PACE Solver Description: DiVerSeS – A Heuristic Solver for the Directed Feedback Vertex Set Problem[*]

Sylwester Swat
Institute of Computing Science, Poznań University of Technology, Poland

Abstract

This article briefly describes the most important algorithms and techniques used in the directed feedback vertex set heuristic solver called "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, heuristic solver, graph algorithms, PACE 2022

Digital Object Identifier 10.4230/LIPIcs.IPEC.2022.27

Supplementary Material *Software (Source Code)*:
`https://zenodo.org/record/6643144#.YqjL2r9ByV4`

Funding Supported by the Foundation for Polish Science (FNP).

1 Problem description

A feedback vertex set of a directed graph $G = (V, A)$ is a set $X \subset V$ such that the induced graph $G[V \setminus X]$ is acyclic. The solver briefly described here is a heuristic approach to the Directed Feedback Vertex Set (DFVS) problem, where the goal is to find a smallest possible feedback vertex set of a given directed graph. The DFVS problem can be also considered equivalently as a Maximum Directed Acyclic Subgraph problem, where for a given directed graph $G = (V, A)$ the task is to find a largest possible set $Y \subset V$ such that $G[Y]$ is a DAG.

2 Solver description

In this paper we provide a short description of the most important algorithms implemented in solver DiVerSeS. Due to a large variety of used methods, this description does not contain full information about used algorithms and their behaviour in many distinct situations. The workflow of DiVerSeS can be described in the following general steps:

1. Reduce the graph using data reduction rules.
2. Find some initial solution of a reduced graph using fast heuristics.
3. Improve found solution using a variety of heuristic approaches.
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 $(b, a) \in A$. 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 *nonpi-graph* of a graph G is a graph

[*] 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.

$npi(G) = (V, A_{npi})$, where $A_{npi} = \{(a,b) \in A : (a,b) \notin A(pi(G))\}$. 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.

3 Preprocessing

We use a large number of different data reduction rules. This includes an implementation of almost all data reduction rules known to us from the literature, modifications of some existing methods and a whole collection of new methods used to reduce the graph size or to modify the graph structure in some specific way (e.g. by adding some arcs, what might be seen as a little bit contradictory to the intuitive comprehension of a term 'data reduction'), from which some constitute a generalization of known data-reduction rules for the vertex-cover problem. To the most well known data reduction rules we can include those from [4] and [5]. These include the following:

1. IN0, OUT0 rules: removing nodes with empty in-neighborhood $N^-(v)$ or empty out-neighborhood $N^+(v)$.
2. 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).
3. PIE rule: removing all arcs (a,b) such that a and b belong to different strongly connected components in graph $npi(G)$.
4. DOME rule: removing from the graph all dominated arcs (see [5] for more details).
5. CORE rule: removing from the graph (and adding to constructed DFVS) all neighbors of a pi-node v whose neighborhood is a clique in $pi(G)$.

These are the most basic (but still very effective in practice) among rules implemented in DiVerSeS. Nevertheless, proper implementation (with a guarantee of best worst-case performance, but also minimizing a constant overhead factor) of some of those rules is not trivial. For example, authors of [5] claim that the CORE rule can be implemented in time $O(|A| + |V| \cdot log|V|)$. We believe that arguments presented by the authors are either not correct or there is no proper explanation, and we were unable to implement the CORE algorithm with such time complexity (our implementation of the CORE rule works in time $O(|A|^{\frac{3}{2}})$). On the other hand the DOME rule can be implemented in time $O(|A|^{\frac{3}{2}})$ instead of $O(|A| \cdot |V|)$ proposed by the authors, what is a significant improvement for large sparse graphs that contain nodes with high degree.

4 Creating initial solution

There are many algorithms used to create an initial solution of a given graph G and are used depending on the graph characteristics. For example, if the fraction $\frac{|A(pi(G))|}{|A|}$ is high (specified by a parameter), then as a DFVS of G we simply take the vertex cover of $spi(G)$. To find a vertex cover we use NuMVC [2] and FastVC [1] algorithms. If the fraction of pi-arcs is not high, then we use other methods. One of them is the *agent-flow* algorithm that works in the following way:

1. Assign a fixed number of tokens to each node.
2. In each of R (some small fixed integer) iterations, for each node and each token assigned to that node, move the token to some out-neighbor of a given node.
3. Add to constructed DFVS a node that contains most tokens after all R iterations and remove that node from the graph.
4. Repeat the procedure until the obtained graph is acyclic.

There are two variations of the agent-flow method: discrete and continuous. In the discrete version we have an integral number of tokens in each node and in each iteration we separately move each token to a random out-neighbor. In a continuous version we have a real-valued number of tokens and in each iteration we distribute the tokens evenly among all out-neighbors.

After a DFVS X of graph G is found, we remove redundant nodes from X (a node $x \in X$ is redundant if $X \setminus \{x\}$ is also a DFVS) using a very efficient algorithm making use of properties of dynamically changing topological order of an induced graph $G[V \setminus X]$.

5 Improvement of solution

When an initial solution is found, we try to improve it using various approaches. Which algorithm is used to improve the solution depends on some characteristics of graph G. For example, if G is a very sparse graph (but not too sparse), we use an efficient improvement of a simulated-annealing-based algorithm (description of the basic algorithm can be found in [3]). If the graph contains a high percentage of pi-arcs, then we use (among others) the following approach:

1. Find an ordering $(v_1, \ldots, v_N)$ of V with as few backgoing arcs as possible (an arc $(a, b) \in A$ is a backgoing arc if b precedes a in the ordering).
2. Take as a DFVS a vertex cover of a graph $H = (V, A_H)$, where A_H is the set containing all backgoing arcs.

There are a few ways implemented in DiVerSeS to create an ordering and they usually take into account structure of current DFVS. This way we can improve existing solution instead of just finding another one. It is also worth mentioning here that finding an optimal ordering (for which the number of backgoing arcs is minimum) is equivalent to finding an optimal solution of an instance of the Directed Feedback Arc Set Problem, which is NP-complete (and in certain sense equivalent to the DFVS problem).

6 Availability

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

References

1 Shaowei Cai, Jinkun Lin, and Chuan Luo. Finding a small vertex cover in massive sparse graphs: Construct, local search, and preprocess. *Journal of Artificial Intelligence Research*, 59:463–494, July 2017. doi:10.1613/jair.5443.

2 Shaowei Cai, Kaile Su, Chuan Luo, and Abdul Sattar. Numvc: An efficient local search algorithm for minimum vertex cover. *Journal of Artificial Intelligence Research*, 46, February 2014. doi:10.1613/jair.3907.

3 Philippe Galinier, Eunice Lemamou, and Mohamed Bouzidi. Applying local search to the feedback vertex set problem. *Journal of Heuristics*, 19, October 2013. doi:10.1007/s10732-013-9224-z.

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

5 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.