The First Parameterized Algorithms and Computational Experiments Challenge

Holger Dell[1], Thore Husfeldt[2], Bart M. P. Jansen[3], Petteri Kaski[4], Christian Komusiewicz[5], and Frances A. Rosamond[6]

2 **Saarland University, Saarbrücken, Germany; and Cluster of Excellence "Multimodal Computing and Interaction" (MMCI), Saarbrücken, Germany**
hdell@mmci.uni-saarland.de
2 **ITU Copenhagen, Denmark, and Lund University, Sweden**
thore@itu.dk
3 **Eindhoven University of Technology, The Netherlands**
b.m.p.jansen@tue.nl
4 **Aalto University, Finland**
petteri.kaski@aalto.fi
5 **Friedrich-Schiller-University Jena, Germany**
christian.komusiewicz@uni-jena.de
6 **University of Bergen, Norway**
frances.rosamond@uib.no

Abstract

In this article, the steering committee of the Parameterized Algorithms and Computational Experiments challenge (PACE) reports on the first iteration of the challenge. Where did PACE come from, how did it go, who won, and what's next?

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, K.7.2 Organizations

Keywords and phrases treewidth, feedback vertex set, contest, implementation challenge, FPT

Digital Object Identifier 10.4230/LIPIcs.IPEC.2016.30

1 Introduction

The Parameterized Algorithms and Computational Experiments Challenge (PACE) was conceived in Fall 2015 when many FPT researchers gathered at the Simons Institute for the Theory of Computing at UC Berkeley. A talk there [14] explored the practical implementability of theoretically tight FPT results, which seemed to offer an area for further investigation. PACE was born from a belief that a challenge could help deepen the relationship between parameterized algorithmics and practice. It was partially inspired by the success of SAT-solving competitions in neighboring communities. The goal of PACE is to investigate the applicability of algorithmic ideas studied and developed in the subfields of multivariate, fine-grained, parameterized, or fixed-parameter tractable algorithms. In particular, it aims to:

1. Bridge between algorithm design and analysis theory and algorithm engineering practice.
2. Inspire new theoretical developments.
3. Investigate the competitiveness of analytical and design frameworks developed in the communities.

4. Produce universally accessible libraries of implementations and repositories of benchmark instances.
5. Encourage dissemination of the findings in scientific papers.

Discussions throughout the community led to a website [18], a steering committee, and two challenge tracks for 2016 with program committees: Track A (Treewidth) and Track B (Feedback Vertex Set). The winners were announced at IPEC 2016 in Aarhus.

2 Competition track A: Treewidth

The treewidth of a graph is an important graph parameter, the theory and complexity of which has been intensely study in graph minor theory and fixed-parameter tractability (FPT). Given a graph G and an integer k, it is NP-complete to determine whether the treewidth of G is at most k, but there is an $\mathcal{O}(n^{k+2})$-time algorithm [1]. The problem can also be solved in FPT-time $2^{\mathcal{O}(k^3)}n$ [3], and a factor-5 approximation can be obtained in time $2^{\mathcal{O}(k)}n$ [4]. It is unknown whether the problem has a polynomial-time approximation scheme (PTAS).

Treewidth implementations are used in various contexts. For example, compilers allocate registers by computing proper colorings on control flow graphs, which turn out to have small treewidth in practice (e.g. [24]). Data structures for shortest path queries can use tree decompositions in a preprocessing phase (e.g. [5]). Graph theory can be guided by treewidth implementations when attempting to rigorously determine the treewidth of specific graph families (e.g. [15]). Finally, many problems in probabilistic inference use tree decompositions in a preprocessing phase (e.g. [13]).

While some treewidth implementations existed before PACE 2016, they were not easily accessible and sometimes buggy (as in the case of the Python SAGE implementation, which can produce non-optimal solutions), and their performances have never been compared in public. For PACE, we imposed a unified input/output format for the challenge and required all implementations to be made available on GitHub. Moreover, the details and raw data of all results mentioned in this document can be found in the GitHub repository [25] that we published. Using the tools and benchmark instances in the repository, it is a trivial matter to reproduce the results.

