The PACE 2020 Parameterized Algorithms and Computational Experiments Challenge: Treedepth

Łukasz Kowalik
Institute of Informatics, University of Warsaw, Poland
kowalik@mimuw.edu.pl

Marcin Mucha
Institute of Informatics, University of Warsaw, Poland
mucha@mimuw.edu.pl

Wojciech Nadara
Institute of Informatics, University of Warsaw, Poland
w.nadara@mimuw.edu.pl

Marcin Pilipczuk
Institute of Informatics, University of Warsaw, Poland
malcin@mimuw.edu.pl

Manuel Sorge
Institute of Informatics, University of Warsaw, Poland
manuel.sorge@mimuw.edu.pl

Piotr Wygocki
Institute of Informatics, University of Warsaw, Poland
p.wygocki@mimuw.edu.pl

Abstract

This year's Parameterized Algorithms and Computational Experiments challenge (PACE 2020) was devoted to the problem of computing the treedepth of a given graph. Altogether 51 participants from 20 teams, 12 countries and 3 continents submitted their implementations to the competition.

In this report, we describe the setup of the challenge, the selection of benchmark instances and the ranking of the participating teams. We also briefly discuss the approaches used in the submitted solvers and the differences in their performance on our benchmark dataset.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases computing treedepth, contest, implementation challenge, FPT

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.37

Funding This work has been supported by the ERC Starting Grant TOTAL, Grant Agreement No 677651 (Ł.K., M.M), the ERC Starting Grant CUTACOMBS, Grant Agreement No 714704 (W.N., M.P., M.S.) and by the Polish National Science Center Grant PRELUDIM 2018/29/N/ST6/00676 (P.W.). The publication of proceedings of PACE 2020 has been supported by the University of Warsaw and the European Research Council (ERC) via European Union's Horizon 2020 research and innovation programme Grant Agreement no. 714704.

Acknowledgements The PACE challenge was supported by Networks [35] and University of Warsaw. The prize money (4000 €) was given through the generosity of Networks [35]. We are grateful to the whole `optil.io` team, led by Szymon Wasik, and especially to Jan Badura for the fruitful collaboration and for hosting the competition at `optil.io` on-line judge system. Special thanks go to Felix Reidl, who designed the eye-catching PACE 2020 poster.

1 Introduction

The Parameterized Algorithms and Computational Experiments Challenge (PACE) is an annually held algorithm engineering competition conceived in Fall 2015 to deepen the relationship between parameterized algorithms and practice.

So far, four iterations of PACE were organized. In each iteration, one or two NP-hard computational problems were chosen and the goal was to prepare an implementation which is able to solve (either exactly or approximately) challenging instances from various application domains in a decent time. The problems included treewidth [12,13], minimum feedback vertex set [12], minimum fill-in [13], Steiner tree [7], vertex cover [16], and hypertree width [16]. These challenges have a significant impact on the research community. Indeed, according to Google Scholar previous PACE reports are cited more than 90 times, in particular by research articles based on concrete implementations competing in previous editions of PACE, published in conferences like ALENEX, SEA, WADS, and ESA (where the best paper award was given in 2017 to a PACE-related work of H. Tamaki [50]).

In this article, we report on the fifth iteration of PACE. The topic of PACE 2020 was computing the treedepth of a graph (see Section 2 for a definition). The challenge was partitioned into two tracks. In the *exact track*, the implementations were supposed to return only optimal solutions and the goal was to maximize the number of solved instances (with total computation time as a tiebreaker). In the *heuristic track*, the implementations were supposed to solve larger instances than in the exact track, but non-optimal solutions were allowed and the goal was to provide solutions which are, on average, better than the solution of others.

The PACE 2020 challenge was announced on 25th October 2019. On December 16th public instances were made available and beginning from 13th March 2020 it was possible to test solutions on the public instances via the `optil.io` platform, which provided also a provisional ranking. The final version of the submissions was due on 1st June 2020. The results were announced on 24th June 2020. The award ceremony is going to take place during the International Symposium on Parameterized and Exact Computation (IPEC 2020) which was supposed to take place in Hong Kong, but due to the COVID-19 pandemic will be held online.

For the first time, short descriptions of the top five solvers in each track are contained as standalone documents in the proceedings of IPEC. Some of them will likely inspire or evolve into research papers. Indeed, at the moment of writing this report, we were already able to identify two such cases, see [53] and [62].

2 Computing Treedepth: Theory and Practice

Treedepth plays a major role in structural graph theory, in particular, the theory of sparse graph classes [31–33]. It is more restrictive than its more well-known counterparts *treewidth* and *pathwidth*, but still graphs of bounded treedepth form quite a rich family of graphs.

Definition

Remarkably, treedepth admits a number of equivalent definitions. Probably the best known is the one using embeddings into a rooted forest. A *rooted forest* is a graph whose every connected component is a tree with a designated root and the *depth* of a rooted forest is the maximum number of vertices on a root-to-leaf path. For a graph G, a *treedepth decomposition* of G consists of a rooted forest F and a bijection $\phi : V(G) \to V(F)$ such that for every

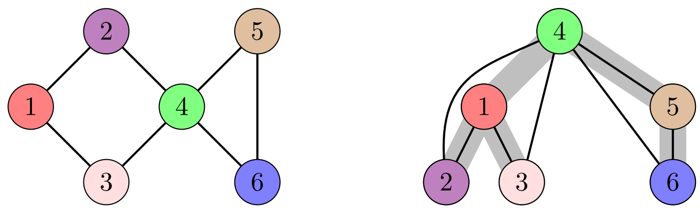

Figure 1 The thick gray tree on the right is an exemplary treedepth decomposition (aka elimination tree) of the graph on the left. Note that all graph edges go bottom-up in the tree.

$uv \in E(G)$, $\phi(u)$ and $\phi(v)$ are in descendant-ancestor relation in F. The *depth* of a treedepth decomposition (F, ϕ) is the depth of F and the *treedepth* of a graph G, denoted $\mathrm{td}(G)$, is the minimum possible depth of its treedepth decomposition.

A basic but important observation about treedepth is that if (F, ϕ) is a treedepth decomposition of a graph G and T is a tree in F with root r, then removing $\phi^{-1}(r)$ from G disconnects vertices whose images under ϕ are in different subtrees of T rooted in the children of r. This leads to the following equivalent recursive definition of treedepth:

$$\mathrm{td}(G) = \begin{cases} 0 & \text{if } V(G) = \emptyset, \\ 1 + \min_{v \in V(G)} \mathrm{td}(G - \{v\}) & \text{if } G \text{ is connected}, \\ \max_{C \in \mathtt{cc}(G)} \mathrm{td}(C) & \text{if } G \text{ is disconnected}. \end{cases}$$

Here, $\mathtt{cc}(G)$ denotes the family of connected components of G.

The above definition can be also interpreted as a game between two players, say Breaker and Chooser. The arena of the game is an induced subgraph of the input graph G, initially the whole graph G. At each round, Breaker first deletes a vertex of the current graph, and then Chooser restricts the arena to one of the connected components of the current graph. The game ends when the current graph becomes empty; Breaker wants to end the game in the minimum number of rounds, and Chooser in the maximum number of rounds. It is immediate from the above recursive definition of treedepth that, if both players play optimally, the game will end in exactly $\mathrm{td}(G)$ rounds.

