PACE Solver Description: DUM Solver

Henri Dickel
Philipps-Universität Marburg
Dickel@students.uni-marburg.de

Lennart Uhrmacher
Philipps-Universität Marburg
Uhrmachl@students.uni-marburg.de

Matija Miskovic
Philipps-Universität Marburg
Miskovic@students.uni-marburg.de

Abstract

In the Directed Feedback Vertex Set (DFVS) problem the objective is to find a minimal set of vertices that intersects all directed cycles in a given digraph. In this paper we present an ILP based solver.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms

Keywords and phrases DFVS, ILP, PACE

Supplement Material `https://github.com/HenriDickel/DFVS-Solver/tree/PACE`
Source Code DOI: 10.5281/zenodo.6599645

1 Introduction

The DFVS problem is a NP-hard graph problem with the goal of finding a minimal subset of vertices $V' \subseteq V$ of a digraph $G = (V, A)$ such that deleting all these vertices and their arcs makes the graph acyclic. This problem has applications like deadlock detection[1] and program verification, therefore there is practical value in finding better solvers and techniques related to it.

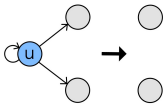
In this paper, we describe an exact solver written in Java, which was initially developed for the lecture "Algorithm Engineering" at Philipps-Universität Marburg and further improved upon for the PACE 2022 challenge. First, we go through the various principles and rules used in order to reduce the input graph. The goal of these rules is to decrease the number of vertices and arcs that need to be examined in the actual solving algorithm which leads to a decrease in time needed to solve them. After that, we move on to the solving process of the graphs, for which we ended up implementing integer linear programming (ILP) using the open-source framework SCIP[1]. We then examine our results and reflect upon where our implementation is lacking and where further improvements could be possible.

2 Reduction Rules

Our solver uses reduction rules to simplify the input graph. This can be done by removing nodes from the graph, but also by adding nodes to the solution set S. In the following we show the reduction rules we use and explain them briefly in both figures and descriptions. The first three rules are based on a paper by Rudolf Fleischer, Xi Wu and Liwei Yuan[2]. Note that gray nodes can have additional arcs.

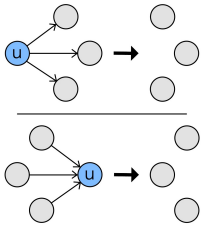
[1] `https://scipopt.org`

Self-edge rule



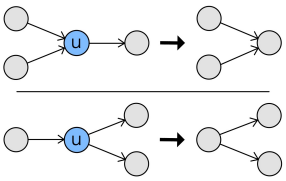
Since u has an arc to itself, a cycle exists which only contains u. Therefore, u can be removed from the graph and added to S.

Trivial vertex rule



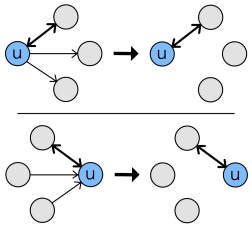
When u has either only outgoing or only ingoing arcs, u can not be part of a cycle and can be removed from the graph.

Chaining rule



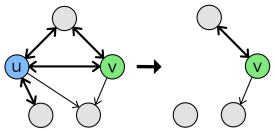
If some vertex u has only one outgoing edge, then remove u and add an edge from all of u's in-neighbors to the single out-neighbor. Similarly, if some vertex u has only one ingoing edge, then remove u and add edges from the single in-neighbor to all of u's out-neighbors. This can be done, since a cycle containing u will also contain the single neighbor and can be destroyed by removing him instead of u.

Bidirectional edge trivial vertex rule



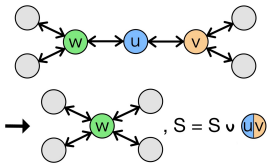
Single edges from u can be removed, when u has either only outgoing or only ingoing arcs (besides bidirectional edges). This can be done, since the bidirectional edges from u have to be destroyed anyway and thus there are no arcs left to form cycles with the remaining single-arcs.

Subset rule



Looking at two double-connected neighbors u and v, if the arcs of v are a subset of the arcs of u, u can be removed from the graph and added to S. This can be done since either u or v have to be destroyed anyway due to the bidirectional edge, and destroying u is more beneficial for the remaining graph.

Bidirectional edge chaining rule



This rule only applies, when u, v and w only have bidirectional edges. When u has exactly two neighbors v and w, remove u and v and add the remaining arcs from v to w. Add one node to S, which is not yet determined. When the graph is solved, check if w is in S. If yes, the undetermined node is v, if not, it is u. Due to the structure of u, v and w, an optimal Vertex Cover can either contain $\{u\}$ or $\{v, w\}$.

3 ILP

For the actual solving part we decided to implement integer linear programming (ILP) using SCIP as it produced the greatest number of solved graphs in the given time restrictions of the PACE challenge. Before that we mostly used a branch-and-bound solver. Unfortunately, after a lot of work and optimization on this branch-and-bound solver, it solved only 56 out of the 100 graphs in the public dataset. This led us to look into different methods of solving the problem and in the end we went with an ILP approach, which solved around 66 instances. Due to the rules of the PACE competition, we used the open-source framework SCIP as it is well-known and has a Java interface which was relatively easy to use.

In the ILP algorithm itself we differentiate between two types of graphs, one where all edges are bidirectional and one where this is not the case. For cases where all vertices of a graph are connected with bidirectional edges, the problem becomes equivalent to computing a minumum vertex cover. In this case we do the following:

- For each node u we add a binary variable x_u, which determines whether u is to be deleted or not.
- For every edge (u, v) in the set of bidirectional edges E we add a constraint that one of the nodes has to be deleted: $x_u + x_v \geq 1 \quad \forall (u, v) \in E$

This approach does not work if the graph contains any non-bidirectional edge and therefore a different method is required. In our case we use variables and constraints to determine if it is possible to create a topological ordering by deleting certain nodes. If a graph has a topological ordering, it is also acyclic and therefore that is a valid solution for our problem. This is achieved by adding two variables for each vertex u, a binary variable x_u which again just determines if a node should be deleted or not, and an integer variable y_u $(1 \leq y_u \leq n)$ which represents the position of the node in the topological ordering. Let $A' \subseteq A$ be the set of arcs (u, v) where no arc (v, u) exists. For every arc (u, v) in A', we add a constraint which pushes arcs with undeleted endpoints forward in the ordering.

$$y_u - y_v + n \cdot x_u \geq 1 \quad \forall (u, v) \in A'$$

Both of these approaches are further improved by adding additional cycle covering constraints. This means that for every node a cycle is found, to which it belongs to, and based upon this cycle a constraint is made which dictates that at least one node in that cycle needs to be deleted.

4 Conclusion

As described earlier, it was unfortunate that we had to rely on an ILP approach instead of using our own branch-and-bound approach. There are further improvements possible but this was the combination that was able to produce the best results in the tests that we were able to conduct.

References

1 Jianer Chen, Yang Liu, Songjian Lu, Barry O'sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *J. ACM*, 55(5), nov 2008. doi:10.1145/1411509.1411511.

2 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, 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*, volume 5757 of *Lecture Notes in Computer Science*, pages 611–622. Springer, 2009. doi:10.1007/978-3-642-04128-0_55.