2.1 Submissions

The list of all implementation submissions is available online [16]. Two researchers submitted real-world instances for benchmarking:
1. Johannes Fichte (TU Wien) submitted transit networks.
2. Ben Strasser (Karlsruhe Institute of Technology) submitted road graphs.

Track A was subdivided into two tracks, based on the distinction between exact algorithms and heuristics. Each track was further subdivided into two subchallenges, based on the distinction between parallel and sequential algorithms. Since the best sequential exact algorithm outperformed the best parallel exact algorithm, the exact parallel subchallenge was discarded.

2.2 Sequential algorithms for computing treewidth exactly

The goal of this challenge was to compute a tree decomposition of minimum width. Three teams participated in this track, and two PACE co-organizers jointly contributed a further implementation. Figure 1 summarizes the results. In total, we used 200 instances in the

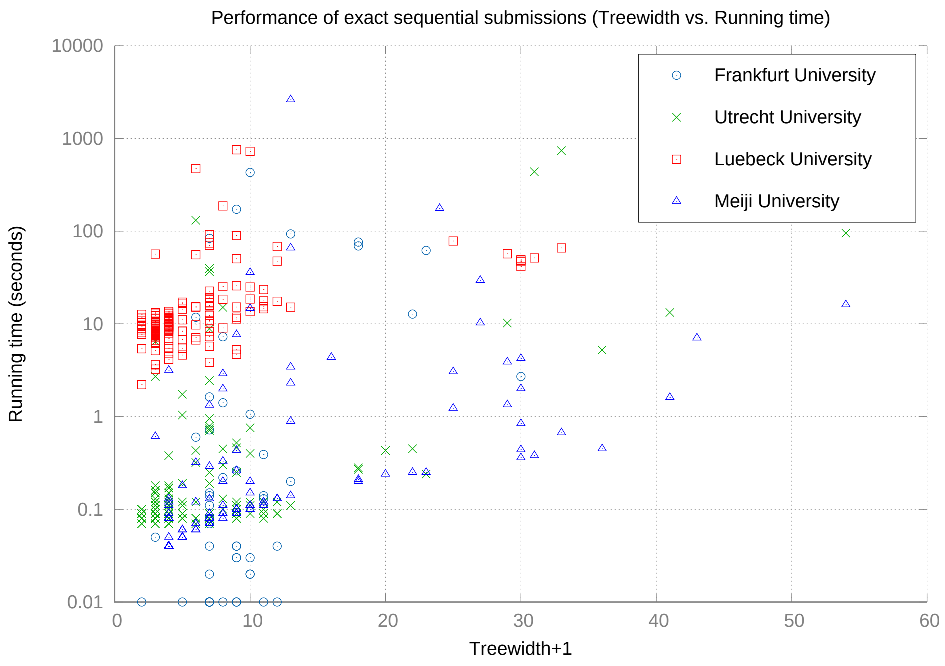

Figure 1 Results for exact sequential algorithms for computing treewidth. This plot shows one data point per solver-instance pair. The x-coordinate corresponds to the treewidth of the instance and the y-coordinate corresponds to the running time. We aborted the computation after a timeout of 100 seconds for most instances; some instances had a timeout of 1000 and 3600 seconds.

exact competition. The instances are samples of named graphs [21], control flow graphs [20], and DIMACS graph coloring instances [7]. The outcome of this challenge is:

1st prize, 350 €: Hisao Tamaki (Meiji University) solved 199 of 200 instances. The submission is written in C++ and is based on a modified version of the brute force approach of Arnborg et al. [1]. [`https://github.com/TCS-Meiji/treewidth-exact`]

2nd prize, 125 €: Hans Bodlaender and Tom Van der Zanden (Utrecht University) solved 173 of 200 instances. The submission is written in C# / Mono and relies on balanced separators as well as dynamic programming. [`https://github.com/TomvdZanden/BZTreewidth`]

3rd prize, 75 €: Max Bannach, Sebastian Berndt, and Thorsten Ehlers (Luebeck University) solved 166 of 200 instances. The submission is written in Java 8 and relies on a SAT-solver to find the optimal elimination order. [`https://github.com/maxbannach/Jdrasil`]

The implementation by Larisch and Salfelder from the track A program committee (Frankfurt University) solved 171 of 200 instances.