Because of the above definition and the intuition of treedepth as an “elimination game”, where one can pay 1 to delete a vertex from the graph and then recurse independently over connected components, treedepth decompositions are sometimes called also *elimination forests* (or *elimination trees* if G is not connected).

Two related equivalent definitions of treedepth come from the theory of sparse graph classes. Let G be a graph. A function $\alpha : V(G) \to \mathbb{N}$ is a *centered coloring* if for every connected subgraph H of G, H contains a vertex of unique color, i.e., there is $i \in \mathbb{N}$ with $|\alpha^{-1}(i) \cap V(H)| = 1$. Furthermore, α is a *vertex ranking* if this unique color i is actually equal to $\max\{\alpha(v) \mid v \in V(H)\}$. In the context of a centered coloring, the values $\alpha(v)$ are called *colors* and in the context of a vertex ranking, they are called *ranks*. It is not difficult to see that the minimum number of colors used for a centered coloring of G and the minimum number of ranks used for a vertex ranking are both equal and equal to the treedepth of G.

Algorithms

From the theory point of view, the complexity of computing or approximating treedepth is much less understood than for treewidth.

The recursive definition of treedepth can be also interpreted as a recipe for a dynamic programming algorithm computing the treedepth of G. This yields a very simple $2^n \cdot n^{\mathcal{O}(1)}$-time algorithm.

Reidl et al. [37] described a dynamic programming algorithm that, given a graph G and a tree decomposition of G of width t, finds $\mathrm{td}(G)$ in time $2^{\mathcal{O}(\mathrm{td}(G) \cdot t)} n^{\mathcal{O}(1)}$. One can pipeline this algorithm with an approximation algorithm for treewidth, say constant-factor approximation algorithm running in time $2^{\mathcal{O}(\mathrm{tw}(G))} n^{\mathcal{O}(1)}$ [38] where $\mathrm{tw}(G)$ denotes the treewidth of G, obtaining an exact algorithm running in time $2^{\mathcal{O}(\mathrm{td}(G)\mathrm{tw}(G))} n^{\mathcal{O}(1)}$. This is the state-of-the-art as far as parameterized exact algorithms for treedepth (in theory) are concerned.

For practical approaches to computing treedepth exactly, we mention a work by Ganian, Lodha, Ordyniak, and Szeider [18] that experimented with encoding computing treedepth as SAT instances and using SAT solvers.

For approximation, a folklore observation (see [26] for a full proof) is that given a graph G and a tree decomposition of G of width t with tree T, one can in polynomial time find a treedepth decomposition of G of depth at most $(t + 1)\mathrm{td}(T)$. Since every tree decomposition of G can be simplified to use $\mathcal{O}(|V(G)|)$ nodes in its tree and every tree T has treedepth $\mathrm{td}(T) \leq \log_2(|V(T)| + 1)$ (which is easy to deduce from the recursive formula mentioned earlier), we obtain

$$\mathrm{td}(G) \leq (\mathrm{tw}(G) + 1) \cdot \mathcal{O}(\log_2(|V(G)|).$$

Combining the above with known treewidth approximation algorithms, one can obtain a polynomial-time $\mathcal{O}(\mathrm{tw}(G)^2 \log \mathrm{tw}(G))$-approximation for treedepth. The study of forbidden structures characterizations for treedepth led to an improved approximation guarantee of $\mathcal{O}(\mathrm{tw}(G) \log^{3/2} \mathrm{tw}(G))$ [9, 26]. Obtaining a constant-factor approximation for treedepth running in time say $2^{\mathcal{O}(\mathrm{td}(G))} n^{\mathcal{O}(1)}$ remains a challenging open problem.

3 The challenge setup

For each track, the PC selected 200 instances. The instances in each track were ordered lexicographically by non-decreasing (n, m) where n is the number of vertices and m is the number of edges. The odd-numbered instances were known to the participants five months before the submission deadline. The even-numbered instances were used to create the official ranking and they were secret until the results of the challenge were announced.

Both in the testing phase and for the final evaluation, the implementations were run for 30 minutes per instance using the `optil.io` on-line judge system [57]. For each instance, the available memory was limited to 8 GB.

In the exact track, the contestants were ranked by the number of instances solved and the total time required for the solved instances as a tiebreaker. Submissions for the exact track were supposed to be based on a provably optimal algorithm, although it was not a formal requirement. Instead, if a submission halted on some instance within the allotted time and output a solution that was worse than the best-known solution (from the PC's solver, other participants, or the way in which the instance was generated) then the submission was disqualified.

In the heuristic track, the goal was to maximize the total score and the score for a single instance for an implementation which returned a result of value d was determined by the formula $100 \cdot \min / d$, where min is the best (though not necessarily optimal) value obtained by any participating team. The reasons for selecting the formula were a) it does not award minor (say, additive) improvements too much (as compared to, e.g., ranking-based methods) and b) the score is within $(0, 100]$ always, so one very bad result does not make the submission to lose (as it could happen, say, for the inverse $d/\min$).

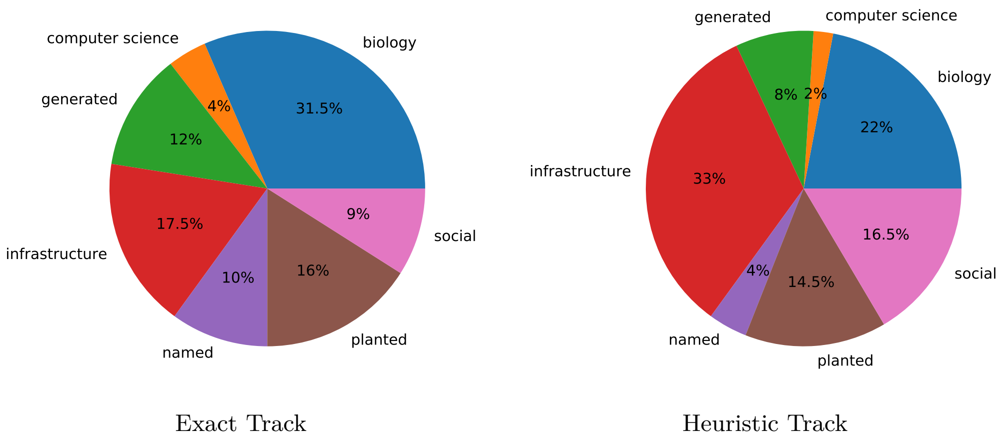

Figure 2 Distribution of origin categories of the test instances (both public and private).

