PACE Solver Description: Finding optimal feedback vertex sets of directed graphs using DiVerSeS*

Sylwester Swat
Institute of Computing Science, Poznań University of Technology, Poland
sylwester.swat@put.poznan.pl

Abstract

This article briefly describes the most important algorithms and techniques used in the exact Directed Feedback Vertex Set solver "DiVerSeS", submitted to the 7th Parameterized Algorithms and Computational Experiments Challenge (PACE 2022).

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Directed feedback vertex set, exact solver, graph algorithms, PACE 2022

Category PACE Solver Description

1 Problem description

In the Directed Feedback Vertex Set (DFVS) Problem we are given a directed graph $G = (V, A)$, and the goal is to find a smallest possible subset $X \subset V$ of nodes whose removal leaves the graph acyclic. The problem of finding an optimum directed feedback vertex set of a graph is equivalent to finding an optimum hitting set of a family containing all cycles in graph G.

2 Solver description

In this paper we provide a short description of the most important algorithms implemented in solver DiVerSeS. Due to space limitation, this description may not contain full information about the behavior of used algorithms in all possible situations. The general scheme of algorithms used in the exact solver DiVerSeS can be described in the following way:

1. Reduce the graph using extensive application of data reduction rules.
2. Find a heuristic solution using the heuristic version of DiVerSeS solver.
3. Run an exact algorithm.
4. Lift the solution to create a final DFVS of the original graph.

Before we proceed to further description, let us introduce some notations. By $A(G)$ we denote the set of arcs of a directed graph G. An arc $(a, b) \in A$ is called a *pi-arc* if there is also an arc $(b, a) \in A$. An *underlying pi-graph*, or simply a *pi-graph* of a graph G is a graph $pi(G) = (V, A_{pi})$, where $A_{pi} = \{(a, b) \in A : (a, b) \text{ is a pi-arc}\}$. A pi-graph $pi(G)$ will also be denoted simply by piG. Since all arcs in piG have a directed counterpart, piG can in some sense be treated as an undirected graph. A *nonpi-graph* of a graph G is a graph $npi(G) = (V, A_{npi})$, where $A_{npi} = \{(a, b) \in A : (a, b) \notin A(piG)\}$. A *superpi-graph* of a graph G is a graph $spi(G) = (V, A_{spi})$, where $A_{spi} = \{(a, b) : (a, b) \in A \text{ or } (b, a) \in A\}$. Node v is called a *pi-node* if all arcs incident to it are pi-arcs.

* This is a brief description of DiVerSeS - an exact directed feedback vertex set solver, submitted to PACE Challenge 2022.

3 Preprocessing

We use a variety of data reduction rules. We implemented majority of data reduction rules known to us from the literature, proposed modifications of some existing methods and created many novel methods to reduce the graph size or to modify the graph structure in some other, specific way. In the initial preprocessing phase we run all implemented data reduction rules to obtain, hopefully, a smaller 'kernel' for the problem.

To the most well known data reduction rules we can include those from [3] and [4]. Here we list the most important ones from them:

1. LOOP rule: add to DFVS a node that contains a self-loop and remove it from the graph.
2. IN0, OUT0 rules: removing nodes with empty in-neighborhood $N^-(v)$ or empty out-neighborhood $N^+(v)$.
3. IN1, OUT1 rules: merging each node v with $|N^-(v)| = 1$ or $|N^+(v)| = 1$ (by merging node v we mean adding to the graph, unless already present, all possible arcs from the set $N^-(v) \times N^+(v)$, then removing v from the graph).
4. PIE rule: removing all arcs (a, b) such that a and b belong to different strongly connected components in graph $npiG$.
5. DOME rule: removing from the graph all dominated arcs (see [4] for more details).

These are the most basic data reduction rules, but also the most often used ones in practice. It is worth mentioning that there are many rules that are 'dominated' by other reductions, but are still useful, due to the fact that they can simply be implemented with smaller constant factor overhead and are faster in practice. As an example let us consider the IN0, OUT0 and PIE rules. Using the PIE rule will remove from the graph all nodes with empty in-neighborhood or out-neighborhood, but it requires determining the set of strongly connected components of given graph, which is usually much slower than removing the nodes in a straightforward way.