2.3 Heuristic algorithms for computing treewidth

The goal of this challenge was to compute a good tree decomposition in a given fixed timeout. We set the timeout to 100 seconds for every instance. We used the 200 instances from the exact competition and 81 additional, harder instances from the same sources; arguably, the instances were generally too easy for the heuristic challenge. Seven teams participated in this track. In total, 6 sequential programs and 3 parallel programs were submitted. Some teams submitted multiple programs, in which case we only kept the best-performing submission

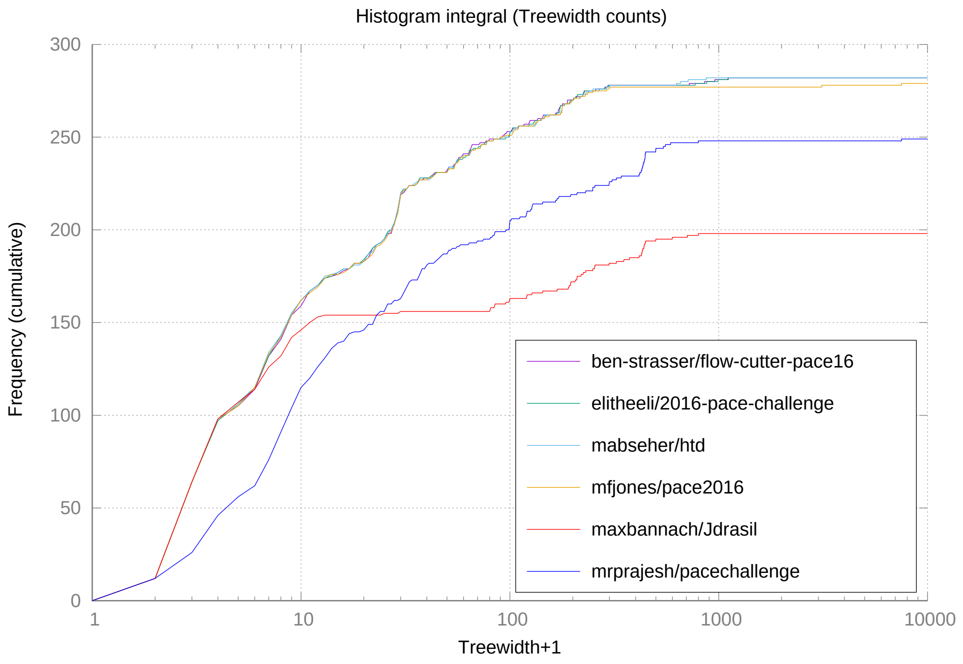

Figure 2 Results for heuristic sequential algorithms for computing treewidth. This plot shows, for every sequential submission and every x, the number y of instances for which the submission computed a tree decomposition of width at most x within the given time limit.

of each team in the ranking. For the evaluation, we viewed each instance as a voter and determined its ranking for the implementations from the width of the tree decomposition produced by the implementation after 100 seconds. We combined these votes using the Schulze method [23]. This process can be inspected in the testbed repository [22].

2.3.1 Sequential heuristic algorithms for computing treewidth

Figure 2 shows the results for sequential heuristic algorithms. As can be seen, the top three submissions nearly converge on the employed metric; this is perhaps explained by the fact that all three implement the basic minimum fill-in heuristic (tweaked in different ways). The other three submissions use more interesting techniques from FPT. The final ranking is:

1st prize, 350 €: Ben Strasser (Karlsruhe Institute of Technology) [`https://github.com/ben-strasser/flow-cutter-pace16`]
2nd prize, 125 €: Eli Fox-Epstein (Brown University) [`https://github.com/elitheeli/2016-pace-challenge`]
3rd prize, 75 €: Michael Abseher, Nysret Musliu, and Stefan Woltran (TU Wien) [`https://github.com/mabseher/htd`]
4th place: Serge Gaspers, Joachim Gudmundsson, Mitchell Jones, Julian Mestre, and Stefan Rümmele (UNSW and University of Sidney) [`https://github.com/mfjones/pace2016`]
5th place: Max Bannach, Sebastian Berndt, and Thorsten Ehlers (Luebeck University) [`https://github.com/maxbannach/Jdrasil`]
6th place: Kaustubh Joglekar, Akshay Kamble, and Rajesh Pandian (IIT Madras) [`https://github.com/mrprajesh/pacechallenge`]

