



# PACE Solver Description: LUNCH — Linear Uncrossing Heuristics

Kenneth Langedal<sup>1</sup>  

University of Bergen, Norway

Matthias Bentert 

University of Bergen, Norway

Thorgal Blanco 

University of Bergen, Norway

Pål Grønås Drange  

University of Bergen, Norway

---

## Abstract

The 2024 PACE challenge is on ONE-SIDED CROSSING MINIMIZATION: Given a bipartite graph with one fixed and one free layer, compute an ordering of the vertices in the free layer that minimizes the number of edge crossings in a straight-line drawing of the graph. Here, we briefly describe our exact, parameterized, and heuristic submissions. The main contribution is an efficient reduction to a weighted version of DIRECTED FEEDBACK ARC SET, allowing us to detect subproblems that can be solved independently.

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

**Keywords and phrases** graph drawing, feedback arc set, algorithm engineering

**Digital Object Identifier** 10.4230/LIPIcs.IPEC.2024.34

**Category** PACE Solver Description

**Supplementary Material** Public Code Repository

*Software (Source Code):* <https://github.com/KennethLangedal/PACE2024-UiB> [1]

## 1 Introduction

Let  $G = ((A \uplus B), E)$  be an undirected bipartite graph with vertex partition  $(A, B)$ . In ONE-SIDED CROSSING MINIMIZATION (OCM), an ordering  $\tau$  of  $A$  is given and the task is to compute an ordering  $\pi$  for  $B$  that minimizes the edge crossings in a straight-line drawing of  $G$ . The number of edge crossings for a linear ordering of  $B$  can be computed by comparing pairs of vertices in  $B$  separately. Let  $c_{u,v}$  denote the number of edge crossings between  $u$  and  $v$  when  $u$  is placed before  $v$ . The cost of an ordering  $\pi$  of  $B$  is  $\sum_{u,v \in B | \pi(u) < \pi(v)} c_{u,v}$ . We will denote by  $\ell_u$  and  $r_u$  the leftmost and rightmost neighbors of  $u$  with respect to  $\tau$  in  $A$ .

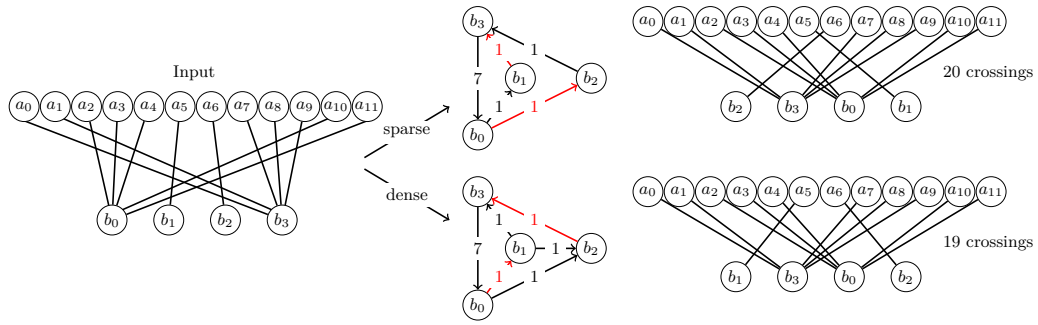
## 2 Preprocessing

The first step of both our exact and heuristic solvers is to reduce OCM to the weighted version of DIRECTED FEEDBACK ARC SET (DFAS). The instance for the latter problem will contain a vertex  $v'$  for each vertex  $v \in B$ . For each pair  $u, v$  of vertices in  $B$ , compute  $c_{u,v}$  and  $c_{v,u}$ . Add a directed arc  $(u', v')$  of weight  $c_{v,u} - c_{u,v}$  if  $c_{u,v} \leq c_{v,u}$ . Otherwise, add the arc  $(v', u')$  with weight  $c_{u,v} - c_{v,u}$ . Assume without loss of generality that  $c_{u,v} < c_{v,u}$ . Then,

---

<sup>1</sup> Corresponding author





■ **Figure 1** Example showing how an optimal solution to a sparse component can be suboptimal. The two DFAS instances in the middle have optimal solutions shown by red edges. When lifting the solution back to the OCM problem, the sparse component places  $b_1$  and  $b_2$  in the wrong order.

if we place  $u'$  before  $v'$  (which corresponds to placing  $u$  before  $v$ ), then we pay  $c_{u,v}$ , and if we place  $u'$  after  $v'$ , then  $c_{v,u} = c_{u,v} + (c_{v,u} - c_{u,v})$ . Since we are only interested in minimizing the number of crossings and the difference between these two options is preserved, we get an equivalent instance of DIRECTED FEEDBACK ARC SET.

