PACE 2022 Exact Track: Hex

S.A. Tanja
Eindhoven University of Technology

Abstract

We attempt to create an efficient exact solver, called Hex, for the Directed Feedback Vertex Set problem for the PACE 2022 competition. Firstly, some simple reduction rules are used to shrink the input graph. If the reduced graph G' is *bidirected*, an open-source Vertex Cover solver can be used to find a solution, otherwise, we perform a *splitting reduction* to obtain an undirected graph G'_u and a remainder G'_d from G', for which its vertex cover plus a set of forced vertices is hopefully a solution for G'. If that fails, we construct an ILP for G'_d and G'_u and solve that instead. The constraints we generate are *shortcut edge cycle covers* of G'_d such that we can find a good upper bound for the ILP that is a solution for G'_d. As a last resort, we include more shortcut edge cycle covers until we have a solution.

2012 ACM Subject Classification Theory of computation → Fixed-Parameter Tractability

Keywords and phrases PACE 2022, Directed Feedback Vertex Set

Supplementary Material The source code is available on GitHub (`https://github.com/satanja/Hex`) and Zenodo (`doi:10.5281/zenodo.6609797`).

Acknowledgements I want to thank my supervisor Kevin Verbeek for his (ongoing) help and suggestions.

1 Preliminaries

Let $G = (V, E)$ be a simple directed graph, i.e., parallel edges and self-loops are forbidden. The Directed Feedback Vertex Set problem asks for a minimum-sized set $X \subseteq V$ such that after deleting X from G, the remainder is an acyclic graph. We say that an edge (u, v) is *bidirectional* if $(v, u) \in E$. We say that a directed graph is *bidirected* if for every $(u, v) \in E$ we have $(v, u) \in E$. On the other hand, G is *unidirected*, when G does not have any bidirectional edges.

If G is bidirected, then the Directed Feedback Vertex Set problem is equivalent to the Vertex Cover problem after transforming G to an undirected graph. Note that for every $(u, v) \in E$, we then require $u \in S$ or $v \in S$, since $(v, u) \in E$ as well.

We introduce some notation before proceeding. For clarity, we also denote the set of vertices and edges of a graph G using $V(G)$ and $E(G)$, respectively. Let $N^+(v)$ be the set of vertices for which v has an outgoing edge, and let $N^-(v)$ be the set of vertices for which v has an incoming edge. We denote the *closed forward neighborhood* of a vertex v as $N^+[v] = N^+(v) \cup \{v\}$, we say that v and u are *twins* if $N^+[v] = N^+[u]$.

2 Reduction Rules

For the reduction rules, we use non-parameterized reductions commonly found in the literature [1, 2]. Due to the reduction rules G can become non-simple, i.e., self-loops or parallel edges are introduced. We can easily prevent inserting parallel edges, so we treat this as an extra step in the following reduction rules, if applicable. These reduction rules are exhaustively applied, note that the order of the reduction rules does not matter. Note that Reduction 1 is a variation of removing all sources and sinks in the graph. Reduction 5 is only used during the *splitting reduction* phase, see Section 4.

▶ **Reduction 1.** Let $\mathcal{C} = \{C_1, \ldots, C_k\}$ be the strongly connected components of G. For every $1 \leq i \leq k$, remove all edges in the cut $(C_i, V \setminus C_i)$. If $C_i = \{v\}$ and v does not have a self-loop, remove v from G.

▶ **Reduction 2.** If $v \in V$ contains a self-loop, remove v with all its incident edges from G and include v in the solution S.

▶ **Reduction 3.** If $N^+(v) = \{u\}$ for some $v \in V$ with $u \neq v$, then redirect all vertices in $N^-(v)$ to u and remove v from G.

▶ **Reduction 4.** If $N^-(v) = \{u\}$ for some $v \in V$ with $u \neq v$, then redirect u to all vertices in $N^+(v)$ and remove v from G.

▶ **Reduction 5** (Twins)**.** Let $X = \{v_1, v_2, \ldots, v_k\}$ with $|X| \geq 2$ be an equivalence class of twins. If the cut $(X, V \setminus X)$ contains only directed or undirected edges, then remove $X \setminus \{v_1\}$ from G and all their incident edges, and include $X \setminus \{v_1\}$ in S.

3 Exploiting Bidirectedness

Let G be a bidirected graph. Finding a directed feedback vertex set of G is equivalent to finding a vertex cover of G. We transform G into an undirected graph G', and then use the solver of Hespe et al., the winning team of the PACE 2019 Exact Vertex Cover track, to obtain a directed feedback vertex set for G. Their solver is based on multiple phases of Branch and Reduce and Branch and Bound [3].