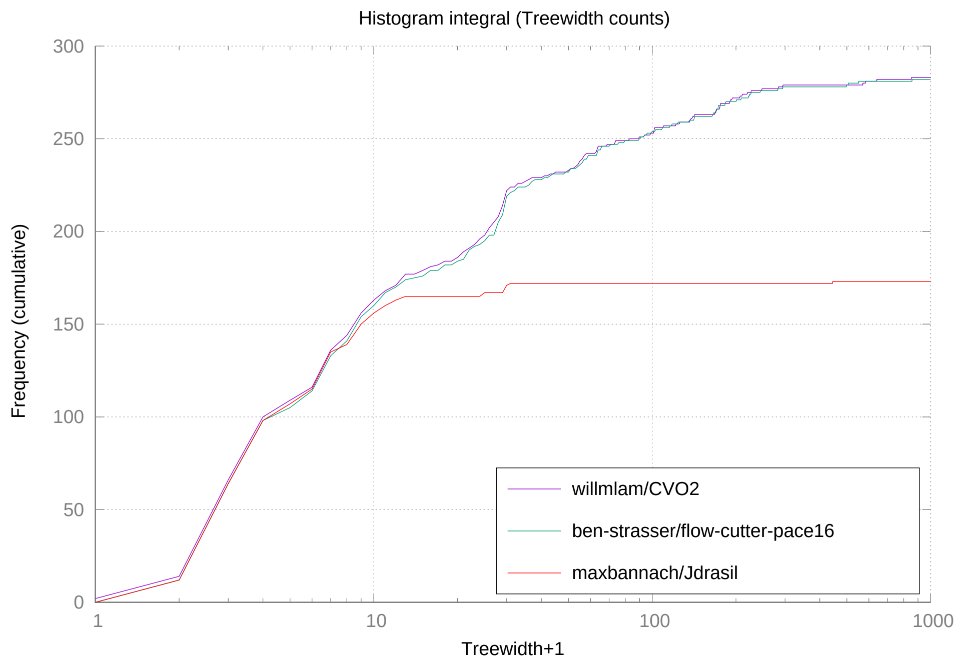

Figure 3 Results for heuristic parallel algorithms for computing treewidth. This plot shows, for every sequential submission and every x, the number y of instances for which the submission computed a tree decomposition of width at most x within the given time limit.

2.3.2 Parallel heuristic algorithms for computing treewidth

The results for parallel heuristic algorithms can be found in Figure 3. It leads to the following ranking:

1st prize, 350 €: Kalev Kask and William Lam (University of California at Irvine) [`https://github.com/willmlam/CVO2`]

2nd prize, 125 €: Ben Strasser (Karlsruhe Institute of Technology) [`https://github.com/ben-strasser/flow-cutter-pace16`]

3rd prize, 75 €: Max Bannach, Sebastian Berndt, and Thorsten Ehlers (Luebeck University) [`https://github.com/maxbannach/Jdrasil`]

3 Competition track B: Feedback Vertex Set

In the (undirected) Feedback Vertex Set problem we are given an undirected graph G and want to compute a smallest vertex set S such that removing S from G results in a forest, that is, a graph without cycles. Feedback Vertex Set is NP-complete [12] and one of the most prominent problems in parameterized algorithmics. Most fixed-parameter algorithms use the parameter solution size $k = |S|$.

Virtually all fixed-parameter algorithms make use of the fact that vertices of degree at most two can be easily removed from the graph. After this initial removal, a range of different techniques were used in the fixed-parameter algorithms. The first constructive fixed-parameter algorithm branches on a shortest cycle in the resulting graph. This cycle has length at most $2k$ in a yes-instance, which results in an overall running time of $(2k)^k n^{\mathcal{O}(1)}$ [8]. By using a randomized approach on the resulting graph, a running time of $4^k n^{\mathcal{O}(1)}$ can be obtained [2]. The first deterministic approaches to achieve running times of the form $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ use the iterative compression technique. It iteratively builds up the graph by adding one vertex at a time, and makes use of the fact that a size-k solution can be stored during this computation [6, 9]. Other fixed-parameter algorithms for this problem can be obtained by branching on a vertex of maximum degree or by LP-based techniques [11].