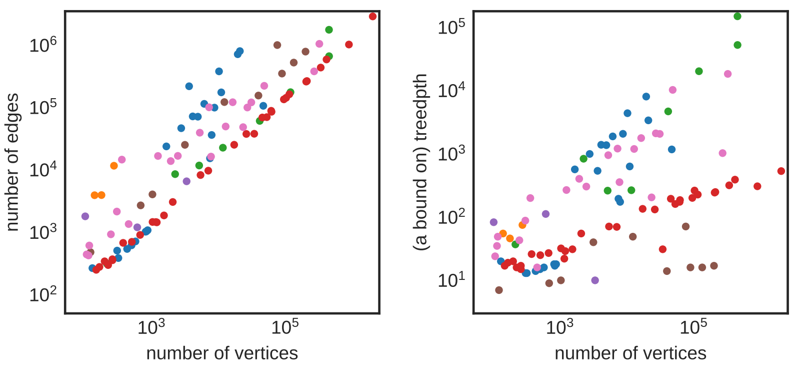

Figure 3 Distribution of origin categories, sizes, densities, and (a bound on) treedepth of the private test instances in the heuristic track. Colors correspond to the origin categories as in Figure 2.

Similarly as for previous editions of PACE, we required that the source code must be published in a public repository and available under an open source license. Following PACE 2019 we also allowed for external dependencies such as ILP, SAT, and treewidth solvers, provided that they were also open source.

4 Selection of the instances

All public and private instances used for PACE 2020 are available at a public repository at the address `https://github.com/lkowalik/Treedepth-PACE-2020-instances`. The set of collected instances contains graphs coming from various applications and graphs generated using a few generators. They can be divided into the following categories (see also Figure 2 for the distribution):

- **biology:** Graphs coming from applications in biology, biochemistry, and medicine. Downloaded from BioGRID [42], SNAP [63], ginsim.org [30], KEGG [24], STRING [48], and network repository [34].
- **computer science:** Control-flow graphs of C functions and graphs originating from register allocation for variables in real codes, created for DIMACS Coloring Challenge 1992–1993.

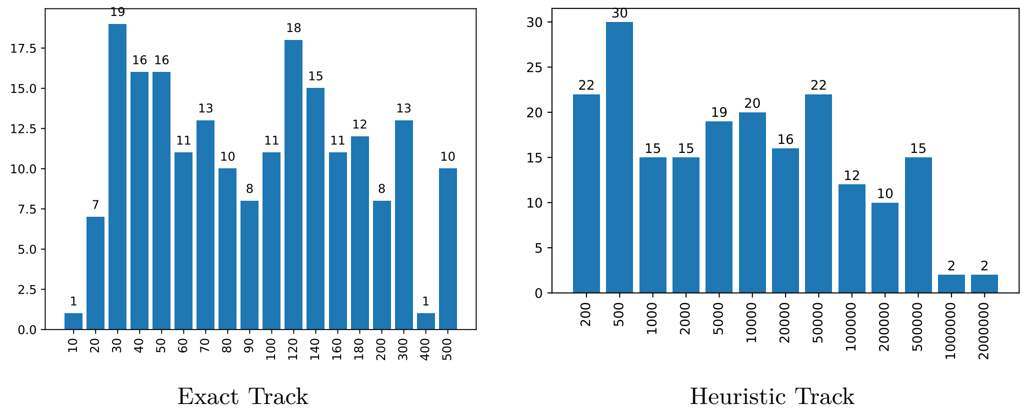

Figure 4 Distribution of sizes in the instance sets.

- **generated:** Graphs obtained from Python's networkx generators: expanders, grids, random cubic graphs, Waxman graphs (random geometric graphs).
- **infrastructure:** Mostly road graphs obtained from open street maps; also power grid networks (from network repository [34]) and public transport graphs contributed by Johannes Fichte for PACE 2016.
- **named:** Small named graphs like the Petersen graph, the flower snark, etc., originating from SageMath.
- **planted:** Random trees and cycles of cliques polluted with random edges that go bottom-up in the optimal treedepth decomposition. These instances were needed for testing correctness of treedepth solvers, because the generator was able to compute the optimum treedepth in polynomial time.
- **social:** Social networks originating from interactions between people, animals, or fictional characters.

Figures 4, 3, and 5 show the distribution of instance sizes and other characteristics depending on track.

5 Participants

There were 15 and 10 teams that officially submitted a solution to the exact and heuristic track, respectively. Five teams participated in both tracks, which gives 20 distinct teams. However, there were 38 more `optil.io` users that submitted a solution to the server during the testing phase (but none of them would be ranked in the top five solves for any track). The 20 teams represented three continents and 12 countries (see Table 1).

6 Exact Track

The results of the Exact Track are as follows.

1. James Trimble (University of Glasgow) solved 78 instances in 6502.97 seconds
   `github.com/jamestrimble/pace2020-treedepth-solvers` [54, 55]
2. Tuukka Korhonen (University of Helsinki) solved 77 instances in 5599.64 seconds
   `github.com/Laakeri/pace2020-treedepth-exact` [28, 29]

Table 1 A sorted table of the 20 participating teams' countries.

Country	Number of teams
Germany	4
Netherlands	4
Japan	2
United Kingdom	2
Brazil	1
Finland	1
France	1
India	1
Kosovo	1
Poland	1
Russia	1
Ukraine	1

3. Ruben Brokkelkamp, Mees de Vries, Raymond van Venetië, Jan Westerdiep (Centrum Wiskunde & Informatica (CWI), University of Amsterdam, Korteweg-de Vries Institute for Mathematics, University of Amsterdam) solved 72 instances in 3149.56 seconds
`github.com/mjdv/tdULL` [8, 11]
4. Max Bannach, Sebastian Berndt, Martin Schuster, Marcel Wienöbst (Institute for Theoretical Computer Science at Universität zu Lübeck, Institute for IT Security at Universität zu Lübeck, Institut für Epidemiologie at Universität Kiel) solved 72 instances in 4267.56 seconds
`github.com/maxbannach/PID-Star` [2, 4]
5. Dejun Mao, Vorapong Suppakitpaisarn, Zijian Xu (The University of Tokyo) solved 68 instances in 8794.42 seconds
`github.com/xuzijian629/pace2020` [60, 61]
6. Narek Bojikian, Alexander van der Grinten, Falko Hegerfeld, Laurence Alec Kluge, Stefan Kratsch (Humboldt-Universität zu Berlin) solved 64 instances in 4514.95 seconds
`github.com/PACE-Challenge-Hu-Berlin/PACE-Challenge-2020` [6]
7. Tom van der Zanden (Maastricht University) solved 44 instances in 6304.91 seconds
`github.com/TomvdZanden/BasicTreedepthSolver`
8. Dmitry Sayutin (ITMO University) solved 37 instances in 11465.50 seconds
`github.com/cdkrot/pace2020-sat-dp-solver` [39]
9. Philip de Bruin, Erik Jan van Leeuwen (Utrecht University) solved 27 instances in 4470.34 seconds
`github.com/PhiliPdB/treedepth-exact` [10]
10. Jun Kawahara, Toshiki Saitoh, Akira Suzuki, Toshiyuki Takase, Katsuhisa Yamanaka (Kyoto University, Kyushu Institute of Technology, Tohoku University, Iwate University) solved 6 instances in 198.43 seconds
`github.com/toshimaru0123/pace-2020/` [25]
11. Blend Arifaj, Ardit Baloku, Blend Berisha, Edon Gashi, Endrit Mëziu, Kadri Sylejmani (University of Prishtina) solved 0 instances in 0.00 seconds
`github.com/ksylejmani/treedepth-iterated-local-search`

The following teams submitted a solver, but, as described in the rules, it was disqualified because of at least one suboptimal solution. The number of solved instances reported below refers to the instances for which neither the PC nor any of the submitted solvers were able to produce a better solution.

