


Reduce and Search: A Heuristic DFVS Solver

Florian Sikora  

Université Paris-Dauphine, PSL University, CNRS, LAMSADE, 75016 Paris, France

Abstract

We present a simple solver for DFVS. It heavily uses reduction rules apriori to prune the graph. Afterwards, it tries to build the largest possible DAG by local search moves, using a simulated annealing strategy.

2012 ACM Subject Classification Theory of computation → Simulated annealing

Keywords and phrases Directed Feedback Vertex Set, Heuristic, Reduction Rules, Local Search

Our solver, in C++ [4], proceed in two phases : we first reduce the graph as much as possible to an equivalent instance, and we then construct a solution with local search. We present these two parts in the following.

1 Reduction Rules

Even if the problem is unknown to admit a polynomial kernel, some safe reduction rules are known (for a long time) and used efficiently.

We use two sets of reduction rules. The first ones (5 rules), that we call *basic*, are described by Levy & Low in [2]. They are just degree rules and self loops checking, thus fast to compute.

The second set of reduction rules (3 rules), that we call *advanced*, are described by Lin & Jou in [3]. They require some time consuming subroutines like Strong Connected Components computation.

The 8 rules are applied exhaustively, as long as they can. However, since the advanced rules are more time consuming, we apply them only when the basic ones cannot be applied anymore.

On Figure 1, we can see that the reduction rules largely reduce the number of vertices in a large part of the (public) instances. They also reduce (even more) the number of edges (data not shown due to space constraints).

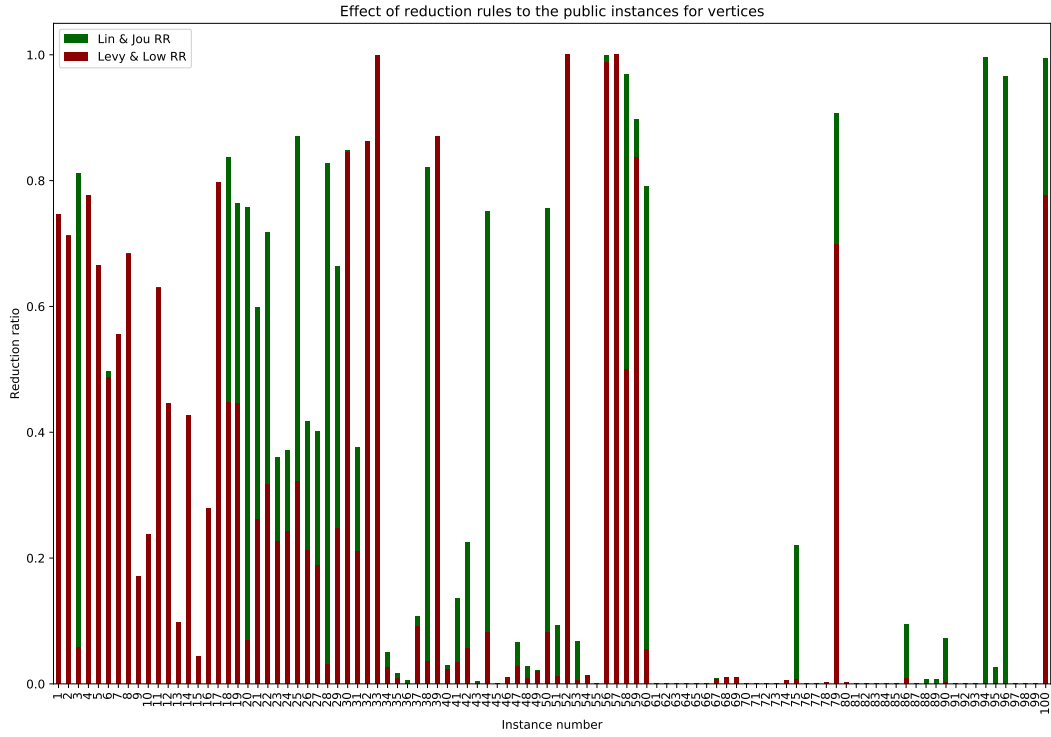
For example, on instance *h_103*, the number of vertices goes from 159 316 to 0 only with the application of the basic rules. Also, on instance *h_187*, the basic reduction rules cannot remove any of the 131 072 initial vertices nor the 3 225 441 initial edges. However, adding the advanced rules creates an equivalent instance with only 516 vertices and 9 108 edges remaining.