One very effective reduction in the constructed DFAS instance is that edges between strongly connected components can be deleted since these edges are not part of any cycles. Furthermore, strongly connected components can be solved independently. We initially construct a sparse instance to speed up the search for strongly connected components. Two types of edges will be ignored at this stage: (1) edges  $(u', v')$  where  $c_{u,v} = 0$  and (2) edges  $(u', v')$  where  $c_{u,v} = c_{v,u}$ . The construction of this sparse instance works as follows:

- 1:  $\pi \leftarrow \{\ell_{u_0} \leq \ell_{u_1} \dots \leq \ell_{u_{|B|-1}}\}$
- 2: **for**  $i = 0, \dots, |B| - 1$  **do**
- 3:     **for**  $j = i + 1, \dots, |B| - 1$  **do**
- 4:          $u \leftarrow \pi[i], v \leftarrow \pi[j]$
- 5:         **if**  $r_u \leq \ell_v$  **then**
- 6:             **break** ▷ all remaining pairs have  $c_{u,v} \neq 0$  and  $c_{v,u} \neq 0$
- 7:         **if**  $c_{uv} < c_{vu}$  **then**
- 8:             Add edge  $(u', v')$  with weight  $c_{vu} - c_{uv}$
- 9:         **if**  $c_{uv} > c_{vu}$  **then**
- 10:             Add edge  $(v', u')$  with weight  $c_{uv} - c_{vu}$

The break statement on lines 5 and 6 (highlighted in red) is what makes this procedure worthwhile. When we first encounter a pair of vertices that have  $c_{u,v} = 0$ , we know that every remaining  $u, v$  pair in the inner most loop must also have  $c_{u,v} = 0$ , since  $\ell_v$  is increasing. This improvement had large effects on many of the test instances. We will show next that looking for strongly connected components in this graph is safe. However, such edges within a strongly connected component cannot be ignored as the example in Figure 1 shows.

We proceed by showing that ignoring edges between different strongly connected components in the described graph  $G'$  is safe. To this end, we consider three types of arcs (represented by colors). We color an arc  $(u', v')$  as follows. If  $c_{u,v} = 0$ , then we color the arc red. If  $c_{u,v} = c_{v,u}$ , then we color the arc green. All other arcs (those that we do not ignore) are blue. We will repeatedly make use of the following lemma.

► **Lemma 1.** *If  $(u', v')$  is a red arc and  $(v', w')$  is a blue arc, then  $(w', u')$  cannot be a green or blue arc. The same holds if  $(u', v')$  is blue and  $(v', w')$  is red.*

**Proof.** We will only show the statement for  $(u', v')$  being red as the other case is symmetric. Assume towards a contradiction that  $(w', u')$  is a green or blue edge. Note that  $(u', v')$  being red implies that  $\ell_v > r_u$ . Let  $L, R \subseteq A$  be the set of neighbors of  $w$  to the left/right of  $r_u$ , that is,  $a \in A$  belongs to  $L$  if  $\tau(a) < \tau(r_u)$  and  $a$  belongs to  $R$  if  $\tau(a) > \tau(r_u)$ . Note that  $c_{w,u} \geq |N(u)| \cdot |R|$ . Since  $(w', u')$  is green or blue, it holds that  $c_{u,w} \geq c_{w,u} \geq |N(u)| \cdot |R|$ . This implies that  $|L| \geq |R|$ . However, since  $\ell_v > r_u$ , this implies also that  $c_{v,w} \geq |L| \cdot |N(v)| \geq |R| \cdot |N(v)| c_{w,v} \geq c_{w,v}$  which contradicts the fact that  $(v', w')$  is a blue arc.  $\blacktriangleleft$

We need to show that there is no directed cycle consisting of arcs with positive weights that contains a red or green arc  $(u', v')$  where  $u'$  and  $v'$  belong to different strongly connected components in the graph induced by all blue arcs. Note that since all green arcs have weight 0, they can never be part of such a cycle. So assume towards a contradiction that there exists a directed cycle  $C$  that contains a red arc  $(u', v')$  where  $u'$  and  $v'$  belong to different strongly connected components in the graph induced by all blue arcs and all arcs in  $C$  are red or blue. We assume without loss of generality that  $C$  is the shortest (in terms of number of vertices) such cycle. Let  $C = (v' = w'_0, w'_1, \dots, w'_c = u')$ . Note that by definition  $(w'_i, w'_{i+1})$  exists and is either a blue or a red arc. Note that if any red or blue arc  $(w'_i, w'_j)$  with  $i < j - 1$  exists, then this is a shortcut and contradicts the fact that  $C$  is a shortest cycle. Moreover, any red arc  $(w'_i, w'_j)$  with  $i > j$  (other than  $i = c$  and  $j = 0$ ) would also imply a shorter cycle than  $C$ .