- Miguel Bosch Calvo, Giorgia Carranza Tejada, Dominik Jeurissen, Steven Kelk, Zhuoer Ma, Alexander Reisach, Borislav Slavchev (Maastricht University) solved 79 instances in 138250.66 seconds
  `github.com/CommanderCero/Treedepth-Pace-2020`
- Sylwester Swat (Poznań University Of Technology) solved 74 instances in 128578.34 seconds
  `github.com/swacisko/pace-2020` [46]
- Marcelo Garlet Milani (Technische Universität Berlin) solved 40 instances in 1469.06 seconds
  `gitlab.tu-berlin.de/mgmillani1/treedepth-pace20` [19]
- Oleg Evseev, Igor Kozin, Alexander Zemlyanskiy (Zaporizhzhya National University) solved 8 instances in 241.10 seconds
  `github.com/oevseev97/pace-2020` [17]

6.1 Details of the solvers

The participants in the exact track used approaches that broadly fell into two categories: bottom-up and top-down.

Bottom-up approaches

In the bottom-up approach, the participants tried to find elimination trees for depths $k = 1, 2, 3, \ldots$ until they succeeded.

The bottom-up approach is to build minimum-depth elimination trees for induced subgraphs of the input graph iteratively from smaller depths to larger depths. Herein, a data structure keeps track of all the vertex sets S for which an elimination tree of $G[S]$ has been computed already. Then, in iterations over the data structure it is tested for which of the subgraphs in the data structure their elimination trees can be combined into an elimination tree of appropriate depth for the union of the subgraphs.

The bottom-up approach is akin to the positive-instance driven approach to dynamic programming [52]. Therein the goal is to avoid unnecessary work that is done in straightforward dynamic programs by carrying the dynamic program out in a forward-looking way. In the usual backward-looking approach we define a signature for subsolutions and we iterate over all signatures that are possible and, for each of them, check whether it is realized by some subsolution, by looking at signatures that are smaller in some well-defined sense and have been computed earlier. In the positive-instance driven way, instead, we directly generate from all the subsolutions that have been generated so far subsolutions which are larger in a well-defined sense. This intuitively avoids checking many signatures if there are only a few possible subsolutions.

The bottom-up approach was used by Trimble (1st place), Bannach et al. (4), Bojikian et al. (6), and van der Zanden (7). All the teams used a number of tricks to speed-up the basic approach and it seems that the winner collected most of them. Perhaps most obviously, many teams used known preprocessing rules [18, 27]. Both Trimble and Bannach et al. observed that the algorithm spends more and more time as k (the upper bound on the depth) grows, simply because then there are more partial solutions to consider. Both teams came up with the following remedy. First, they run a fast heuristic that computes a treedepth decomposition of some depth t. Then the original exact algorithm follows, with $k = 1, \ldots, t-1$. The hope is that the *optimum* depth is actually t and then we save on the (dominating) time needed for the last iteration. Other speed-ups involved pruning some subsolutions to consider, for example using a domination rule of Ganian et al. [18, Lemma 4.1], or the observation that for a subsolution X all neighbors $N(X)$ must be ancestors in the final elimination tree, so we can discard it when its treedepth plus $|N(X)|$ exceeds the target bound k.

Top-down approaches

In the top-down approach, we try to build an optimal elimination tree from the root to the leaves. In this approach, it is useful to observe that, barring the trivial case of cliques, the top vertices of the elimination tree can be chosen to be an inclusion-wise minimal separator of the input graph [14]. Hence, a natural idea is to enumerate all such separators, branch into all possibilities of taking such a separator for the top vertices of the elimination tree, and recurse into doing the same for each remaining connected component.

The top-down approach was taken by Korhonen (2nd place), Brokkelkamp et al. (3), Mao et al. (5), de Bruin and van Leeuwen (9) and Kawahara et al. (10), though the latter two teams did not use the minimal separators.

Generating minimal separators turned out to be a quite time consuming part of the approach, so different teams used different methods to cope with this issue. Brokkelkamp et al. used the standard minimal separators listing algorithm of Berry et al. [5]. Korhonen used an approach by Tamaki [51]. His algorithm solves the decision problem beginning with *high* upper bound on the treedepth k, e.g., $k = n$, and then improves k until possible. For enumerating minimal separators, first a fast heuristic algorithm [51, Section 4.3] is used, which may not find all the separators (but usually does so). It may happen that this already results in an improved upper bound for the treedepth. Otherwise, the algorithm is run for the second time, this time using a slower, but exact algorithm [51, Section 4.2] for the separator enumeration. Finally, Mao et al. used the space-efficient minimal separator listing algorithm of Takata [49]. Moreover, they skip separators of a size larger than the treewidth of the current graph, since they conjecture that this does not change the resulting treedepth. (Note that the correctness of their algorithm relies on this conjecture.)

Most of the top-down solvers used memoization, i.e., storing upper or lower bounds for subproblems to avoid repeated computation.

Another common technique used by all best solvers using the top-down approach is branch-and-bound, i.e., pruning the branching tree by using lower and upper bounds for the treedepth of subproblems. The combined arsenal of lower bound techniques includes memoized lower bounds for subgraphs, finding a long path or cycle, clique minors, and degeneracy. Korhonen used an interesting upper bound technique. It begins by finding a chordal completion of the graph. A fast heuristic upper bound algorithm is run on the resulting graph, in particular, utilizing the fact that chordal graphs have only a linear number of minimal separators.

Surprisingly, at least according to the submitted descriptions, it seems that only Korhonen applies preprocessing. Namely, he compresses induced subtrees connected to the rest of the graph by a single vertex (using the linear time algorithm for trees of Schäffer [40]) and also generalizes the shared neighborhood rule of Kobayashi and Tamaki [27, Lemma 6].

Table 2 Differences between the top five teams in the exact track (columns contain instances).

Name	068	074	084	088	090	094	108	112	120	148	150	174	180	182	186
Trimble	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓		✓	✓	
Korhonen	✓	✓	✓	✓		✓	✓		✓	✓	✓	✓		✓	
Brokkelkamp et al.		✓		✓						✓			✓	✓	✓
Bannach et al.				✓			✓		✓	✓			✓	✓	
Mao et al.		✓									✓				

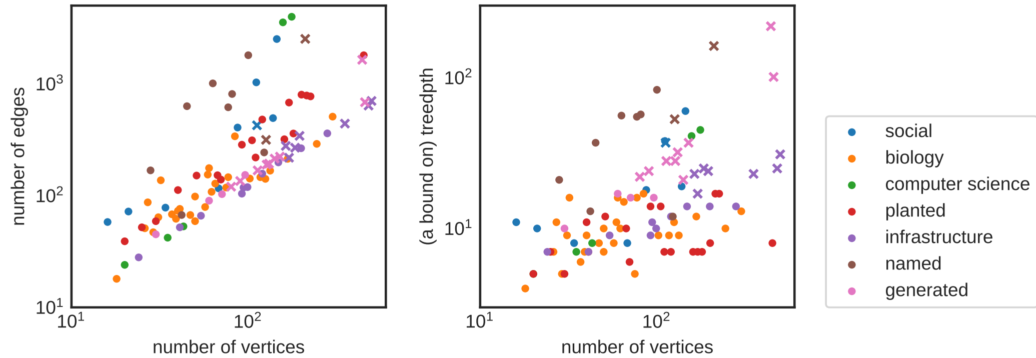

Figure 5 Instances of the exact track: number of edges vs. vertices (left) and (an upper bound on) treedepth vs. vertices (right). Crosses denote instances that were *not* solved by any team.