4 Splitting Reduction

Let G' be the reduced graph. If G' is bidirected, we try solve it using the solver from Hespe et al. for five minutes[1], and if G' is not bidirected or we exceeded the time limit, we attempt to perform a *splitting reduction* on G'. The goal of the splitting reduction is to obtain two graphs, a unidirected graph G'_d and an undirected graph G'_u from G', with G'_d possibly empty. Starting from G', for each pair of bidrected edges $(u, v) \in E$ and $(v, u) \in E$, we insert the undirected edge (u, v) in G'_u. We then remove these edges from G', and mark the endpoints as *restricted*. For restricted vertices, only Reductions 1 and 2 are applicable. We then further reduce G'. We repeat this process until G' no longer has any bidirectional edges, obtaining G'_d. During this process, we collect the forced vertices in S', and at the end, we remove all vertices in S' from G'_u.

Finally, if a vertex cover X of G'_u, is a directed feedback set for G'_d, then $X \cup S'$ is a minimum-sized directed feedback vertex set for G'. If we did not use the vertex cover solver yet (because G' is not bidirected), we attempt to find a vertex cover of G'_u. In the event that X is not a directed feedback vertex set for G'_d or we already exceeded the time limit for the vertex cover solver, we resort to constructing an Integer Linear Program. An important note here is that for an optimal solution Y for G'_d, $X \cup Y \cup S'$ is not necessarily an optimal solution for G', hence the need for an alternative method.

[1] Since we are running the binary directly, we need a couple additional threads to read the output buffer, and to terminate the solver after it exceeded the time limit.

5 Integer Linear Programming

We consider the standard ILP formulation. We create the ILP from both G'_d and G'_u. Our goal is to find a set of constraints that has a small solution which is a directed feedback vertex set of G'_d. To do this, we combine a constraint generation phase with finding a good upper bound on these constraints.

Firstly, for our constraints we include all vertex cover constraints from G'_u. Secondly, to generate constraints from G'_d, we compute a *shortcut edge cycle cover* of G'_d. For every edge (u, v) in G'_d, we find a path P from v to u using BFS, which exists, since all reduction rules have been applied exhaustively, and so u and v must be in the same strongly connected component, yielding a cycle C. We sort C and include it in a set $\mathcal{C}$ to prevent generating redundant constraints. If C contains a smaller cycle $C' \subsetneq C$, we use C' instead of C.

Note that these constraints can be transformed into a Hitting Set instance, for which we can apply a Simulated Annealing algorithm to find a good upper bound. We first compute a greedy solution as its initial solution. Next, We uniformly choose a variable that is set to 1, and flip it to 0. Invalidated constraints are then greedily fixed. After one million iterations we return the best solution Z found thus far. If Z is a directed feedback vertex set for G'_d, then we return Z and the generated constraints, otherwise we compute another shortcut edge cycle cover on $G'_d - Z$, i.e., the remaining graph after removing all vertices of Z, and we repeat this until Z is indeed a directed feedback vertex set for G'_d.

To improve the lower bound obtained by the ILP-relaxation, we insert the 3-cliques found in G'_u to the ILP, i.e., for every 3-clique in G'_u at least two of its vertices need to be in the solution. Computing all 3-cliques of G'_u turned out to be efficient, and managed to improve the continuous objective value significantly. Computing 4-cliques was also efficient, but did not yield an improvement in many instances.

For G'_d we can attempt to find a similar structure, namely given a cycle of three vertices $C = (u, v, w)$, we determine whether there is a cycle in the opposite direction from two consecutive vertices without using the third vertex, e.g., we determine whether there exists a path P from v to u such that $P \cap \{w\} = \emptyset$. If we have three such paths P_1, P_2, P_3, then we require that at least two vertices of $P_1 \cup P_2 \cup P_3$ are included. Note that Z is still a solution for the ILP after generating the above constraints, so we feed the solution Z to the ILP to hopefully improve its performance.

Suppose X is the optimal solution of the ILP. If $|X| = |Z|$, then $Z \cup S' \cup S$ must be an optimal solution for G, otherwise we verify whether X is a solution for G'_d and use X instead of Z if it is. If X is not a solution for G'_d, we keep including shortcut edge cycle covers of $G'_d - X$ in the ILP, for which we also compute a new upper bound which we supply to the ILP, until X finally is a solution for G'_d. At the end we can return the optimal solution $X \cup S' \cup S$, where S is the set of vertices of the very first reduction phase.

References

1 Benjamin Bergougnoux, Eduard Eiben, Robert Ganian, Sebastian Ordyniak, and M. S. Ramanujan. Towards a polynomial kernel for directed feedback vertex set. *Algorithmica*, 83(5):1201–1221, may 2021. `doi:10.1007/s00453-020-00777-5`.

2 Philippe Galinier, Eunice Lemamou, and Mohamed Wassim Bouzidi. Applying local search to the feedback vertex set problem. *Journal of Heuristics*, 19(5):797–818, Oct 2013. `doi:10.1007/s10732-013-9224-z`.

3 Demian Hespe, Sebastian Lamm, Christian Schulz, and Darren Strash. *WeGotYouCovered: The Winning Solver from the PACE 2019 Challenge, Vertex Cover Track*. 2020.