The PACE 2017 Parameterized Algorithms and Computational Experiments Challenge: The Second Iteration

Holger Dell¹, Christian Komusiewicz², Nimrod Talmon³, and Mathias Weller⁴

¹ Saarland University and Cluster of Excellence (MMCI), Saarbrücken, Germany
hdell@mmci.uni-saarland.de

² Friedrich-Schiller-University Jena, Germany
christian.komusiewicz@uni-jena.de

³ Weizmann Institute of Science, Rehovot, Israel
nimrod.talmon@weizmann.ac.il

⁴ Laboratory of Informatics, Robotics, and Microelectronics of Montpellier (LIRMM), France
weller@lirmm.fr

Abstract

In this article, the Program Committee of the Second Parameterized Algorithms and Computational Experiments challenge (PACE 2017) reports on the second iteration of the PACE challenge. Track A featured the Treewidth problem and Track B the Minimum Fill-In problem. Over 44 participants on 17 teams from 11 countries submitted their implementations to the competition.

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

Keywords and phrases treewidth, minimum fill-in, contest, implementation challenge, FPT

Digital Object Identifier 10.4230/LIPIcs.IPEC.2017.30

1 Introduction

The Parameterized Algorithms and Computational Experiments Challenge (PACE) was conceived in Fall 2015 to help deepen the relationship between parameterized algorithmics and practice. 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.

* The PACE challenge was supported by Networks, 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 first iteration of PACE was held in 2016 [10] and has met many of its aims. For instance, PACE is mentioned as inspiration in numerous papers [1, 2, 12, 13, 17, 19, 23, 31, 34, 35]. Here, we report on the second iteration of PACE.

The PACE 2017 challenge was announced on December 1, 2016 with two tracks: Track A (Treewidth) and Track B (Minimum Fill-In). The final version of the submissions was due on May 1, 2017. We informed the participants of the result on June 1, 2017, and announced them to the public on September 6, 2017, during the award ceremony at the International Symposium on Parameterized and Exact Computation (IPEC 2017) in Vienna.

2 Competition track A: Treewidth

The objective of this track is to compute a minimum tree-decomposition of a given graph:

**Input:** An undirected graph.

**Output:** A minimum-width tree-decomposition of the graph.

The treewidth, which is the width of a minimum-width tree-decomposition, is one of the most important graph parameters in parameterized algorithms. Treewidth implementations are used in various contexts, for example in register allocation (e.g. [33]), shortest path computation (e.g. [9]), probabilistic inference (e.g. [21]), graph theory (e.g. [22]), and low-dimensional topology (e.g. [7]).

Last year’s PACE challenge achieved the first systematic comparison between different implementations that compute minimum tree decompositions [10]. Since treewidth is such a central problem, we wanted to see more improvement. We required all submission to be single-threaded (with the exception of wrapper threads forced by limitations of the programming language). The PACE 2017 treewidth track consisted of the following two independent competitions.

**Exact competition:** Compute a tree-decomposition of minimum width. You have 30 minutes per instance. Win by solving more instances than the other participants.

**Heuristic competition:** Compute some tree-decomposition. You have 30 minutes per instance. Win by printing solutions of smaller width than the other participants.

All instances used in the competition of PACE 2017 were derived from the following sources: transit and road networks that were submitted to PACE 2016, instances from the SAT competition, instances from the UAI 2014 inference competition, and some instances from treedecomposition.com. In contrast to the treewidth competition of PACE 2016, no randomly generated instances were used. We cleaned up all instances as follows. We deleted vertices of degree one. For vertices $v$ with exactly two non-adjacent neighbors $u$ and $w$, we deleted $v$ and added the edge $uw$. Finally, we kept only the largest connected component and deleted all other connected components. The base pool and all derived competition instances can be downloaded from github.com/PACE-challenge/Treewidth.

For the heuristic competition, we selected 200 instances he001.gr, ..., he200.gr from the base pool, ordered by file size and anonymized. For the exact competition, we systematically derived 200 instances ex001.gr, ..., ex200.gr from the base pool, roughly ordered by their believed hardness. We made the odd-numbered instances public when PACE 2017 was announced, and we kept the even-numbered instances secret until the submission deadline. The final competition and ranking was based only on the secret instances. Figure 1 shows some statistics for the two secret instance sets.
Figure 1 The PACE 2017 secret instance sets consist of the even-numbered instances ex002.gr, ... , ex200.gr (left) and he002.gr, ... , he200.gr (right). We plot the number $n$ of vertices and the number $m$ of edges. The exact competition contained instances whose treewidth was between 6 and 540, with a median of 11 and a mean of 31; the median best treewidth found for the heuristic competition was 93, and the mean was 13,038.