Other approaches

The solver of Sayutin used the standard $O^*(2^n)$ dynamic programming for small n and the SAT-encoding of Ganian et al. [18]. The solver of Milani implemented the $2^{\mathcal{O}(\mathrm{td}(G)\mathrm{tw}(G))}n^{\mathcal{O}(1)}$ algorithm using dynamic programming over tree decomposition [37] of Reidl et al. The disqualified solvers of Calvo et al. and Swat used heuristic methods (they were written mainly for the heuristic track).

6.1.1 Summary

The top ranked solvers used an impressive number of new and existing ideas. It should be noted that, when properly optimized, both bottom-up and top-down approaches seem to give similar results. Indeed, the solver of Trimble solved just one instance more than the one of Korhonen. This is in strong contrast with exact treewidth computation, where the bottom-up positive-instance driven approach outperforms other known methods (see [13, 52]).

6.2 A closer look at the results of the exact track

Altogether 81 out of the 100 private instances were solved by the participants, which means that the winning team has not solved three instances solved by others. Table 2 shows the differences between the top five teams in the exact track (a common subset of 66 tests was solved by all of them). The participants managed to solve all the tests with less than 80 vertices. The smallest treedepth of an *unsolved* instance was at most 17 (a 170-vertex road network), i.e., we computed an upper bound of 17 by a heuristic solver. The largest treedepth of a *solved* instance was 83 (the 100-vertex Hall-Janko graph). The plots in Figure 5 present solved and unsolved instances divided into categories.

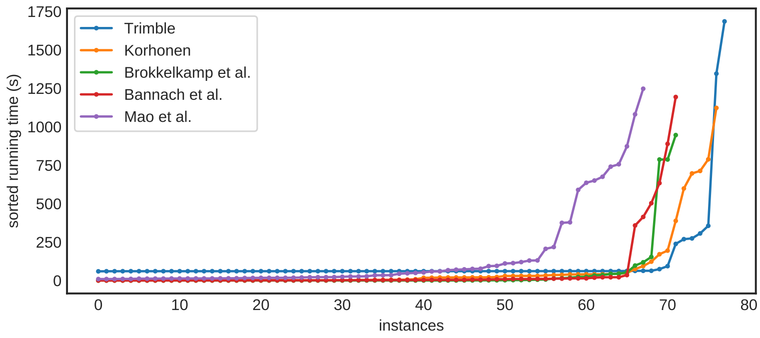

Figure 6 Running times of the five top solvers in the exact track. For each solver, all the instances solved by it were sorted by the running time.

It might be also interesting to look at the *running times* of the solvers. The plot in Figure 6 suggests that the top solvers needed substantial time only for a small fraction of solved instances.

7 Heuristic Track

The results of the Heuristic Track are as follows.

1. Sylwester Swat (Poznań University Of Technology) scored 9710.94 points
   `github.com/swacisko/pace-2020` [46,47]
2. Ben Strasser scored 9684.10 points
   `github.com/ben-strasser/flow-cutter-pace20` [44,45]
3. Marcin Wrochna (University of Oxford) scored 9591.21 points
   `github.com/marcinwrochna/sallow` [58,59]
4. James Trimble (University of Glasgow) scored 9447.96 points
   `github.com/jamestrimble/pace2020-treedepth-solvers` [54,56]
5. Max Bannach, Sebastian Berndt, Martin Schuster, Marcel Wienöbst (Institute for Theoretical Computer Science at Universität zu Lübeck, Institute for IT Security at Universität zu Lübeck, Institut für Epidemiologie at Universität Kiel) scored 8935.63 points
   `github.com/maxbannach/Fluid` [1,3]
6. Stéphane Grandcolas (LIS) scored 8880.58 points
   `gitlab.lis-lab.fr/stephane.grandcolas/treedepth-sga/-/tree/master/pace-2020` [22]
7. Miguel Bosch Calvo, Giorgia Carranza Tejada, Dominik Jeurissen, Steven Kelk, Zhuoer Ma, Alexander Reisach, Borislav Slavchev (Maastricht University) scored 6320.19 points
   `github.com/CommanderCero/Treedepth-Pace-2020`
8. Gabriel Duarte, Uéverton Souza, Samuel Silva (Fluminense Federal University) scored 5068.50 points
   `github.com/SamuelEduardoSilva/pace-2020` [15]
9. Aman Singal (Indian Institute of Technology Dharwad) scored 4254.87 points
   `github.com/AmanSingal/pace-2020-submission1` [41]
10. Oleg Evseev, Igor Kozin, Alexander Zemlyanskiy (Zaporizhzhya National University) scored 1071.72 points
   `github.com/oevseev97/pace-2020` [17]

7.1 Details of the solvers

In the heuristic track, similarly as in the exact track, it was not sufficient to come up with a single good idea. All the top solvers used a portfolio of many approaches. This was also forced by the test dataset in which number of vertices varied from 100 to 2 000 000 and treedepth ranged from 7 to up to 150 000. Clearly, in very large graphs the time limit allows only for simple and fast heuristics, while in small and medium instances one can try also more time consuming and possibly more meaningful computation. In contrast to the exact track, here one can try several approaches and output the best result, and this property was frequently used. Again, the most natural classification of methods is bottom-up and top-down processing.

Bottom-up heuristics

The basic scheme of a bottom-up heuristic is to find a so-called *elimination order*, as follows. Pick a vertex v in G (which minimizes some heuristic score), remove v from the graph and connect its remaining neighbors into a clique, obtaining graph G'. Put v in the end of the ordering and find the rest of the ordering recursively. In the resulting treedepth decomposition, the neighbor of v in G' which is the latest in the order becomes v's parent.

In the classic application of the above strategy [21] we always choose a vertex v of minimum degree in the current graph. In the context of treedepth, this is a meaningful measure, because it is a lower bound on the number of ancestors of v. However, one should also take into account the height of v, defined as the maximum of the heights of its eliminated neighbors plus 1, which represents the depth of the subtree rooted at v if it is eliminated at the given moment. Strasser (place 2) and Wrochna (3) used a linear combination of these two values as the vertex score. Wrochna designed also a few increasingly faster and more approximate versions of this approach.

