Experimental Algorithmics*, 24(1):1.9:1–1.9:19, February 2019. doi:10.1145/3301297.

111 Philipp Klaus Krause, Lukas Larisch, and Felix Salfelder. The tree-width of c. *Discr. Appl. Math.*, 2019. doi:10.1016/j.dam.2019.01.027.

112 R. Krithika, Diptapriyo Majumdar, and Venkatesh Raman. Revisiting connected vertex cover: FPT algorithms and lossy kernels. *Theory Comput. Syst.*, 62(8):1690–1714, 2018.

113 Srijan Kumar, Bryan Hooi, Disha Makhija, Mohit Kumar, Christos Faloutsos, and VS Subrahmanian. Rev2: Fraudulent user prediction in rating platforms. In Yi Chang and Yan Liu, editors, *Proceedings of the 11th ACM International Conference on Web Search and Data Mining (WSDM'18)*, pages 333–341, Marina Del Rey, CA, USA, 2018. Assoc. Comput. Mach., New York.

114 Srijan Kumar, Francesca Spezzano, VS Subrahmanian, and Christos Faloutsos. Edge weight prediction in weighted signed networks. In Francesco Bonchi and Josep Domingo-Ferrer, editors, *Proceedings of the IEEE 16th International Conference on Data Mining (ICDM'16)*, pages 221–230, Barcelona, Catalonia, Spain, 2016. IEEE Computer Soc.

115 Hung V. Le. *Structural Results and Approximation Algorithms in Minor-free Graphs*. PhD thesis, Oregon State University, 2018. URL: `https://ir.library.oregonstate.edu/concern/graduate_thesis_or_dissertations/d217qv924`.

116 Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. How good are query optimizers, really? *Proceedings of Very Large Data Bases (VLDB) Endowment*, 9(3):204–215, November 2015.

117 Jure Leskovec. Stanford network analysis project. `https://snap.stanford.edu`, 2009.

118 Jure Leskovec, Daniel Huttenlocher, and Jon Kleinberg. Predicting positive and negative links in online social networks. In *Proceedings of the 19th International Conference on World Wide Web (WWW '10)*, pages 641–650, New York, NY, USA, 2010. Assoc. Comput. Mach., New York. `doi:10.1145/1772690.1772756`.

119 Jure Leskovec, Daniel Huttenlocher, and Jon Kleinberg. Signed networks in social media. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10)*, pages 1361–1370, New York, NY, USA, 2010. Assoc. Comput. Mach., New York. `doi:10.1145/1753326.1753532`.

120 Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Trans. Knowl. Discov. Data*, 1(1), March 2007. `doi:10.1145/1217299.1217301`.

121 Zijie Li and Peter van Beek. Finding small backdoors in SAT instances. In *Canadian Conference on AI*, volume 6657 of *Lecture Notes in Computer Science*, pages 269–280. Springer, 2011.

122 Cong Han Lim and Stephen Wright. k-support and ordered weighted sparsity for overlapping groups: Hardness and algorithms. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Proceedings of Advances in Neural Information Processing Systems 30 (NIPS 2017)*, pages 284–292. Curran Associates, Inc., 2017.

123 Neha Lodha, Sebastian Ordyniak, and Stefan Szeider. Sat-encodings for special treewidth and pathwidth. In Serge Gaspers and Toby Walsh, editors, *Proceedings of the 20th International Conference on Theory and Applications of Satisfiability Testing (SAT 2017)*, volume 10491 of *Lecture Notes in Computer Science*, pages 429–445, Melbourne, VIC, Australia, 2017. Springer Verlag. `doi:10.1007/978-3-319-66263-3\_27`.

124 Davide Mario Longo. Pace2019 hypertree width exact. Zenodo, May 2019. `doi:10.5281/zenodo.3236358`.

125 Davide Mario Longo. Pace2019 hypertree width heuristic. Zenodo, May 2019. `doi:10.5281/zenodo.3236369`.

126 Diptapriyo Majumdar, Venkatesh Raman, and Saket Saurabh. Polynomial kernels for vertex cover parameterized by small degree modulators. *Theory Comput. Syst.*, 62(8):1910–1951, 2018.