3.1 Challenge setup and participation

We collected 230 graphs, which were mostly from various application fields such as social networks, biological networks, road networks, or incidence graphs of CNF-SAT formulas. The graphs were selected so that there was a steady progression from easy to hard instances. The set of public training and hidden test instances is available in a GitHub repository [19].

To determine the winners, we counted the number of instances that could be solved within the given time limit. To avoid overemphasizing low-level improvements of the algorithms, we set the time limit to 30 minutes per instance. To identify programs that report non-optimal solutions we precomputed the optimal solutions for some instances using an ILP that was given at least 30 minutes on each instance. This ILP is based on cycle constraints. More precisely, we add constraints enforcing that for each cycle at least one vertex must be deleted by any solution. Since the number of constraints is usually exponential, they are added in a lazy fashion, that is, we compute a solution with only some initial constraints and check whether the solution is a feedback vertex set. If this is the case, then we have found an optimal solution, otherwise we add constraints for some of the remaining cycles and compute a new solution until a feedback vertex set is found.

Overall, 14 teams registered out of which seven eventually submitted a program. From those teams that submitted a program, three were from Germany, one from India, one from Japan, one from Poland, and one from Russia.

3.2 Results

In the following, we give for each team further details such as the number of solved instances, a brief algorithm description, the names of the participants, and a link to the code repository. All submissions apply a reduction rule that removes all vertices of degree at most two. The final ranking for track B is:

1st prize, 500 €: Yoichi Iwata (NII) and Kensuke Imanishi (University of Tokyo). This submission solved 84 out of 130 instances. The algorithm utilizes an LP-based branching [11] and an LP-based kernelization [10]. The program is written in Java. [`https://github.com/wata-orz/fvs`]

2nd prize, 300 €: Marcin Pilipczuk (University of Warsaw). This submission solved 66 out of 130 instances. The algorithm branches on a vertex of maximum degree. In addition, instances with small treewidth are solved by dynamic programming on tree decompositions and subcubic instances are solved by a polynomial-time algorithm that is based on a reduction to the graphic matroid parity problem. The program is written in C++. [`https://bitbucket.org/marcin_pilipczuk/fvs-pace-challenge`]

3rd prize, 200 €: Ruben Becker, Karl Bringmann, Dennis Gross, Erik Jan van Leeuwen, and Natalie Wirth (MPI Saarbrücken). This submission solved 50 out of 130 instances. The algorithm also branches on vertices of the highest degree. The search tree is pruned by computing upper and lower bounds. The program is written in C++. [`https://github.com/erikjanvl/FVS_MPI`]

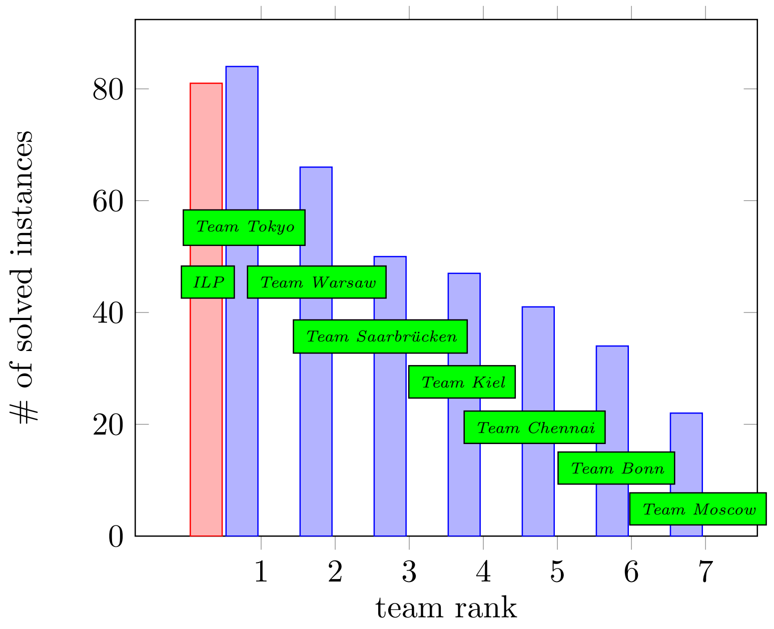

Figure 4 Results for track B. The figure shows, for each participating team, the number of instances for which an optimal solution was found within 30 minutes. The leftmost column corresponds to the Gurobi-based ILP used by the program committee.