Top-down heuristics

Treedepth can be defined by removing one vertex and recursing to connected components of what remained. However, it is a pretty rare case that by removing just one vertex we get a nice partition into many connected components. If we "unravel" this recursion we may come to a conclusion that it is good to think about removing at once whole *sets* of vertices that are in some way good separators. Ideally, we would like to drive our search of separators by breaking the graph into components with smaller treedepth, but we do not have the desired knowledge about the treedepth of subgraphs, so we need to measure the complexity of subproblems in a different way, for example, by the number of their vertices or edges or some other measure that is easily computed. In theory, approaches driven by an assumption that the treedepth and, say, the number of vertices are closely related can be easily fooled by some hand-crafted instances, but our test dataset was not constructed in such a way, as it contained mostly instances from real-life applications. This motivates an approach where we search for some balanced vertex cuts in our heuristic solver. More precisely, we would like these cuts to be both relatively small and partitioning our graph into components which are significantly smaller than the original graph. This general approach is usually called nested dissection [20].

The participants used various approaches for extracting balanced separators. The most common approach was FlowCutter [23, 43], used in particular by all three top solvers. It uses a maximum flow algorithm to find a minimum cut, and then refines the balance of the

Table 3 Top five solvers. The first column is the number of instances solved *not worse* than any other team. The second column is the number of instances solved *better* than any other team.

Name	The best (with ties)	Single best (no ties)
Swat	51	22
Strasser	47	8
Wrochna	46	7
Trimble	32	1
Bannach et al.	10	0

cut by subsequent flow computations. As a result it identifies a family of cuts with good trade-offs between the size of the cut and its balance. Frequently, the teams used more than one strategy to find separators. For example the winning solver of Swat uses four more ad-hoc separator-finding heuristics, based on articulation points, BFS, and flows. Strasser (2) proposes a local-search heuristic that starts from a cut and improves its size and balance step by step. Bannach et al. (5) use two strategies: an ad-hoc greedy algorithm and a community detection heuristic [36].

The solver of Strasser detects whether the graph passed to the recursive procedure is a clique or a tree and, if so, computes the optimal treedepth (for trees using Schäffer [40]).

Most of the teams just run the whole algorithm from scratch several times, every time using a different balanced separator approach, or different parameters, or a different random seed. However, the winning solver of Swat uses deviating scheme. It always computes many separators, next chooses five of them using a separator scoring function, then attempts to further improve each of the five separators, to finally choose the best of them and recurse.

We also note that Calvo et al. (7), Duarte et al. (8), and Singal (9) used the top-down approach based on removing single vertices (instead of separators), for example, with high centrality measures. However, there was a significant gap between the scores of these solvers and approaches that applied removing separators (1-6), at least as one of the options in their portfolio.

Preprocessing and postprocessing

A bit surprisingly, very few teams used preprocessing. However, Swat (1), Trimble (4), and Grandcolas (6) apply post-processing to see if the resulting tree can be easily improved.

7.2 A closer look at the results of the heuristic track

A closer look at the results obtained by the teams at particular tests shows that there was no single team that dominated the others, in particular each of the top four teams has solved at least one instance *better* than all the others (see Table 3). However, the depths returned by the winning solver of Swat were always within the ratio of 1.13 to the output of any other solver, see Figures 7 and 8. (In Table 3 and Figures 7 and 8 we take into account all the submissions to `optil.io`, including the teams who have not made an official submission to PACE.)

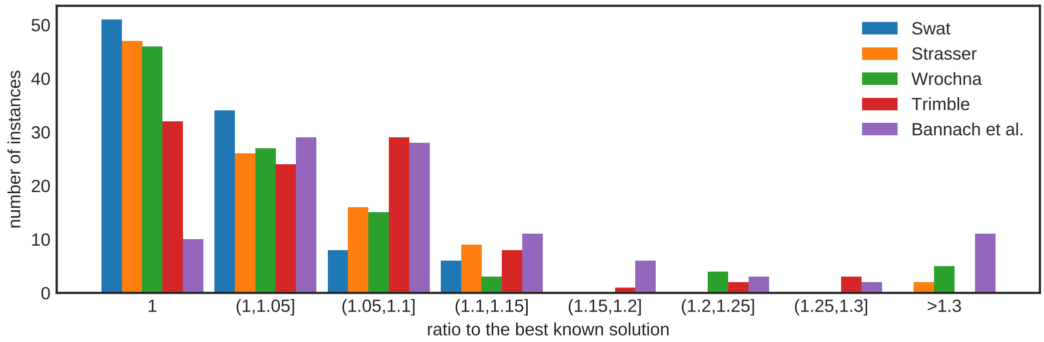

Figure 7 Comparison of the five top solvers in the heuristic track: how many tests they solved within the given ratio to the minimum depth reported by any team.

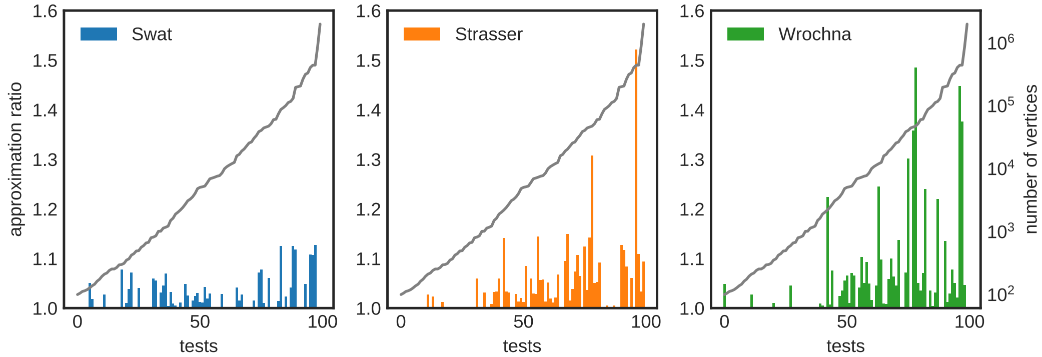

Figure 8 Comparison of the three top solvers in the heuristic track. A bar for team α for test i represents the ratio of the depth of the elimination tree for instance i found by α to the best depth found by of *any other team* for i. In particular no bar means that the team has found the minimum. The gray curve represents size of the tests (in the number of vertices), in the logarithmic scale.

8 PACE organization

The Program Committee of PACE 2020 consisted of Łukasz Kowalik (chair), Marcin Mucha, Wojciech Nadara, Marcin Pilipczuk, Manuel Sorge and Piotr Wygocki, all from University of Warsaw. During the organization of PACE 2020 the Steering Committee was as follows.

Édouard Bonnet	ENS Lyon
Holger Dell	Saarland Informatics Campus
Johannes Fichte	Technische Universität Dresden
Markus Hecher	Technische Universität Wien
Bart M. P. Jansen (chair)	Eindhoven University of Technology
Petteri Kaski	Aalto University
Christian Komusiewicz	Philipps-Universität Marburg
Florian Sikora	Paris-Dauphine University

In October 2020, Łukasz Kowalik, Marcin Pilipczuk, and Manuel Sorge have joined the SC, while Christian Komusiewicz, Florian Sikora, and Petteri Kaski left.

The Program Committee of PACE 2021 will be chaired by André Nichterlein (TU Berlin).

9 Conclusion

We thank all the participants for their impressive work and look forward to the next PACE.

