KennethLangedal

Kenneth Langedal
University of Bergen

Fredrik Manne
University of Bergen

Johannes Langguth
Simula Research Laboratory, Norway

Abstract

Submission for the PACE challenge 2022 - heuristic track [3]. The heuristic starts by preprocessing the input graph using reduction rules. Existing rules from the directed feedback vertex set (DFVS) and vertex cover (VC) problems are used. Some reduction rules for the VC problem are also relaxed to work in more cases. When the graph can no longer be reduced, a simulated annealing procedure repeats until the stop signal is received. At the end of each cooling schedule, the heuristic also greedily looks for one-zero and two-one swaps before starting a new cooling cycle.

2012 ACM Subject Classification Mathematics of computing → Combinatorial algorithms; Theory of computation → Randomized local search

Keywords and phrases Directed feedback vertex set, local search, simulated annealing

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Supplementary Material `https://github.com/KennethLangedal/DFVS`

1 Preprocessing

Reduction rules are used to reduce the size of the problem so that an optimal solution for the reduced instance can be extended to an optimal solution for the original instance. Many reduction rules are known for the DFVS problem, mainly due to the similarities with the VC problem. The VC problem can be reduced to the DFVS problem by replacing every undirected edge with a cycle of length two. Therefore, reduction rules from the VC literature can be used directly in segments of the graph that only contain cycles of length two.

Reduction rules mainly include or exclude vertices from the solution. These operations are defined as follows.

Include u **in** S. Remove u from the graph along with all adjacent edges. $S = S \cup \{u\}$ and $G' = G \setminus \{u\}$.

Exclude u **from** S. Create new edges from each incoming neighbor to each outgoing neighbor of u. A precondition for excluding a vertex is that it does not have an edge to itself, in which case it must be included in S. The remaining graph is then $G' = G \setminus \{u\}$ and $E' = E \cup \{(v, w) \mid (v, u) \in E \wedge (u, w) \in E\}$.

There are also folding rules that result in smaller instances but require the solution to the reduced instance before deciding whether the removed vertices should be in the solution or not. The reduction rules we have used during preprocessing are as follows.

- Reductions based on in-degree or out-degree less than two [4]. All these vertices can be excluded from the graph.
- Self-loop reduction [4]. These vertices must be part of every feedback vertex set. There are no self cycles in the input data. However, they can appear after other reduction rules.
- PIE, CORE, and DOME rules [5]. The PIE and DOME operations remove edges that are not part of any minimal cycle. CORE refers to a vertex whose neighbors form a clique, and every edge (u, v) in the neighborhood implies that (v, u) also exists.

- Dominance [2] and unconfined [6] for VC. Unconfined is a generalization of the dominance rule, which states that for two adjacent vertices u and v where $N[u] \supseteq N[v]$, we can include u.
- Folding rules for VC based on twins, funnels, and desks [6]. These rules are applied cautiously and only when the distance two neighborhood of the vertices involved exclusively have cycles of length two. This is due to problems where folding can introduce new cycles in the reduced instance that do not need to be broken for the original instance.

A few generalizations are made for some of the VC reduction rules. The dominance reduction rule does not need to be restricted to parts of the graph with only cycles of length two. Given two vertices u and v that form a cycle of length two, if $\Pi[u] \supseteq N^+[v]$ or $\Pi[u] \supseteq N^-[v]$, include u in S. Where $\Pi[u] = N^+[u] \cap N^-[u]$. The correctness of this relaxation follows the same arguments as the dominance rule. Assume u was not part of the solution, then $\Pi(u)$, which includes v, would need to be included. Since $\Pi(u)$ contains every in- or out-neighbor of v, an equally large solution exists that swaps v for u.

This idea can be generalized further. Any minimal cycle can be used as a starting point instead of requiring a cycle of length two. The rule applies if some vertex u dominates every other vertex in the cycle. The solver uses this rule with cycles of lengths two and three.

The twin rule can also be relaxed. Given u and v where $N^+(u) = N^+(v)$ and $N^-(u) = N^-(v)$, if $|N^+(u)| = 2$ or $|N^-(u)| = 2$ they can both be excluded from S. The argument for this is the same as the degree 1 rule. You can always choose the two neighbours instead of u and v.

Some less general patterns are also included. Given a vertex u with exactly two in or out edges, if these two neighbors form a cycle, exclude u. If u had two in and two out edges, it is sufficient that any two neighbors form a cycle. The idea is that after breaking this cycle, u would have in- or out-degree one.