2.1 Exact competition: Compute an optimal tree decomposition

Three implementations were submitted to the exact treewidth competition of PACE 2017. Let us compare the three submissions to PACE 2017 and the winning submission from last year on a particular instance ex196.gr, which has 242 vertices, 443 edges, and treewidth 11:

<table>
<thead>
<tr>
<th>Running time on input ex196.gr (in seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>winner of PACE 2016</td>
</tr>
<tr>
<td>third place of PACE 2017</td>
</tr>
<tr>
<td>second place of PACE 2017</td>
</tr>
<tr>
<td>winner of PACE 2017</td>
</tr>
</tbody>
</table>

All three PACE 2017 submissions are two orders of magnitude faster than last year!

The instance ex196.gr was generated as follows: We started with the MMAP instance Promedas_30 from the UAI 2014 inference competition. Since many probabilistic inference algorithms first compute a good tree decomposition of the input, instances like Promedas_30 are great to include in the base pool for the treewidth competition. Exact treewidth solvers don’t scale enough yet to solve the entire instance, which has 1155 vertices and 2290 edges. It took the winning submission for the PACE 2017 heuristic treewidth challenge about 7 days to compute an upper bound of 51 on the treewidth of Promedas_30. Thus for the exact treewidth competition, we had to make the instance more tractable. To do so, a random center vertex was chosen and the graph induced by all vertices at distance at most $r$ from this center was kept. The instance generated this way was then cleaned again by taking care of vertices of degree one and two. The radius $r$ was increased as much as possible so that last year’s winning submission can solve the instance in just under two hours. The setting $r = 5$ gives rise to the graph ex196.gr.

All PACE 2017 instances for the exact treewidth competition were derived in a similar fashion from the base pool of instances and it took several CPU months to choose the radii appropriately. While the instances were chosen to be hard, we were surprised by the fantastic
improvements seen in treewidth implementations when compared to last year: most of the instances we generated can now be solved in a few seconds. The mean running time on the secret instances `ex002.gr`, `ex200.gr` was only 12 seconds, and the slowest was an instance solved in 162 seconds. As a consequence, the best two submissions solved all PACE 2017 competition instances within the allotted time of 30 minutes. In the announcement of the PACE 2017 competition, we defined the following tie-breaker rule for this situation: The winning solver is the one which is faster on most instances. The winning submission solved about 67% of the instances faster than the one we ranked second. Here is the final ranking:

1st place: Lukas Larisch (King-Abdullah University of Science and Engineering) and Felix Salfelder (University of Leeds) solved all 100 instances

2nd place: Hiromu Ohtsuka and Hisao Tamaki (Meiji University) solved all 100 instances

3rd place: Max Bannach (University of Lübeck), Sebastian Berndt (University of Lübeck), and Thorsten Ehlers (University of Kiel) solved 89 instances

While the solver by Larisch and Salfelder is fastest on most instances, its running time has a large variance and there are some instances where its running time almost reaches the allowed 30 minutes. The cumulative total time to solve all 100 instances was 4478 seconds for Larisch and Salfelder and 2747 seconds for Ohtsuka and Tamaki.

2.2 Heuristic competition: Compute some tree decomposition

There were six participating teams in the heuristic challenge. The selected instances were chosen from the largest few instances from each category in the base pool.