We welcome anyone who is interested to add their name to the mailing list on the website `https://pacechallenge.org/` to receive PACE updates and join the discussion. The updates appear also at Twitter at `https://twitter.com/pace_challenge`. In particular, plans for PACE 2021 will be posted there.

References

1 Max Bannach. maxbannach/fluid: pace-2020, June 2020. doi:10.5281/zenodo.3871709.

2 Max Bannach. maxbannach/pid-star: pace-2020, June 2020. doi:10.5281/zenodo.3871800.

3 Max Bannach, Sebastian Berndt, Martin Schuster, and Marcel Wienöbst. PACE solver description: Fluid. In Yixin Cao and Marcin Pilipczuk, editors, *15th International Symposium on Parameterized and Exact Computation, IPEC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*, volume 180 of *LIPIcs*, pages 27:1–27:3. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.IPEC.2020.27.

4 Max Bannach, Sebastian Berndt, Martin Schuster, and Marcel Wienöbst. PACE solver description: PID*. In Yixin Cao and Marcin Pilipczuk, editors, *15th International Symposium on Parameterized and Exact Computation, IPEC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*, volume 180 of *LIPIcs*, pages 28:1–28:4. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.IPEC.2020.28.

5 Anne Berry, Jean Paul Bordat, and Olivier Cogis. Generating all the minimal separators of a graph. *Int. J. Found. Comput. Sci.*, 11(3):397–403, 2000. doi:10.1142/S0129054100000211.

6 Narek Bojikian, Alexander van der Grinten, Falko Hegerfeld, Laurence Alec Kluge, and Stefan Kratsch. Pace-challenge-2020-hu-berlin-exact-track, June 2020. doi:10.5281/zenodo.3894555.

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

8 Ruben Brokkelkamp, Raymond van Venetië, Mees de Vries, and Jan Westerdiep. PACE solver description: tdull. In Yixin Cao and Marcin Pilipczuk, editors, *15th International Symposium on Parameterized and Exact Computation, IPEC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*, volume 180 of *LIPIcs*, pages 29:1–29:4. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.IPEC.2020.29.

9 Wojciech Czerwinski, Wojciech Nadara, and Marcin Pilipczuk. Improved bounds for the excluded-minor approximation of treedepth. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPIcs*, pages 34:1–34:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ESA.2019.34.

10 Philip de Bruin. Philipdb/treedepth-exact: Submission for pace, May 2020. doi:10.5281/zenodo.3866006.

11 Mees de Vries, Ruben Brokkelkamp, Raymond van Venetië, and Jan Westerdiep. tdull, June 2020. doi:10.5281/zenodo.3881472.

12 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, *Proceedings of the 11th International Symposium on Parameterized and Exact Computation (IPEC 2016)*, volume 63 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:9, Dagstuhl, Germany, 2017. doi:10.4230/LIPIcs.IPEC.2016.30.

13 Holger Dell, Christian Komusiewicz, Nimrod Talmon, and Mathias Weller. The PACE 2017 parameterized algorithms and computational experiments challenge: The second iteration. In *12th International Symposium on Parameterized and Exact Computation, IPEC 2017, September 6-8, 2017, Vienna, Austria*, pages 30:1–30:12, 2017. doi:10.4230/LIPIcs.IPEC.2017.30.

14 Jitender S. Deogun, Ton Kloks, Dieter Kratsch, and Haiko Müller. On the vertex ranking problem for trapezoid, circular-arc and other graphs. *Discret. Appl. Math.*, 98(1-2):39–63, 1999. doi:10.1016/S0166-218X(99)00179-1.

15 Gabriel Duarte, Samuel Eduardo, and Uéverton Souza. Ufffptteam, June 2020. doi:10.5281/zenodo.3872029.

16 M. Ayaz Dzulfikar, Johannes K. Fichte, and Markus Hecher. The PACE 2019 Parameterized Algorithms and Computational Experiments Challenge: The Fourth Iteration (Invited Paper). In Bart M. P. Jansen and Jan Arne Telle, editors, *14th International Symposium on Parameterized and Exact Computation (IPEC 2019)*, volume 148 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:23, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.IPEC.2019.25.

17 Oleg Evsees, Alexander Zemlyanskiy, and Igor Kozin. Gensol.exe, June 2020. doi:10.5281/zenodo.3882767.

18 Robert Ganian, Neha Lodha, Sebastian Ordyniak, and Stefan Szeider. Sat-encodings for treecut width and treedepth. In Stephen G. Kobourov and Henning Meyerhenke, editors, *Proceedings of the Twenty-First Workshop on Algorithm Engineering and Experiments, ALENEX 2019, San Diego, CA, USA, January 7-8, 2019*, pages 117–129. SIAM, 2019. doi:10.1137/1.9781611975499.10.

19 Marcelo Garlet Milani. td-milani: A treedepth solver, June 2020. doi:10.5281/zenodo.3885116.

20 Alan George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, 10(2):345–363, 1973. doi:10.1137/0710032.

21 Alan George and Joseph W.H. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Review*, 31(1):1–19, 1989. doi:10.1137/1031001.

22 Stéphane Grandcolas. sga, June 2020. doi:10.5281/zenodo.3884054.

23 Michael Hamann and Ben Strasser. Graph bisection with pareto optimization. *ACM J. Exp. Algorithmics*, 23, 2018. doi:10.1145/3173045.

24 Minoru Kanehisa and Susumu Goto. KEGG: Kyoto encyclopedia of genes and genomes. *Nucleic acids research*, 28(1):27–30, 2000.

25 Jun Kawahara, Toshiki Saitoh, Akira Suzuki, Toshiyuki Takase, and Katsuhisa Yamanaka. wankosoba: Exact treedepth decomposition solver, June 2020. doi:10.5281/zenodo.3894127.

26 Ken-ichi Kawarabayashi and Benjamin Rossman. A polynomial excluded-minor approximation of treedepth. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 234–246. SIAM, 2018. doi:10.1137/1.9781611975031.17.

27 Yasuaki Kobayashi and Hisao Tamaki. Treedepth parameterized by vertex cover number. In Jiong Guo and Danny Hermelin, editors, *11th International Symposium on Parameterized and Exact Computation, IPEC 2016, August 24-26, 2016, Aarhus, Denmark*, volume 63 of *LIPIcs*, pages 18:1–18:11. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.IPEC.2016.18.

28 Tuukka Korhonen. Pace 2020 exact treedepth submission: sms, June 2020. doi:10.5281/zenodo.3872898.

29 Tuukka Korhonen. PACE solver description: Sms. In Yixin Cao and Marcin Pilipczuk, editors, *15th International Symposium on Parameterized and Exact Computation, IPEC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*, volume 180 of *LIPIcs*, pages 30:1–30:4. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.IPEC.2020.30.

30 A. Naldi, D. Berenguier, A. Fauré, F. Lopez, D. Thieffry, and C. Chaouiya. Logical modelling of regulatory networks with ginsim 2.3. *Biosystems*, 97(2):134–139, 2009. doi:10.1016/j.biosystems.2009.04.008.

