PACE Solver Description: GraPA-JAVA*

Moritz Bergenthal
Universität Bremen, Germany

Jona Dirks
Universität Bremen, Germany

Thorben Freese
Universität Bremen, Germany

Jakob Gahde
Universität Bremen, Germany

Enna Gerhard
Universität Bremen, Germany

Mario Grobler
Universität Bremen, Germany

Sebastian Siebertz
Universität Bremen, Germany

Abstract

We present an exact solver for the Directed Feedback Vertex Set Problem (DFVS), submitted for the exact track of the Parameterized Algorithms and Computational Experiments challenge (PACE) in 2022. The solver heavily relies on data reduction (known from the literature and new reduction rules). The instances are then further processed by integer linear programming approaches. We implemented the algorithm in the scope of a student project at the University of Bremen.

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

Keywords and phrases complexity theory, parameterized complexity, linear programming, java, directed feedback vertex set, PACE 2022

Digital Object Identifier 10.4230/LIPIcs.IPEC.2022.30

Supplementary Material

Software (Source Code Release): `https://doi.org/10.5281/zenodo.6647003`
Software (Public Git Repository): `https://gitlab.informatik.uni-bremen.de/grapa/java/fptg-library`, archived at `swh:1:dir:fd00e212eda2d4eab270dadc83923224551db839`
Software (Public Git Repository): `https://gitlab.informatik.uni-bremen.de/grapa/java/maxcliqueenumeration`, archived at `swh:1:dir:a4cde68f2e0de0e5b873eb5a02dc975c4a37fbb1`
Software (Public Git Repository): `https://gitlab.informatik.uni-bremen.de/grapa/java/pace-2022-dfvs-solver`, archived at `swh:1:dir:a80285858dd46e917d4f0c89fd13948177c6a048`

Preliminaries

We use standard notation for directed graphs as in [2], with $u, v \in V(G)$ being two vertices, and $(u, v) \in E(G)$ being an edge from vertex u to vertex v. We use the term *directed edge* for any edge $(u, v) \in E(G)$ if $(v, u) \notin E(G)$ and *undirected edge* (or 2-cycle) otherwise.

1 Solver overview

On a high level, our algorithm works as follows. It uses three steps to find a minimum DFVS for an input graph G:

1. Parse the input and apply the most basic data reduction rules.
2. Apply advanced data reduction rules iteratively (details are presented in Section 2).
3. Solve the remaining instance, possibly iteratively (details are presented in Section 3).

* This is a brief description of one of the highest ranked solvers of PACE Challenge 2022. It has been made public for the benefit of the community and was selected based on the ranking. PACE encourages publication of work building on the ideas presented in this description in peer-reviewed venues.

Table 1 Data reduction rules.

	Name	Note	Source
1.	Delete Loops	Implicitly applied	Folklore, rule 1 of [6]
2.	Remove connected to $n-1$		Special case of rule 5
3.	In-/out-degree 0	Included in CRR	Rule 3 of [6], rule 5 of [1]
4.	In-/out-degree 1	Included in CRR	Rule 4 of [6], rule 6 of [1]
5.	Dominating 2-cycle	Superset of both predecessors and successors	Adapted from rule 6 of [4]
6.	Strongly connected components	Previous rules inbetween	One step of [5] at a time
7.	Delete unnecessary edges	Partially included in CRR	Special case of rule 10
8.	Contract isolated paths of length 3 (lines)	Possible when not creating new induced cycles	Adapted from rule 7 of [4]
9.	Contract degree three	Possible when not creating new induced cycles	Adapted from special case of rule 8 of [4]
10.	Delete non-induced edges		New
	We only need to consider induced edges. For every edge, we start a BFS and do not visit a vertex if a cycle is closed. For this, we track disallowed predecessors that get reduced if an alternative path exists. It is not exhaustive, as there might be cases where two alternative paths would be closed.		
11.	Pick dominating vertices on 2-cycles		New
	If a vertex dominates all predecessors/successors of the other vertex on an undirected edge with undirected edges, we can include it in the solution.		
12.	Crown reduction	Special case of connected to all in foot	Adapted from rule 3 of [4]
13.	Three hitting set	Only applies in Section 3.3	New
	If all cycles running through a vertex are at most 3-cycles, we can replace them with a hitting set gadget. If all vertices of the internal clique are selected, we are allowed to push one outwards, hitting the cycle		
14.	Any hitting set	Only applies in Section 3.3	New
	Applied to a cycle that is isolated except for at most one edge, in that case create a gadget as in rule 13		

2 Data reduction rules

We apply the data reduction rules presented in Table 1. The rules are applied iteratively. Each rule is applied exhaustively. After a successful rule application, we return to the first rule.

We apply commonly known rules for DFVS (Rules 1, 3, 4, 6). The most local rules are applied recursively in a rule that we call Combined Recursive Reduction (CRR). As DFVS is a generalization of the Vertex Cover problem, we were able to adapt several rules designed for Vertex Cover (Rules 2, 5, 8, 9, 12). Additionally, we have found several rules generalazing known rules from the literature, as well as several completely new rules not known from the literature. Many of these rules rely on the observation that we only need to hit induced cycles when solving a DFVS instance, as all other cycles will be hit in that case as well. The correctness of all new rules will be presented in a companion paper.