4th place: Niklas Paulsen, Kevin Prohn, Malin Rau, and Lars Rohwedder (Kiel University). This submission solved 47 out of 130 instances. The algorithm is based on the combination of iterative compression with an improved branching strategy. Subcubic graphs are solved again by reduction graphic matroid parity. The program is written in C#. [`https://git.informatik.uni-kiel.de/npau/FFF`]

5th place: Shivam Garg (IIT Bombay), G. Philip, and Apoorva Tamaskar (Chennai Mathematical Institute). This submission solved 41 out of 130 instances. The algorithm branches on a shortest cycle. This program is written in Python. [`https://bitbucket.org/gphilip_bitbucket/pace-code`]

6th place: Fabian Brand, Simon Gehring, Florian Nelles, Kevin Wilkinghoff, and Xianghui Zhong (University of Bonn). This submission solved 34 out of 130 instances. The algorithm is based on iterative compression and also solves subcubic instances in polynomial time. The program is written in C++. Xianghui Zhong received a travel award of 780 € to be able to attend the award ceremony. [`https://github.com/s-gehring/feedback-vertex-set`]

7th place: Svyatoslav Feldsherov (Moscow State University). This submission solved 22 out of 130 instances. The algorithm uses the randomized approach with running time $4^k n^{\mathcal{O}(1)}$. An improvement is gained for the case where two vertices are connected by a multi-edge. In this case, the algorithm branches directly on these two vertices. The program is written in C++. [`https://github.com/feldsherov/pace2016`]

As a final remark, the ILP used by the program committee solved 81 out of 130 instances within the time limit. Thus, the best FPT approaches were competitive with this particular ILP formulation. Since alternative ILP formulations are possible, a more thorough comparison with further ILP-based approaches would be necessary to gain insight into the relative performance of FPT-based and ILP-based approaches for Feedback Vertex Set.

4 PACE organization

The PACE steering committee consists of the present authors, with Frances Rosamond as chair. The program committees for the two tracks in 2016 consisted of:

Track A:	Isolde Adler	University of Leeds
	Holger Dell (Chair)	Saarland University & Cluster of Excellence
	Thore Husfeldt	ITU Copenhagen & Lund University
	Lukas Larisch	University of Leeds
	Felix Salfelder	Goethe University Frankfurt
Track B:	Falk Hüffner	Industry
	Christian Komusiewicz	Friedrich-Schiller-University Jena

5 The future of PACE

As organizers, we consider the first iteration of PACE to be a huge success: we had great submissions building on existing and new theoretical ideas, which led to fast programs that performed well on the real-word inputs to which they were applied. The award ceremony at IPEC was very well attended, and many of the ALGO 2016 participants showed an interest in the competition.

To continue driving the transfer of algorithmic ideas from theory to practice, we intend PACE to become an annual event. The target problem(s) will change whenever relevant, but the same problem may be used for several consecutive years when there are indications that further developments are possible. Plans for the next iteration of PACE can be found on the challenge website [18].

We thank all the participants for their enthusiastic participation and look forward to many interesting iterations of the challenge in the future. We also thank all members of the community for their input in formulating the goals and setup of the challenge. We welcome anyone who is interested to add their name to the mailing list on the website [18] to receive PACE updates and join the discussion.

Acknowledgments. Prize money and travel grants were given through the generosity of Networks [17], an NWO Gravitation project of the University of Amsterdam, Eindhoven University of Technology, Leiden University, and the Center for Mathematics and Computer Science (CWI). The steering committee thanks the two program committees for their hard work in making PACE a success. In particular, we are grateful to Felix Salfelder for his help in preparing plots of the results.

References

1 Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Algebra. Discr.*, 8:277–284, 1987. `doi:10.1137/0608024`.