For the ranking, each instance was considered to be a “voter” where smaller width leads to a better rank for the submission; outputting no solution was ranked the same as outputting a trivial solution. To avoid misguided micro-optimization incentives, a solution was considered
Table 3 This table lists statistics on the solution quality for each competitor, across the even-numbered instances he002.gr, ..., he200.gr. The approximation ratio that a solver achieves on an instance is the fraction of the treewidth upper bound achieved by the solver divided by the best width achieved by any solver. The average approximation ratio takes the average across all 100 instances, while the maximum approximation ratio is the worst approximation ratio achieved across the instances. The four columns on the right display additive errors. For example, Tamaki et al. achieve the best treewidth upper bound on 51 of the instances, achieve at most the best plus one on 52 instances, achieve the best plus two on 52 instances, and obtain a non-trivial solution (width ≤ n - 5) on 92 instances. The best result in each column is highlighted in bold.

```
<table>
<thead>
<tr>
<th>Competitor</th>
<th>Approx. Ratio</th>
<th>Diff. to best Sol. [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Avg.</td>
<td>Max.</td>
</tr>
<tr>
<td>tamaki_tw-heuristic</td>
<td>1.30</td>
<td>9.61</td>
</tr>
<tr>
<td>strasser_flow_cutter_pace17</td>
<td>1.08</td>
<td>1.55</td>
</tr>
<tr>
<td>abseher_ltd_gr2td_exhaustive.sh</td>
<td>1.11</td>
<td>2.37</td>
</tr>
<tr>
<td>bannach_tw-heuristic</td>
<td>1.19</td>
<td>3.90</td>
</tr>
<tr>
<td>terrioux1_minfill_mrs</td>
<td>1.30</td>
<td>2.43</td>
</tr>
<tr>
<td>terrioux2_minfillbg_mrs</td>
<td>4.68</td>
<td>139.70</td>
</tr>
<tr>
<td>larisch_tw-heuristic</td>
<td>1.56</td>
<td>3.74</td>
</tr>
</tbody>
</table>
```

*Figure 3* This table lists statistics on the solution quality for each competitor, across the even-numbered instances he002.gr, ..., he200.gr. The approximation ratio that a solver achieves on an instance is the fraction of the treewidth upper bound achieved by the solver divided by the best width achieved by any solver. The average approximation ratio takes the average across all 100 instances, while the maximum approximation ratio is the worst approximation ratio achieved across the instances. The four columns on the right display additive errors. For example, Tamaki et al. achieve the best treewidth upper bound on 51 of the instances, achieve at most the best plus one on 52 instances, achieve the best plus two on 52 instances, and obtain a non-trivial solution (width ≤ n - 5) on 92 instances. The best result in each column is highlighted in bold.

The solver by Tamaki et al. dominates the one of Strasser on 51.65% of the instances. This means that, after subtracting the 9 instances on which they produce the same width, just a bit more than half of the remaining instances (namely, 47) are solved better by the first solver, and a bit less than half (namely, 44) are solved better by the second (see Figure 4). Strasser’s submission dominates the one of Abseher et al. on 52.05% of the non-tied instances. The top 3 submissions dominate the submission of Bannach et al. with 62%, and the top 4 submissions each dominate the remaining two submission by 90% or more.

The solver of Tamaki et al. is particularly good at small instances, where it is able to avoid off-by-one errors well. In fact, as can be seen from Figure 3, the Schulze method ranking
Figure 4 Comparison between Tamaki et al. and Strasser (top) and Strasser and Abseher et al. (bottom). For each even-numbered instance $\text{he}002, \ldots, \text{he}200$ (on the horizontal axis), we plot its number of edges in logscale (yellow line) and the best upper bound on the treewidth achieved during the competition (red line), also in logscale. It can be observed that the treewidth upper bound grows with the number of edges in our instance set. The blue bars depict the advantage for the first solver over the second (negative bars represent a disadvantage). More precisely, the advantage is $\varepsilon$ if $T = (1 - \varepsilon)S$ holds, where $T$ is the width produced by the first and $S$ is the width produced by the second solver. Interestingly, the solver of Tamaki et al. appears to do consistently well on the smaller instances, many of which were derived from the UAI 2014 inference competition and from treewidth.com, while Strasser’s submission performs consistently better on larger instance.

system has, in this case, punished off-by-one errors quite a bit. On very large instances, a more meaningful measure of quality would be the average or maximum approximation ratio achieved, where the solver of Strasser et al. excels.
3 Competition track B: Minimum fill-in

The objective of this track was to solve the NP-hard **Minimum Fill-In** problem, defined as follows.

**Input:** An undirected simple graph $G = (V, E)$.

**Output:** A minimum-size set of edges such that adding these edges to $G$ results in a simple chordal graph, that is, a graph in which every induced cycle has length exactly three.