3 Exact solving

After applying the data reduction rules, depending on the relative number of undirected edges, we employ different solving strategies. If less than half of the edges are undirected edges, we immediately resort to the iterative addition of cycles as a constraint for a hitting set ILP formulation, explained in Section 3.1.

In mixed or largely undirected graphs, we resort to an ILP formulation that models the problem as finding a topological order (Section 3.2) with additional hints to improve the internal lower bounds of the solver. ILPs are solved with SCIP. If the ILP solver does not terminate within twelve minutes, the attempt will be terminated. Instead, we compute an exact solution to the Vertex Cover problem. If this is not a solution to DFVS, we do not return a solution. Details are presented in Section 3.3.

3.1 Iterative cycle hitting set ILP

Over time, we generate a set $\mathcal{K}$ of induced cycles. We initialize $\mathcal{K}$ with all induced cycles of length 2, 3 and 4, and add longer disjoint cycles that are packed greedily until no further disjoint cycles remain. We then solve the ILP (Algorithm 1) and interpret the chosen variables as vertices to remove from the graph. If no cycle remains, we have obtained the optimal solution, otherwise, we greedily compute a new cycle packing on the remainder of the graph and add all these cycles to $\mathcal{K}$.

Algorithm 1 Hitting set ILP formulation.

foreach $v_i \in V(G)$ **add** variable $x(v_i) = x_i \in X$ with constraint $x_i \in [0, 1]$
foreach *cycle* $K \in \mathcal{K}$ **add** constraint $\sum_{v_i \in K} x(v_i) \geq 1$
minimize $\sum_{x_i \in X} x_i$

3.2 Topological order ILP

A graph is acyclic if and only if its vertices can be ordered as $v_1, \ldots, v_n$ such that every edge (v_i, v_j) satisfies $i < j$. We use this observation for the following ILP formulation (Algorithm 2). For the undirected subcomponents of the graph, we compute a set of maximum cliques $\mathcal{C}$ using [3]. This external solver does not always find all 2-cliques, so we compute all 2-cycles and remove them if they are somewhere included in a clique. We create a partial order on all vertices adjacent to directed edges, denoted as $\mathcal{E}$. The constraints can be interpreted as $o(s) < o(t) \vee x(s) \vee x(t)$: edges must be part of a DAG over the order or either vertex chosen for the solution. We furthermore compute a hint of short cycles $\mathcal{K}$ as in Section 3.1. The vertices chosen can directly be interpreted as a result for DFVS.

3.3 Underlying Vertex Cover

We take the undirected part of the graph and compute a minimum vertex cover using [7]. If no cycles are remaining, we return the solution. Otherwise, we apply the hitting set gadget reduction rules (Rules 13 and 14) and solve the resulting instance again. At this point, we could turn towards iterative solving as in Section 3.1 but did not implement this as the above approach was already sufficient on all of the test instances.

Algorithm 2 Linear ordered ILP formulation.

foreach $v_i \in V(G)$ **add**
  variable $x(v_i) = x_i \in X$ with constraint $x_i \in [0, 1]$
  variable $o(v_i) = y_i \in Y$ with constraint $y_i \in [0, n]$
foreach *clique* $C \in \mathcal{C}$ **add** constraint $\sum_{v_i \in C} x(v_i) \geq |C| - 1$
foreach *edge* $(s, t) \in \mathcal{E}$ **add** constraint $o(t) - o(s) + x(t) \cdot (n + 1) + x(s) \cdot (n + 1) \geq 1$
foreach *cycle* $K \in \mathcal{K}$ **add** constraint $\sum_{v_i \in K} x(v_i) \geq 1$
minimize $\sum_{x_i \in X} x_i$

References

1 Benjamin Bergougnoux, Eduard Eiben, Robert Ganian, Sebastian Ordyniak, and M. S. Ramanujan. Towards a Polynomial Kernel for Directed Feedback Vertex Set. In Kim G. Larsen, Hans L. Bodlaender, and Jean-Francois Raskin, editors, *42nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2017)*, volume 83 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 36:1–36:15, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.MFCS.2017.36.

2 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2016. doi:10.1007/978-3-319-21275-3.

3 David Eppstein, Maarten Löffler, and Darren Strash. Listing all maximal cliques in large sparse real-world graphs. *ACM J. Exp. Algorithmics*, 18, November 2013. doi:10.1145/2543629.

4 Michael R. Fellows, Lars Jaffke, Aliz Izabella Király, Frances A. Rosamond, and Mathias Weller. What is known about vertex cover kernelization? In *Adventures Between Lower Bounds and Higher Altitudes*, pages 330–356, 2018. doi:10.1007/978-3-319-98355-4_19.

5 Lisa K. Fleischer, Bruce Hendrickson, and Ali Pınar. On identifying strongly connected components in parallel. In José Rolim, editor, *Parallel and Distributed Processing*, pages 505–511, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. doi:10.1007/3-540-45591-4_68.

6 Rudolf Fleischer, Xi Wu, and Liwei Yuan. Experimental study of fpt algorithms for the directed feedback vertex set problem. In Amos Fiat and Peter Sanders, editors, *Algorithms – ESA 2009*, pages 611–622, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. doi:10.1007/978-3-642-04128-0_55.

7 Demian Hespe, Sebastian Lamm, Christian Schulz, and Darren Strash. WeGotYouCovered: The winning solver from the pace 2019 implementation challenge, vertex cover track. *ArXiv*, abs/1908.06795, 2019. arXiv:1908.06795.