4 Main algorithm

After obtaining a reduced graph in the preprocessing phase, we run the main algorithm. There are two different approaches implemented and used in DiVerSeS: a branching approach and an iterative hitting set approach. A branching approach is used when the fraction $\frac{|A(piG)|}{|A|}$ is large (specified by a parameter, by default set to 0.3). Otherwise the iterative hitting set approach is used. In both methods we use an upper bound on the solution size, which is found using a heuristic version of DiVerSeS.

4.1 Branching approach

In this method we use the branch-and-reduce-and-bound technique. For a given graph we first run some data reduction rules - we selected only the fastest ones to reduce time spent on reductions after each branching step. After performing reductions we find a vertex cover C of a pi-graph piG. The size of found vertex cover is clearly a lower bound on the size of an optimum DFVS of G. We find C using WeGotYouCovered - the winning solver from PACE 2019 [2]. If C is a DFVS of G, then it is an optimum one. If the size of currently constructed DFVS (including the nodes that will be added in solution-lifting) exceeds or equals an upper bound, we prune currently checked branch of the search tree. Otherwise we need to branch further. To find a branching node we first create a graph $G[V \setminus C]$, then reduce the graph using LOOP, IN0, OUT0, IN1 and OUT1 rules. If a LOOP rule was applied to some subset $D \subset V$, then as a branching node v we select the one from D that maximizes value $|N^-(v)| \cdot |N^+(v)|$. Otherwise we select the node not from D but from V instead. Finally, we can run the algorithm recursively. In the first case we remove v from the graph, in the second case we merge node v. Let us note here that merging node v does not increase the size of constructed DFVS, but it reduces the graph size and makes the graph denser, thus increasing efficiency of reduction rules and vertex-cover-bound in further search steps.

4.2 Iterative hitting set approach

For graphs with small value of fraction $\frac{|A(piG)|}{|A|}$ the branching approach is very inefficient, because the vertex-cover-bound is not very useful unless the graph contains a large number of pi-arcs. In that case we use a different approach. First we generate a set H of all induced cycles of length at most L (initially L is set to 3) in graph G. Then we find a minimum-size hitting set hs of H using a solver from [1]. If hs is a DFVS of G, then we found an optimal DFVS. Otherwise we find all induced cycles of length L in graph $G[V \setminus hs]$. If no induced cycle was found, we keep increasing L by 1, until we find a cycle of length L in $G[V \setminus hs]$. Now we add found cycles to H and try to find a hitting set of H of size $|hs|$ using a local search solver. If a hitting set of that size was not found, then we simply repeat the whole procedure. If, however, a hitting set of size $|hs|$ was found, then it is optimal and we can repeat the whole procedure, but skip using an exact hitting set solver (which is the most time-consuming part) in next iteration.

5 Availability

The source code of DiVerSeS is freely available and can be found at
`https://zenodo.org/record/6643144#.YqjL2r9ByV4`.

References

1 Thomas Bläsius, Tobias Friedrich, David Stangl, and Christopher Weyand. *An Efficient Branch-and-Bound Solver for Hitting Set*, pages 209–220. doi:10.1137/1.9781611977042.17.

2 Demian Hespe, Sebastian Lamm, Christian Schulz, and Darren Strash. Wegotyoucovered: The winning solver from the pace 2019 challenge, vertex cover track. In *2020 Proceedings of the SIAM Workshop on Combinatorial Scientific Computing*, pages 1–11. SIAM, 2020.

3 Hanoch Levy and David W Low. A contraction algorithm for finding small cycle cutsets. *Journal of Algorithms*, 9(4):470–493, 1988. doi:https://doi.org/10.1016/0196-6774(88)90013-2.

4 Hen-Ming Lin and Jing-Yang Jou. On computing the minimum feedback vertex set of a directed graph by contraction operations. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(3):295–307, 2000. doi:10.1109/43.833199.