However, for several instances, none of the 8 rules can apply. A more detailed study of these instances could lead to additional rules.

2 Local Search

After the application of the reduction rules, we start the local search part, following the algorithm of Galinier et al. [1]. The idea is to build the largest possible induced DAG with the help of local moves (the DFVS solution is therefore the complement of the vertices). The representation of the DAG is made by a topological ordering of the vertices, labeling vertices s.t. for every directed arc from *u* to *v*, *u* got a smaller label than *v* (or equivalently *u* comes before *v* in the ordering). A local move consists in inserting a candidate (unlabeled) vertex in the topological ordering at *some* position (and remove the vertices violating the ordering after this insertion).

■ **Figure 1** Effect of the two sets of reduction rules for each public instance. The ratio is defined as $(n - n')/n$, where n is the number of vertices in the original instance and n' is the number of vertices after the reductions. Thus, a ratio of 0 means that the reduction rules has no effect, while a ratio of 1 means that the resulted instance is empty. Note that the rules of Lin & Jou (advanced rules) come in addition to the ones of Levy & Low (basic), thus are always at least as good.



Local moves are done in a simulated annealing procedure, thus accepting any move making the solution better, but also accepting a “bad” move with a given probability, in order to escape local optimums.

To reduce the number of possibilities for a vertex to be added in the ordering, we follow the procedure of [1], adding the vertex either at a position just after its largest labeled predecessor, or just before its smallest labeled successor.

Altogether, the algorithm is the same as [1, Section 4.1] with the following modifications:

- The reduction rules are exhaustively applied before the local search phase,
- The parameters of the SA procedure are set according to the instance shape (number of vertices, edges, density),
- A greedy solution (adding vertices one by one sorted by difference of degrees if it does not create a cycle) is built *before* the local search for large instances,
- *Restarts* of the local search procedure after too much fails (too much iterations without any improvement) from the previous best solution with a different set of parameters.

Implementation details

In addition to the implementation details given in [1], we also noticed the following.

The relabeling phase after adding a vertex was taking almost the whole computation time. Since it is not a important phase, we chose to avoid this part and we use floating number as labels.

Surprisingly, most of the CPU time was dedicated to *saving* the best solution after an improvement. Indeed, for large instances, since a local move only improve by one the solution, we had to save the whole solution each time (and saving the solution require the copy of a $O(n)$ length vector). We chose to save only every 100 improvements, or if this is the first improvement after a long time, or if the solution is small. This allows us to make more trials during the given time.

Instead of choosing a vertex in the candidate list randomly, we tried to do a weighted choice according to the degree of the vertices. However, this was taking more time than just picking randomly a candidate vertex and we were making way less trials during the given time, thus giving less good solutions.

3 Conclusion

The reduction rules used are quite effective, but maybe more rules could be added.

The second phase (local search with simulated annealing) is maybe too naive. It would be interesting to tune more finely the parameters. Also, making other reductions *during* the local phase stage could help. Using more memory to remember past moves could help (like tabu search). Finally, it would be also interesting to see more involved heuristics (like genetic or memetic algorithms) to compare performances, or to try some Monte Carlo tree search for determining the best vertices to add in the solution.

Source code is available at: [10.5281/zenodo.6624196](https://zenodo.org/record/6624196)

References

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

- 2 Hanoch Levy and David W. Low. A contraction algorithm for finding small cycle cutsets. *J. Algorithms*, 9(4):470–493, 1988. doi:10.1016/0196-6774(88)90013-2.
- 3 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.
- 4 Florian Sikora. <https://github.com/fsikora/pace22>. Jun 2022. doi:10.5281/zenodo.6624196.