**Minimum Fill-In** has applications in optimizing Gaussian elimination on sparse symmetric matrices and in computational phylogenetics [6, 18] and it is notorious for being one of the open problems in the first edition of Garey and Johnson’s monograph on NP-completeness [16]. Yannakakis later proved that the problem is NP-complete [36]. **Minimum Fill-In** is fixed-parameter tractable when parameterized by the number $k$ of edges that need to be added to make the graph chordal [8, 4, 15, 20]. The fastest parameterized algorithm for this parameter has a subexponential running time of $2^{o(k)} + O(k^2nm)$ [15]. The problem also admits a polynomial problem kernel with respect to this parameter [20, 26], the current best bound on the kernel size is $O(k^2)$ [26]. The best exact algorithms for solving **Minimum Fill-In** are based on the technique of dynamic programming over so-called potential maximal cliques [5, 14, 15].

**Test Instances**

The test instances were derived from several sources. In light of the application of optimizing Gaussian elimination, several instances were taken as-is from the Matrix Market [24] and the Network Repository [27]. In context of phylogenetic tree reconstruction, we used 39 alignments of different mammalian DNA markers obtained from the OrthoMaM database [30, 11]. Each alignment is then interpreted as a character-state matrix. Buneman’s theorem [6] states that each such character-state matrix allows reconstructing a perfect phylogeny if and only if the “character-state intersection graph” has a chordal supergraph with some specific properties. Thus, we wrote a program to convert the character-state matrices to character-state intersection graphs [25] and used these graphs for the competition. To allow for a smooth transition from easy to hard instances, we sampled connected subgraphs of the real-world instances whose density did not deviate too much from the density of the original instance. An overview of the number of vertices and edges in the set of hidden instances is shown in Figure 5. The 200 resulting graphs were divided into 100 public instances (that the participants could test their implementations on), and 100 hidden instances (used to determine the ranking of the submissions).

**Rules**

The PACE 2017 **Minimum Fill-In** track was a competition to solve as many test instances of **Minimum Fill-In** within the allowed time limit of 30 minutes per instance. The winners were selected based on the number of solved hidden instances. Concentrating on fixed-parameter tractability, we disallowed the use of SAT solvers, ILP solvers, or similar general solvers for NP-hard problems. Further, as the competition should be about exact solvers, submissions producing at least one incorrect or suboptimal solution were ranked below any other solvers producing only optimal solutions.

---

1 For each species $s$, the character-state intersection graph contains a clique of the vertices $(c, i)$ such that character $c$ has state $i$ in species $s$. 
Submissions

Eight implementations were submitted to the **Minimum Fill-in** competition of PACE 2017. Their ranking is given below, while we provide a brief explanation on the solutions of the three top-ranked submissions.

- **1st place**: Yasuaki Kobayashi (Kyoto University) and Hisao Tamaki (Meiji University). Their submission is written in Java and solved 54 of 100 instances (indeed, all but one instance that could not be solved by their submission could also not be solved by any other submission returning only optimal solutions). The submission first uses data reduction where some of the reduction rules are generalizations of rules developed by Bodlaender et al. [4]. Then the instance is decomposed using Tarjan’s clique decomposition algorithm [32]. The resulting instances are solved by adapting Tamaki’s modification [31] of Bouchité and Todinca’s dynamic programming algorithm [5]. The source code is available at [github.com/TCS-Meiji/PACE2017-TrackB](https://github.com/TCS-Meiji/PACE2017-TrackB).

- **2nd place**: Jeremias Berg, Matti Järvisalo, and Tuukka Korhonen (University of Helsinki). Their submission is written in C++ and solved 45 of 100 instances. The general idea of the submission is to use Tarjan’s clique decomposition [32], then to apply parts of the known kernelization algorithms, and finally to use dynamic programming over potential maximal cliques [5]. The source code is available at [github.com/Laakeri/PACE2017-min-fill](https://github.com/Laakeri/PACE2017-min-fill).

- **3rd place**: Édouard Bonnet (University Paris-Dauphine), R.B. Sandeep (Hungarian Academy of Sciences), and Florian Sikora (University Paris-Dauphine). Their submission is written in Python and solved 23 of 100 instances. The submission uses Tarjan’s clique decomposition followed by kernelization and dynamic programming over potential maximal cliques [5]. The source is available at [bitbucket.org/Florian_dauphine/mfi](https://bitbucket.org/Florian_dauphine/mfi).

