DireFeVer: A solver for the directed feedback vertex set propblem

Timon Behr ✉
University of Konstanz, Germany

Abstract

This document briefly describes our implementation to solve the directed feedback vertex set problem. We give an exhaustive list of all the used techniques and explain how they are put together.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases directed feedback vertex set, algorithms, solver

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

1 Preliminaries

In this Chapter we give a few important definitions.

A *self-loop* is a directed edge that connects a node to itself.
Given a directed graph $G(V, E)$ a *strongly connected component* is a subgraph $G'(V', E')$ of G, such that for each pair of nodes $v, w \in V'$ with $v \neq w$ there exists a directed path from v to w.
We call a node subset $V' \subseteq V$ a *clique* if for all $v, w \in V'$ with $v \neq w$, $(v, w) \in E$ and $(w, v) \in E$.
A *cycle of size two* is any edge pair $(v, w) \in E$ and $(w, v) \in E$.
For convenience, we abbreviate directed feedback vertex set with *DFVS*.

2 Techniques

In this Section we provide an overview over all the techniques that we used in our solver.

2.1 Instance reduction

Given a DFVS instance $I(G)$ we use reduction rules that solve certain substructures of G to receive a hopefully smaller instance $I'(G')$, such that an optimal solution of I' is also an optimal solution for I'. Following, we give a complete list of the reduction rules that we used in our solver.

- A collection of simple rules as described by [8]. These include:
  - the removal of nodes with an in- or outdegree of 0,
  - the contraction of nodes with an in- or outdegree of 1, and
  - adding nodes with a self-loop to the solution.
- A strongly connected component rule that removes directed edges between inclusion-wise maximal strongly connected components.
  This rule is extended, so that for a graph $G(V, E)$ we also remove directed edges between maximal strongly connected component of the graph $G'(V, E')$ with $E' = \{(v, w) | (v, w) \in E \vee (w, v) \notin E\}$. This procedure was proposed by [9].
- The *core* rule. This rule adds cliques also called *cores* that satisfy the following properties to the solution: A clique C is the *core* of cliques $C \in \mathcal{C}$ if $\forall C' \in \mathcal{C}$ $C \cup C'$ forms a clique. C can be added to the solution if and only if there exists a clique $C \neq C'' \in \mathcal{C}$ such that

42nd Conference on Very Important Topics (CVIT 2016).
Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:1–23:4
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

C'' has either only incoming or outgoing neighbors that are not in cliques of $\mathcal{C}$.
Our *core* rule is a generalized version of the *core*-rule described by [9]. Their rule works for a clique C that contains a node $q \in C$ that has only incoming or outgoing neighbors not in C.

- A *dome* rule as described by [9], that removes dominated edges.
- Some vertex cover rules that can be applied on subgraphs where most edges also contain their reciprocal counterpart:
  - The *link-node* rule as described by [5]. This rule is extended for the application on the directed feedback vertex set problem. Namely, nodes that have exactly two neighbors to which they are strongly connected and no other neighbors, are contracted, if and only if no simple path exists from one neighbor to the other.
  - The *twin* rule as described by [10].
  - The *unconfined* rule as described by [2].
  - The *crown* rule as described by [1].

2.2 Vertex Cover

Given an instance of the DFVS problem $I(G(V,E))$, if for each edge $(v,w) \in E$ there also exists $(w,v) \in E$ the optimal solution for $I(G)$ is identical to the optimal solution if a vertex cover instance $I'(G'(V',E'))$ where $V' = V$ and $E' = \bigcup_{(v,w)\in E}\{v,w\}$.
Thus, we can convert $I(G)$ to a vertex cover instance $I'(G')$ by only regarding edges $(v,w) \in E$ which also have the reciprocal counterpart $(w,v) \in E$. If the found optimal solution S for $I'(G')$ is also a solution for $I(G)$, we know that S is an optimal solution for $I(G)$. To solve the vertex cover we used our implementation[1] that implements the following techniques:

- Kernelization rules that include:
  - Removal of isolated nodes and contracting nodes of degree 1.
  - Contraction of link nodes [5].
  - Reduction of cliques [3].
  - The *twin* rule [10].
  - The *unconfined* node rule [2].
  - A rule that uses network flows [1] and an improved variant [7].
  - The *alternative* rule [10].
- Heuristic and approximation algorithms that include:
  - A simple 2-approximation.
  - A lower bound heuristic that first removes cliques and then unbalanced edges.
  - An upper bound heuristic that removes the node with the highest degree, applies reduction rules and repeats until no more nodes remain.
- A branch-and-reduce algorithm to compute the optimal solution. Besides branching on the nodes with the highest degree, we include mirror branching as described by [6].