31 Jaroslav Nesetril and Patrice Ossona de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *Eur. J. Comb.*, 27(6):1022–1041, 2006. doi:10.1016/j.ejc.2005.01.010.

32 Jaroslav Nesetril and Patrice Ossona de Mendez. *Sparsity - Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012. doi:10.1007/978-3-642-27875-4.

33 Jaroslav Nesetril and Patrice Ossona de Mendez. On low tree-depth decompositions. *Graphs and Combinatorics*, 31(6):1941–1963, 2015. doi:10.1007/s00373-015-1569-7.

34 Network repository. URL: http://networkrepository.com/.

35 Networks project, 2017. URL: http://www.thenetworkcenter.nl.

36 Ferran Parés, Dario Garcia Gasulla, Armand Vilalta, Jonatan Moreno, Eduard Ayguadé, Jesús Labarta, Ulises Cortés, and Toyotaro Suzumura. Fluid communities: A competitive, scalable and diverse community detection algorithm. In Chantal Cherifi, Hocine Cherifi, Márton Karsai, and Mirco Musolesi, editors, *Complex Networks & Their Applications VI*, pages 229–240, Cham, 2018. Springer International Publishing.

37 Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, and Somnath Sikdar. A faster parameterized algorithm for treedepth. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 931–942. Springer, 2014. doi:10.1007/978-3-662-43948-7_77.

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

39 Dmitry Sayutin. PACE 2020 Magnolia Tree Depth solver (DP & Sat based), June 2020. doi:10.5281/zenodo.3888908.

40 Alejandro A. Schäffer. Optimal node ranking of trees in linear time. *Inf. Process. Lett.*, 33(2):91–96, 1989. doi:10.1016/0020-0190(89)90161-0.

41 Aman Singal. Pace2020 - heuristic treedepth, June 2020. doi:10.5281/zenodo.3871918.

42 Chris Stark, Bobby-Joe Breitkreutz, Teresa Reguly, Lorrie Boucher, Ashton Breitkreutz, and Mike Tyers. Biogrid: a general repository for interaction datasets. *Nucleic acids research*, 34:D535–D539, 2006.

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

44 Ben Strasser. Flowcutter pace 2020 submission, May 2020. doi:10.5281/zenodo.3870928.

45 Ben Strasser. PACE solver description: Tree depth with flowcutter. In Yixin Cao and Marcin Pilipczuk, editors, *15th International Symposium on Parameterized and Exact Computation, IPEC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*, volume 180 of *LIPIcs*, pages 32:1–32:4. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.IPEC.2020.32.

46 Sylwester Swat. swacisko/pace-2020: First release of extreem, June 2020. doi:10.5281/zenodo.3873126.

47 Sylwester Swat. PACE solver description: Finding elimination trees using extreem - a heuristic solver for the treedepth decomposition problem. In Yixin Cao and Marcin Pilipczuk, editors, *15th International Symposium on Parameterized and Exact Computation, IPEC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*, volume 180 of *LIPIcs*, pages 33:1–33:4. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.IPEC.2020.33.

48 Damian Szklarczyk, Annika L Gable, David Lyon, Alexander Junge, Stefan Wyder, Jaime Huerta-Cepas, Milan Simonovic, Nadezhda T Doncheva, John H Morris, Peer Bork, et al. String v11: protein–protein association networks with increased coverage, supporting functional discovery in genome-wide experimental datasets. *Nucleic acids research*, 47(D1):D607–D613, 2019.

49 Ken Takata. Space-optimal, backtracking algorithms to list the minimal vertex separators of a graph. *Discret. Appl. Math.*, 158(15):1660–1667, 2010. doi:10.1016/j.dam.2010.05.013.

50 Hisao Tamaki. Positive-instance driven dynamic programming for treewidth. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms (ESA 2017)*, volume 87 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 68:1–68:13, Dagstuhl, Germany, 2017. doi:10.4230/LIPIcs.ESA.2017.68.

51 Hisao Tamaki. Computing treewidth via exact and heuristic lists of minimal separators. In Ilias S. Kotsireas, Panos M. Pardalos, Konstantinos E. Parsopoulos, Dimitris Souravlias, and Arsenis Tsokas, editors, *Analysis of Experimental Algorithms - Special Event, SEA^2 2019, Kalamata, Greece, June 24-29, 2019, Revised Selected Papers*, volume 11544 of *Lecture Notes in Computer Science*, pages 219–236. Springer, 2019. doi:10.1007/978-3-030-34029-2_15.

52 Hisao Tamaki. Positive-instance driven dynamic programming for treewidth. *Journal of Combinatorial Optimization*, 37(4):1283–1311, 2019. doi:10.1007/s10878-018-0353-z.

53 James Trimble. An algorithm for the exact treedepth problem. In Simone Faro and Domenico Cantone, editors, *18th International Symposium on Experimental Algorithms, SEA 2020, June 16-18, 2020, Catania, Italy*, volume 160 of *LIPIcs*, pages 19:1–19:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.SEA.2020.19.

54 James Trimble. jamestrimble/pace2020-treedepth-solvers: v1.0.0, June 2020. doi:10.5281/zenodo.3881441.

55 James Trimble. PACE solver description: Bute-plus: A bottom-up exact solver for treedepth. In Yixin Cao and Marcin Pilipczuk, editors, *15th International Symposium on Parameterized and Exact Computation, IPEC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*, volume 180 of *LIPIcs*, pages 34:1–34:4. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.IPEC.2020.34.

56 James Trimble. PACE solver description: Tweed-plus: A subtree-improving heuristic solver for treedepth. In Yixin Cao and Marcin Pilipczuk, editors, *15th International Symposium on Parameterized and Exact Computation, IPEC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*, volume 180 of *LIPIcs*, pages 35:1–35:4. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.IPEC.2020.35.

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

58 Marcin Wrochna. marcinwrochna/sallow: pace-2020, May 2020. doi:10.5281/zenodo.3870565.

59 Marcin Wrochna. PACE solver description: Sallow: a heuristic algorithm for treedepth decompositions. In Yixin Cao and Marcin Pilipczuk, editors, *15th International Symposium on Parameterized and Exact Computation, IPEC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*, volume 180 of *LIPIcs*, pages 36:1–36:4. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.IPEC.2020.36.

60 Zijian Xu, Dejun Mao, and Vorapong Suppakitpaisarn. PACE solver description: Computing exact treedepth via minimal separators. In Yixin Cao and Marcin Pilipczuk, editors, *15th International Symposium on Parameterized and Exact Computation, IPEC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*, volume 180 of *LIPIcs*, pages 31:1–31:4. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.IPEC.2020.xx.

61 Zijian Xu, Dejun Mao, and Vorapong Suppakitpaisarn. solver from xuzijian629, May 2020. doi:10.5281/zenodo.3870624.

62 Zijian Xu and Vorapong Suppakitpaisarn. On the size of minimal separators for treedepth decomposition, 2020. arXiv:2008.09822.

63 Marinka Zitnik, Rok Sosič, Sagar Maheshwari, and Jure Leskovec. BioSNAP Datasets: Stanford biomedical network dataset collection. http://snap.stanford.edu/biodata, August 2018.