127 Silviu Maniu, Pierre Senellart, and Suraj Jog. An Experimental Study of the Treewidth of Real-World Graph Data. In Pablo Barcelo and Marco Calautti, editors, *Proceedings of the 22nd International Conference on Database Theory (ICDT 2019)*, volume 127 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:18. Dagstuhl Publishing, 2019. `doi:10.4230/LIPIcs.ICDT.2019.12`.

128 Marco Maratea, Francesco Ricca, Wolfgang Faber, and Nicola Leone. Look-back techniques and heuristics in DLV: Implementation, evaluation, and comparison to QBF solvers. *J. Algorithms*, 63(1–3):70–89, 2008. Benchmarks can be found at `http://asparagus.cs.uni-potsdam.de/contest/downloads/benchmarks-score-dlp.tgz`. `doi:10.1016/j.jalgor.2008.02.006`.

129 Thorsten Ehlers Max Bannach, Sebastian Berndt and Dirk Nowotka. Sat-encodings of tree decompositions. In Marijn Heule, Matti Juhani Järvisalo, and Martin Suda, editors, *Proceedings of SAT Competition 2018: Solver and Benchmark Descriptions*, number Report B-2018-1 in B, page 72. University of Helsinki, Department of Computer Science, 2018.

130 Julian McAuley and Jure Leskovec. Learning to discover social circles in ego networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems (NIPS'12)*, pages 539–547, USA, 2012. Curran Associates Inc.

131 Naomi Nishimura, Prabhakar Ragde, and Stefan Szeider. Solving #SAT using vertex covers. *Acta Informatica*, 44(7-8):509–523, 2007.

132 Dan Olteanu and Jakub Závodný. Size bounds for factorised representations of query results. *ACM Trans. Database Syst.*, 40(1):2:1–2:44, March 2015. doi:10.1145/2656335.

133 Steven Prestwich. Cnf encodings. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, chapter 2, pages 75–97. IOS Press, 2009.

134 Noam Ravid, Dori Medini, and Benny Kimelfeld. Ranked enumeration of minimal triangulations. In Dan Suciu and Christoph Koch, editors, *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS 2019)*, pages 74–88, Amsterdam, Netherlands, 2019. Assoc. Comput. Mach., New York. doi:10.1145/3294052.3319678.

135 Matei Ripeanu, Adriana Iamnitchi, and Ian Foster. Mapping the gnutella network. *IEEE Internet Computing*, 6(1):50–57, January 2002. doi:10.1109/4236.978369.

136 Neil Robertson and P.D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986.

137 Marko Samer and Stefan Szeider. Backdoor trees. In *AAAI*, pages 363–368. AAAI Press, 2008.

138 Marko Samer and Stefan Szeider. Fixed-parameter tractability. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, chapter 13, pages 425–454. IOS Press, 2009.

139 André Schidler and Stefan Szeider. HtdSMT - An SMT based solver for hypertree decompositions. Zenodo, May 2019. doi:10.5281/zenodo.3236333.

140 C. Sinz, A. Kaiser, and W. Küchlin. Formal methods for the validation of automotive product configuration data. *AI EDAM*, 17(01):75–97, 2003. URL: http://www-sr.informatik.uni-tuebingen.de/~sinz/DC.

141 Ben Strasser. Road graphs as tree-decomposition pace test instances. https://github.com/ben-strasser/road-graphs-pace16, 2016.

142 Ben Strasser. Computing tree decompositions with flowcutter: PACE 2017 submission. *CoRR*, abs/1709.08949, 2017. arXiv:1709.08949.