2.3 Heuristics

We implement different heuristics to compute lower- and upper bounds to prime an exact algorithm. These heuristics include:

[1] https://github.com/mndmnky/duck-and-cover (version 1.5.0)

- A upper- and a lower bound from the received vertex cover solution: If a found vertex cover solution S was not a solution for the current DFVS instance $I(G)$ we can still use the size of the vertex cover solution as a lower bound. Further, by removing all nodes in S from G, we can solve the leftover graph, using any other upper bound heuristic, to receive an upper bound.
- A lower bound heuristic that counts and removes small cycles.
- A lower bound heuristic that removes inclusion maximal cliques until no more clique remain. Let $\mathcal{C}$ be all those cliques, the sum $\sum_{C \in \mathcal{C}} |C| - 1$ is a lower bound. This heuristic was also described by [9].
- Upper bound heuristics that use different weight heuristics. Given those weights, the heuristics either always contracts the node with the lowest weight, or adds the node with the highest weight to the solution. We implemented two weight functions, one by [4] and one by [9], but we only use the latter.

2.4 Exact Algorithm

For the exact algorithm we use a branch, reduce and bound strategy that first applies all reduction rules exhaustively and then branches on the first available option in a given branch priority list. This procedure is repeated recursively on every branch until no more cycles remain, or the best current upper bound is equals or greater than the best current lower bound.

The branching options are the following, ordered from the highest to the lowest priority:

1. Branching on the biggest found clique C of size 3 or more, by adding all nodes in C to the solution, or contracting any one node in C while adding the rest to the solution.
2. Branching on a link node and its two neighbors for which the rule could not be applied.
3. Branching on the node that is part of the most cycles of size two, by adding that vertex, or adding all the neighbors to the solution.
4. Branching, by adding, or contracting the node with the highest chosen weight.

3 Pipeline

To solve a given DFVS instance we first apply all the reduction rules exhaustively, then we split the instance into maximal strongly connected components. On each new instance we compute the vertex cover for the respective subgraph and check whether we found an optimal solution. If not we run the exact algorithm primed with the best upper and lower bound.

References

1 Faisal N Abu-Khzam, Rebecca L Collins, Micheal R Fellows, Micheal A Langston, W Henry Suters, and Christopher T Symons. Kernelization algorithms for the vertex cover problem. 2017.

2 Takuya Akiba and Yoichi Iwata. Branch-and-reduce exponential/fpt algorithms in practice: A case study of vertex cover. *Theoretical Computer Science*, 609:211–225, 2016.

3 Sergiy Butenko, Panos Pardalos, Ivan Sergienko, Vladimir Shylo, and Petro Stetsyuk. Finding maximum independent sets in graphs arising from coding theory. In *Proceedings of the 2002 ACM symposium on Applied computing*, pages 542–546, 2002.

4 X Cai, J Huang, and G Jian. Search algorithm for computing minimum feedback vertex set of a directed graph. *Comput Eng*, 32(4):67–69, 2006.

5 Jianer Chen, Iyad A Kanj, and Weijia Jia. Vertex cover: further observations and further improvements. *Journal of Algorithms*, 41(2):280–301, 2001.

6 Fedor V Fomin, Fabrizio Grandoni, and Dieter Kratsch. A measure & conquer approach for the analysis of exact algorithms. *Journal of the ACM (JACM)*, 56(5):1–32, 2009.

7 Yoichi Iwata, Keigo Oka, and Yuichi Yoshida. Linear-time fpt algorithms via network flow. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 1749–1761. SIAM, 2014.

8 Hanoch Levy and David W Low. A contraction algorithm for finding small cycle cutsets. *Journal of algorithms*, 9(4):470–493, 1988.

9 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.

10 Mingyu Xiao and Hiroshi Nagamochi. Confining sets and avoiding bottleneck cases: A simple maximum independent set algorithm in degree-3 graphs. *Theoretical Computer Science*, 469:92–104, 2013.