PACE Solver Description: UzL Exact Solver for One-Sided Crossing Minimization

Max Bannach
European Space Agency, Noordwijk, The Netherlands

Florian Chudigiewitsch
Institute for Theoretical Computer Science, University of Lübeck, Germany

Kim-Manuel Klein
Institute for Theoretical Computer Science, University of Lübeck, Germany

Marcel Wienöbst[1]
Institute for Theoretical Computer Science, University of Lübeck, Germany

Abstract

This document contains a short description of our solver *pingpong* for the one-sided crossing minimization problem that we submitted to the exact and parameterized track of the PACE challenge 2024. The solver is based on the well-known reduction to the weighted directed feedback arc set problem. This problem is tackled by an implicit hitting set formulation using an integer linear programming solver. Adding hitting set constraints is done iteratively by computing heuristic solutions to the current formulation and finding cycles that are not yet "hit." The procedure terminates if the exact hitting set solution covers all cycles. Thus, optimality of our solver is guaranteed.

2012 ACM Subject Classification Theory of computation → Integer programming; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases integer programming, exact algorithms, feedback arc set, crossing minimization

Digital Object Identifier 10.4230/LIPIcs.IPEC.2024.28

Supplementary Material *Software (Exact Track)*: `https://github.com/mwien/pingpong`
Software (Parameterized Track): `https://github.com/mwien/pingpong-light`

1 Introduction

One-sided crossing minimization is a fundamental problem in graph drawing. For a given bipartite graph $G = (V_1 \cup V_2, E)$ and a linear ordering τ of V_1, the goal is to find a linear ordering π of V_2 that minimizes the number of *crossings*, i.e., tuples $(\{u_1, u_2\}, \{v_1, v_2\}) \in E^2$ such that $\tau^{-1}(u_1) < \tau^{-1}(v_1)$ and $\pi^{-1}(u_2) > \pi^{-1}(v_2)$, with $\tau^{-1}(x)$ and $\pi^{-1}(x)$ being the position of vertex x in the respective ordering. Vice versa $\pi(i)$ and $\tau(i)$ denote the vertex at position i in π and τ, respectively.

The objective can be formulated concisely using the notion of crossing numbers: If c_{uv} is the number of crossings of edges incident to u or v given that u is ordered to the left of v, the goal is to find a linear ordering π that minimizes

$$\sum_{i=1}^{|V_2|} \sum_{j=i+1}^{|V_2|} c_{\pi(i)\pi(j)}.$$

[1] Corresponding author

19th International Symposium on Parameterized and Exact Computation (IPEC 2024).
Editors: Édouard Bonnet and Paweł Rzążewski; Article No. 28; pp. 28:1–28:4
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In any linear ordering, either u comes before v or the other way around and, hence, it suffices to consider the difference $c_{uv} - c_{vu}$. A sensible objective function is thus:

$$\min_{\pi} \sum_{i=1}^{|V_2|} \sum_{j=i+1}^{|V_2|} \max(c_{\pi(i)\pi(j)} - c_{\pi(j)\pi(i)}, 0)$$

This formulation is well-known to be identical to the weighted directed feedback arc set problem with arc $u \to v$ having weight $c_{vu} - c_{uv}$ if this weight is positive. To solve this instance, it is obvious that each strongly connected component can be considered separately. These insight already renders a considerable amount of instance trivially solvable.

2 Implicit Hitting Set Formulation of Feedback Arc Set

The *feedback arc set* problem (FAS) can be expressed as an instance of the *hitting set* problem: Add a set per cycle containing all its edges. Clearly, such a formulation in its explicit form is infeasible with the number of cycles growing exponentially in the size of the FAS instance.

A better approach is to start with a small set of cycles and iteratively add more constraints. This approach is known as the *implicit hitting set* algorithm [3] and, in the context of integer programming, also referred to as *lazy constraint generation* or *row generation*. In its simplest form (which can be refined in many ways), such an iterative procedure could look like the following for FAS:

1. Initialize the set of cycle constraints C in some way.
2. Repeatedly,
   a. find the optimal solution of the hitting set instance C and
   b. check if this solution is an FAS. If this is the case, then terminate and output the solution. If not, then add cycles to C that are not "hit" by the current solution.

The procedure terminates only when the optimal hitting set solution contains an edge per cycle in the FAS instance, guaranteeing correctness. Importantly, the algorithm often terminates before all cycles of G were added, thus making it more practical than the naive explicit formulation mentioned above.

An implementation of this general method for solving FAS has recently been described in [1] and was discussed earlier in the form of a branch-and-cut algorithm in [5]. In both cases, integer programming is used to find the optimal solution to the hitting set instance. Denoting the number of vertices in the FAS instance by n, the number of edges by m and having a variable x_i per edge with weight w_i as given in the FAS instance, one obtains

$$\begin{aligned} &\min_{x} \sum_{i=1}^{m} w_i x_i \\ &\text{s.t.} \sum_{x_i \in C_j} x_i \geq 1, \quad \text{for each } j = 1, 2, \ldots, |C| \\ &\quad x_i \in \{0, 1\} \quad \text{for all } i \in \{1, \ldots, n\}, \end{aligned}$$

where the hitting set constraints are added lazily until there are no more violations.