143 Hisao Tamaki. Positive-Instance Driven Dynamic Programming for Treewidth. In Kirk Pruhs and Christian Sohler, editors, *Proceedings of the 25th Annual European Symposium on Algorithms (ESA 2017)*, volume 87 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 68:1–68:13, Vienna, Austria, 2017. Dagstuhl Publishing. doi:10.4230/LIPIcs.ESA.2017.68.

144 Hisao Tamaki. Positive-instance driven dynamic programming for treewidth. *J. Comb. Optim.*, 37(4):1283–1311, May 2019. doi:10.1007/s10878-018-0353-z.

145 Transaction Processing Performance Council (TPC). TPC-H decision support benchmark. Technical report, TPC, 2014. URL: http://www.tpc.org/tpch/default.asp.

146 James Trimble. jamestrimble/heidi: v1.0.0: Pace Challenge 2019 version. Zenodo, June 2019. doi:10.5281/zenodo.3237427.

147 James Trimble. jamestrimble/hypebeast: v1.0.0: PACE Challenge 2019 version. Zenodo, May 2019. doi:10.5281/zenodo.3082314.

148 James Trimble. jamestrimble/peaty: v1.0.0: PACE Challenge 2019 entry. Zenodo, May 2019. doi:10.5281/zenodo.3082356.

149 Edward Tsang. *Foundations of Constraint Satisfaction.* Academic Press, 1993.

150 René van Bevern, Till Fluschnik, and Oxana Yu. Tsidulko. Parameterized algorithms and data reduction for safe convoy routing. *CoRR*, abs/1806.09540, 2018. arXiv:1806.09540.

151 Tom C. van der Zanden and Hans L. Bodlaender. Computing Treewidth on the GPU. In Daniel Lokshtanov and Naomi Nishimura, editors, *Proceedings of the 12th International Symposium on Parameterized and Exact Computation (IPEC 2017)*, volume 89 of *Leibniz*

International Proceedings in Informatics (LIPIcs), pages 29:1–29:13, Vienna, Austria, 2018. dp. doi:10.4230/LIPIcs.IPEC.2017.29.

152 Tom Cornelis van der Zanden. *Theory and Practical Applications of Treewidth.* PhD thesis, Utrecht University, 2019.

153 Rim van Wersch and Steven Kelk. Toto: An open database for computation, storage and retrieval of tree decompositions. *Discr. Appl. Math.*, 217:389–393, 2017. doi:10.1016/j.dam.2016.09.023.

154 Szymon Wasik, Maciej Antczak, Jan Badura, Artur Laskowski, and Tomasz Sternal. Optil.io: Cloud based platform for solving optimization problems using crowdsourcing approach. In Eric Gilbert and Karrie Karahalios, editors, *Proceedings of the 19th ACM Conference on Computer Supported Cooperative Work and Social Computing Companion*, CSCW '16 Companion, pages 433–436, New York, NY, USA, 2016. Assoc. Comput. Mach., New York. doi:10.1145/2818052.2869098.

155 Ke Xu. Bhoslib: Benchmarks with hidden optimum solutions for graph problems (maximum clique, maximum independent set, minimum vertex cover and vertex coloring). http://sites.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm, 2014.

156 Bogdan Zavalnij and Sandor Szabo. zbogdan/pace-2019 a. Zenodo, May 2019. doi:10.5281/zenodo.3228802.

157 Yuting Zhao and Fangzhen Lin. Answer set programming phase transition: A study on randomly generated programs. In Catuscia Palamidessi, editor, *Proceedings of the 19th International Conference on Logic Programming (ICLP'03)*, volume 2916 of *Lecture Notes in Computer Science*, pages 239–253, Mumbai, India, December 2003. Springer Verlag. doi:10.1007/978-3-540-24599-5_17.

158 Michal Ziobro and Marcin Pilipczuk. Finding hamiltonian cycle in graphs of bounded treewidth: Experimental evaluation. *CoRR*, abs/1803.00927, 2018. arXiv:1803.00927.