2 Simulated Annealing

The simulated annealing strategy is a modified version of the heuristic by Galinier et al. [1]. The main idea is to consider the complement problem of finding a maximum induced forest (the remaining vertices are the feedback vertex set). The forest is represented by a topological ordering. This representation gives a natural neighborhood of modifying moves that maintain a valid solution. A move consists of inserting a vertex u from the current feedback vertex set into the topological ordering at a specific position. To maintain a valid topological ordering, the outgoing neighbors before u and incoming neighbors after u are removed. Not every position is considered. Only the position just before the first outgoing neighbor or after the last incoming neighbor is considered. These positions will not have conflicts with either the in- or out-neighbors. Restricted to such moves that only have conflicts among either the in- or out-neighbors, these positions also minimize the number of conflicts.

Similarly to Galinier et al. [1], our heuristic uses a classical simulated annealing strategy with a geometric cooling schedule. The temperature T, initialized to T_0, is used to accept or reject moves. If the number of conflicts for a particular move is C, the move would be accepted if either $C \leq 1$ or $e^{\frac{-(C-1)}{T}} \geq \text{rand}()$. Where rand() returns a random value between zero and one. Several random moves are performed before the temperature decreases along $T = T * \alpha$. When T reaches a fixed T_f, the heuristic looks for greedy improvements, described in the next section, before setting $T = T_0$ and continuing. The number of random moves before decreasing the temperature starts from one at $T = T_0$ and then grows linearly up to $|V|$ at $T = T_f$.

3 One-zero and Two-one Swaps

The one-zero swaps use a breadth-first search (bfs) to find vertices that do not cause any cycles if added. Such a vertex could exist not only because of the random nature of the preceding search but also because a back edge in the topological ordering does not necessarily imply the existence of a cycle. In the worst case, checking every vertex costs $|V|^2$ time. However, in practice, the search usually terminates quickly, and the number of vertices to check can be much smaller than $|V|$. Therefore, the one-zero swap step is performed regardless of the size of the graph.

Two-one swaps are identified by first finding every one-one swap for each vertex not currently in the topological ordering. To find every one-one swap for a vertex u, we split it into two abstract vertices s and t, each having only the out and in edges, respectively. Next, s is placed at the start of the topological ordering, and t at the end. Remove every source and sink in two passes, except s and t. At this point, every vertex remaining is part of some $s - t$ path. Finally, starting from s, scan across the ordering while keeping track of the size of the edge cut. If at some vertex it becomes zero, that vertex is a one-one candidate. With this information, finding a two-one swap is only a matter of finding two vertices with a common one-one swap. These two vertices may create a new cycle, so a breadth-first search is also needed at the very end before committing to the swap. In the worst case, looking for two-one swaps could have a $|V|^3$ running time and is also slow on large graphs in practice. Therefore, the two-one swaps are only performed on smaller instances.

4 Implementation Details

The constants used in the final version of the solver are picked based on experiments from the public instances. The initial temperature $T_0 = 0.15$, and the final temperature $T_f = 0.05$. These values are modified slightly depending on the graph and the initial solution after greedy one-zero swaps. The solution size compared to $|V|$ increases T_0 by at most 0.3. Where a smaller solution moves T_0 more, and if they are almost equal, T_0 is unchanged. Furthermore, the average move cost increases T_0 and T_f by at most 0.1, where higher move costs increase it more. The α used is always 0.999.

The topological ordering is stored using a double-linked list, implemented using two fixed-size arrays. Each vertex also has an integer token representing its relative position in the topological ordering. Meaning vertices earlier in the list have smaller tokens, and vertices later have larger ones. This makes it inexpensive to find the two candidate positions mentioned earlier. It is sufficient to look through the neighbors and find the highest token among the in-neighbors or lowest among the out-neighbors. Performing a move is also inexpensive in most cases by simply changing the next and previous pointers in the double-linked list. The token is set to the midpoint between the previous and next vertices. There is a potential to run out of space between these two tokens. When this occurs, the values for the entire topological ordering are generated again. To decrease the frequency of this relabeling step, the tokens are spaced out in the entire 64-bit range.

References

1 Philippe Galinier, Eunice Lemamou, and Mohamed Wassim Bouzidi. Applying local search to the feedback vertex set problem. *Journal of Heuristics*, 19(5):797–818, 2013.

2 Joachim Kneis, Alexander Langer, and Peter Rossmanith. A fine-grained analysis of a simple independent set algorithm. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2009.

3 Kenneth Langedal. Kennethlangedal/dfvs: pace-2022, jun 2022. doi:10.5281/zenodo.6630611.

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

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.

6 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.