2 Ann Becker, Reuven Bar-Yehuda, and Dan Geiger. Randomized algorithms for the loop cutset problem. *J. Artif. Intell. Res. (JAIR)*, 12:219–234, 2000. `doi:10.1613/jair.638`.

3 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996. `doi:10.1137/S0097539793251219`.

4 Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michal Pilipczuk. A $c^k n$ 5-approximation algorithm for treewidth. *SIAM J. Comput.*, 45(2):317–378, 2016. `doi:10.1137/130947374`.

5 Krishnendu Chatterjee, Rasmus Ibsen-Jensen, and Andreas Pavlogiannis. Optimal reachability and a space-time tradeoff for distance queries in constant-treewidth graphs. In *Proc.*

24th ESA, volume 57 of *LIPIcs*, pages 28:1–28:17, 2016. doi:10.4230/LIPIcs.ESA.2016.28.

6 Frank K. H. A. Dehne, Michael R. Fellows, Michael A. Langston, Frances A. Rosamond, and Kim Stevens. An $O(2^{O(k)}n^3)$ FPT algorithm for the undirected feedback vertex set problem. *Theory Comput. Syst.*, 41(3):479–492, 2007. doi:10.1007/s00224-007-1345-z.

7 DIMACS graph coloring instances. URL: http://mat.gsia.cmu.edu/COLOR/instances.html.

8 Rodney G. Downey and Michael R. Fellows. Parameterized computational feasibility. In *Feasible Mathematics II*, pages 219–244. Birkhauser, 1994.

9 Jiong Guo, Jens Gramm, Falk Hüffner, Rolf Niedermeier, and Sebastian Wernicke. Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization. *J. Comput. Syst. Sci.*, 72(8):1386–1396, 2006. doi:10.1016/j.jcss.2006.02.001.

10 Yoichi Iwata. Linear-time kernelization for feedback vertex set. *CoRR*, abs/1608.01463, 2016. URL: http://arxiv.org/abs/1608.01463.

11 Yoichi Iwata, Magnus Wahlström, and Yuichi Yoshida. Half-integrality, LP-branching, and FPT algorithms. *SIAM J. Comput.*, 45(4):1377–1411, 2016. doi:10.1137/140962838.

12 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proc. of a Symp. on the Complexity of Computer Computations*, IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

13 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*. AAAI Press, 2011. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI11/paper/view/3771.

14 Petteri Kaski. Engineering motif search for large graphs, 2015. URL: https://simons.berkeley.edu/talks/petteri-kaski-2015-11-05.

15 Masashi Kiyomi, Yoshio Okamoto, and Yota Otachi. On the treewidth of toroidal grids. *Discrete Applied Mathematics*, 198:303–306, 2016. doi:10.1016/j.dam.2015.06.027.

16 List of all submissions for track A, 2016. URL: https://github.com/holgerdell/PACE-treewidth-testbed/blob/github/pace2016-submissions.yaml.

17 Networks project, 2016. URL: http://www.thenetworkcenter.nl.

18 Parameterized Algorithms and Computational Experiments website, 2015. URL: http://pacechallenge.wordpress.com.

19 Repository of all hidden and public inputs for track B on GitHub, 2016. URL: https://github.com/ckomus/PACE-fvs.

20 Repository of control flow graphs on GitHub, 2016. URL: https://github.com/freetdi/CFGs.git.

21 Repository of named graphs on GitHub, 2016. URL: https://github.com/freetdi/named-graphs.

22 Schulze rankings of heuristic treewidth implementations, 2016. URL: https://github.com/holgerdell/PACE-treewidth-testbed/blob/github/logs/2016-08-13.02-08-25/ranks-he-se.txt.

23 Markus Schulze. A new monotonic, clone-independent, reversal symmetric, and Condorcet-consistent single-winner election method. *Social Choice and Welfare*, 36(2):267–303, 2011. doi:10.1007/s00355-010-0475-4.

24 Mikkel Thorup. All structured programs have small tree-width and good register allocation. *Inf. Comput.*, 142(2):159–181, 1998. doi:10.1006/inco.1997.2697.

25 Treewidth testbed on GitHub, 2016. URL: https://github.com/holgerdell/PACE-treewidth-testbed.