- **4th place**: Anders Wind Steffensen and Mikael Lindemann (IT University of Copenhagen). Their submission is written in Java and solved 13 out of 100 instances.

- **5th place**: Kaustubh Joglekar, Akshay Kamble, and Rajesh Pandian (Indian Institute Of Technology, Madras). Their submission is written in C++ and solved 10 of 100 instances.

- **6th place**: Saket Saurabh and Prafullkumar Tale (Institute of Mathematical Sciences,
Chennai). Their submission is written in Python and solved 19 of 100 instances. For 28 instances the provided solution was suboptimal. For 13 instances, the submission was the only one to output any solution. The suboptimal solutions are usually quite close to the optimal solutions. Hence, this submission can be viewed as a good heuristic.

- **7th place:** Mani Ghahremani (University of Portsmouth). Their submission is written in Java and solved 5 of 100 instances. For one instance, the provided solution was suboptimal.

- **8th place:** Frederik Madsen, Mikkel Gaub, and Malthe Kirkbro (IT University of Copenhagen). Their submission is written in C++ and solved 3 of 100 instances. For 3 instances, the provided solution was suboptimal.

We remark that the hidden instances were chosen to be especially challenging, hence the organizers expected that submissions would solve only a minority of the instances. Thus, solving more than 50% of the instances seems to be a remarkable contribution. All three winning contributions were faster than our naïve ILP formulation. An implementation we obtained of a more involved ILP formulation [3] was proven incorrect by the submissions; it remains to evaluate whether the error is due to the formulation itself or the scripts that generate the ILP from the graph data.

## 4. PACE organization

In September 2017, Frances Rosamond transferred the Steering Committee Chair to Bart Jansen. The Steering Committee and the PACE 2017 Program Committee are as follows.

*Steering committee:*

<table>
<thead>
<tr>
<th>Name</th>
<th>Institution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Holger Dell</td>
<td>Saarland University &amp; Cluster of Excellence</td>
</tr>
<tr>
<td>Thore Husfeldt</td>
<td>ITU Copenhagen &amp; Lund University</td>
</tr>
<tr>
<td>Bart M. P. Jansen (chair)</td>
<td>Eindhoven University of Technology</td>
</tr>
<tr>
<td>Petteri Kaski</td>
<td>Aalto University</td>
</tr>
<tr>
<td>Christian Komusiewicz</td>
<td>Friedrich-Schiller-University Jena</td>
</tr>
<tr>
<td>Frances A. Rosamond</td>
<td>University of Bergen</td>
</tr>
</tbody>
</table>

*Track A:*

<table>
<thead>
<tr>
<th>Name</th>
<th>Institution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Holger Dell</td>
<td>Saarland University &amp; Cluster of Excellence</td>
</tr>
<tr>
<td>Christian Komusiewicz</td>
<td>Friedrich-Schiller-University Jena</td>
</tr>
</tbody>
</table>

*Track B:*

<table>
<thead>
<tr>
<th>Name</th>
<th>Institution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nimrod Talmon</td>
<td>Weizmann Institute of Science</td>
</tr>
<tr>
<td>Mathias Weller</td>
<td>Laboratory of Informatics, Robotics, and Microelectronics of Montpellier (LIRMM)</td>
</tr>
</tbody>
</table>

The Program Committee of PACE 2018 will be chaired by Édouard Bonnet (Middlesex University, London) and Florian Sikora (Université Paris Dauphine).

## 5. Conclusion

As organizers, we consider the second 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 2017 participants showed an interest in the competition. Tamaki [31] won a best paper award at ESA 2017 for ideas that led to his three PACE 2017 submissions, and his paper was directly inspired by PACE.

We thank all the participants for their enthusiasm 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 [29] to receive PACE updates and join the discussion. In particular, plans for PACE 2018 will be posted there.

**Acknowledgments.** We are grateful to Szymon Wasik for the fruitful collaboration and for hosting the competition at optil.io. Prize money and travel grants were given through the generosity of Networks [28], an NWO Gravitation project of the University of Amsterdam, Eindhoven University of Technology, Leiden University, and the Center for Mathematics and Computer Science (CWI). We thank Ben Strasser for suggesting Figure 3.

**References**


27 Network repository. URL: [http://networkrepository.com/](http://networkrepository.com/).