We note that the same constraints could also be expressed with a MaxSAT formulation. However, early benchmarks showed that MaxSAT solvers perform significantly worse for the instances in this challenge obtained through the reduction from the one-sided crossing minimization. Hence, we abandoned this approach and focused on the ILP formulation.

3 Description of Our Approach

We follow the general method described above closely. In this section, we provide more details regarding our concrete implementation.

We solely used cycles of length three as the initial set of cycles C for most of the challenge and this approach is still used for larger instances (where a subset of all 3-cycles is randomly selected). Shortly before the submission deadline, we switched to a more involved generation procedure for the other instances, in which we first run our heuristic solver [2] to find a good initial feedback arc set and generate cycles based on it (see more details below on how new cycles are generated). The improvements here are, however, minor even in cases when the heuristic can already identify the optimal solution.[2]

Generally, cycles are added based on utilizing a (not necessarily optimal) solution of the current hitting set formulation. For this, we consider the subgraph obtained by removing all edges in this solution set. In the resulting subgraph, an FAS is found by a heuristic, which builds the topological order greedily from left to right,[3] and for each edge $u \to v$ in this FAS, new cycles are generated by finding short paths from v to u using a breadth-first search. Our implementation of the approach from the previous section can be described as follows:

1. Initialize the set of cycle constraints C.
2. Find a hitting set based on a degree-based heuristic and add cycles constraints as described above. If a non-empty set of cycles is added, repeat this step.
3. Find a hitting set based on a rounded LP solution and add cycle constraints as described above. If a non-empty set of cycles is added, repeat this step.
4. If the objective value of the rounded LP solution and the fractional LP solution differ by less than one, the found hitting set is optimal. In this case, output the FAS and terminate.
5. Try to add violated cycle constraints based on the fractional LP solution. If a non-empty set of cycles is added, go to step 2.
6. Find a hitting set by solving the ILP optimally. If this set is not an FAS, add further constraints as described above and go to step 2. Else output the FAS and terminate.

So, instead of directly starting an exact solver on the given hitting set instance, we first use heuristics for the purpose of adding new cycle constraints. Due to this interplay between the heuristics for hitting set and the ones for FAS that are combined to generate cycle constraints, we name our solver *pingpong*.

As hitting set heuristics we use, on the one hand, a simple degree-based heuristic (which has the advantage of being extremely fast) and, on the other hand, a relaxed version of the LP without the integrality constraint enforced (i.e., having $0 \leq x_i \leq 1$). For cycle generation, we round the solution and further improve it iteratively by a simulated annealing scheme. Further constraints are added this way as long as the heuristic solution does not cover all cycles, i.e., is not a valid FAS. Our goal is to start the ILP solver only a few times – or not at all if the rounded LP solution matches the objective value of the fractional LP solution.

Once this subgraph is acyclic, we try to add further cycles based on the fractional LP solution. We do this only once as this procedure, which is based on Dijkstra's algorithm [4] to find violated LP constraints, has significantly larger computational cost compared to the

[2] Interestingly, our heuristic solver finds optimal solutions quickly even for some of the instances our exact solver fails to solve within the 30 minute time limit. This shows that proving the lower bound is the main problem for these inputs.

[3] The next vertex to place in the ordering is chosen such that the sum of violated edge weights is minimized.

BFS cycle generation procedure. If it adds no further cycles, the integer program is started and solved optimally. In the same way as before, the solution of the integer program is used to add further cycles, unless it already hits every cycle – in this case we found the optimum and terminate.

We used the HiGHS ILP solver [6] without any further tuning. As this solver does not yet offer lazy constraint generation, we restarted the solver for any new hitting set instance. In the future, it would be interesting to analyze how much gains could be made when implementing a lazy constraint callback.

Our algorithm manages to solve 195 of the 200 total instances in the exact track. Due to randomness in the cycle generation procedure the run-time can fluctuate for some instances. For the parameterized track, we submitted a simplified version of our solver, which gives slightly better performance on smaller and easier instances. For example, it does not use the degree heuristic for hitting set and the initial cycles are simply all cycles of length 3. The solver does not make use of the given cutwidth ordering – still, it solves all instances in about a minute total.

References

1 Ali Baharev, Hermann Schichl, Arnold Neumaier, and Tobias Achterberg. An exact method for the minimum feedback arc set problem. *Journal of Experimental Algorithmics (JEA)*, 26:1–28, 2021.

2 Max Bannach, Florian Chudigiewitsch, Kim-Manuel Klein, Till Tantau, and Marcel Wienöbst. UzL heuristic solver for one-sided crossing minimization. Technical report, University of Lübeck, 2024.

3 Karthekeyan Chandrasekaran, Richard M. Karp, Erick Moreno- Centeno, and Santosh S. Vempala. Algorithms for implicit hitting set problems. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 614–629, 2011. doi:10.1137/1.9781611973082.48.

4 Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959. doi:10.1007/BF01386390.

5 Martin Grötschel, Michael Jünger, and Gerhard Reinelt. A cutting plane algorithm for the linear ordering problem. *Operations research*, 32(6):1195–1220, 1984. doi:10.1287/OPRE.32.6.1195.

6 Qi Huangfu and JA Julian Hall. Parallelizing the dual revised simplex method. *Mathematical Programming Computation*, 10(1):119–142, 2018. doi:10.1007/S12532-017-0130-5.