Next, assume that some arc  $(w'_i, w'_{i+1})$  is red. Now any blue arc  $(w'_j, w'_k)$  with  $j > i$  and  $k \leq i$  would contradict the fact that  $C$  is a shortest cycle. Hence, all such arcs are green. Now consider the arc  $(w'_{i-1}, w'_i)$  (where  $w'_{i-1} = u'$  if  $i = 0$ ). If this arc is red, then  $r_{w_{i-1}} \leq \ell_{w_i} \leq r_{w_i} \leq \ell_{w_{i+1}}$ , showing that  $(w'_{i-1}, w'_{i+1})$  is a red arc, a contradiction. Hence, the arc is blue (and  $w'_{i-1} \neq u'$ ). However, now  $(w'_{i-1}, w'_i)$  is blue,  $(w'_i, w'_{i+1})$  is blue, and  $(w'_{i+1}, w'_{i-1})$  is green as  $i + 1 > i$  and  $i - 1 \leq i$  shows that the arc cannot be blue and it can also not be red as shown above. This contradicts Lemma 1 and shows that all arcs  $(w'_i, w'_{i+1})$  are blue.

To conclude the argument that  $C$  cannot exist, consider the pair  $\{u', w'_1\}$ . By Lemma 1, there cannot be a green arc between the two. We now consider two cases,  $c = 2$  (that is,  $C$  consist of  $u', v'$ , and  $w'_1$ ) or  $c > 2$ . If  $c = 2$ , then  $(w'_1, u')$  is a blue arc and this contradicts Lemma 1. If  $c > 2$ , then there cannot be an arc  $(w'_1, u')$  as this would contradict the fact that  $C$  is a shortest cycle. Hence, in this case  $(u', w')$  is a blue arc (it cannot be red arc as this would mean that we could exclude  $v'$  from  $C$  to get a shorter cycle through a red arc). Hence,  $w'_1$  and  $u'$  (and in fact all  $w'_i$  other than  $v'$ ) belong to the same strongly connected component in  $G'$ . Now consider any vertex  $w'_i$  with  $i \notin \{0, 1, c\}$ . As shown above, the arc  $(v', w'_i)$  is not red or blue. Hence, the arc  $(w'_i, v')$  is red, blue, or green. It cannot be red as this would result in a shorter cycle than  $C$  through this arc. It can also not be blue as this would mean that  $v'$  is in the same strongly connected component in  $G'$  as  $w'_i$  (and therefore as  $u'$ ). Thus, all such edges are green but this means that the edge  $(v', w'_{c-1})$  is green, a final contradiction to Lemma 1.

### 3 Tiny components

We use a dynamic-programming algorithm to solve tiny components quickly when the number of vertices is at most twenty. It could also be used to solve larger components, but our dedicated exact solver will solve larger instances faster. The algorithm is based on the dynamic-programming algorithm for DFAS by Lawler [2] and can simply be described by the recurrence

$$\text{dp}(\emptyset) = 0 \quad | \quad \text{dp}(S) = \min_{u \in S} \text{dp}(S \setminus \{u\}) + \text{deg}(u, S)$$

where  $\text{deg}(u, S)$  is the number of edges going from  $u$  to  $S$ .

#### 4 Heuristic

Our heuristic mainly relies on the cutting technique introduced by Park and Akers [3]. Instead of explicitly breaking cycles, consider the DFAS problem as finding an ordering of the vertices that minimizes the number of edges going from right to left. Now, the cutting technique used by Park and Akers searches every continuous subgraph in the current ordering, and any cut within each subgraph. If, at any time, the number of edges going backward across the cut is larger than the number of edges going forward across the cut, we swap the vertices before and after the cut. While this procedure seems like it would take  $O(n^4)$  time, it can be done in  $O(n^3)$  time using a cut matrix [3]. To speed up the computation further, we also limit the distance the cut can be from any side of the subgraph.

In several instances, the cutting idea is still too slow, so we only use it after cheaper greedy improvements fail to make progress. We randomize the current solution to escape local minimums by swapping 1–3 random pairs of vertices while always returning to the best solution if the next local minimum was worse. In very few cases, the graphs are so large that reducing to DFAS is impossible without exceeding the memory or time limits. In these cases, we only use greedy improvements while repeatedly computing  $c_{u,v}$  when needed.

#### 5 Exact

We first run our heuristic for each large component to get an upper bound on the DFAS solution. Then, our exact method starts by enumerating all short cycles in the graph (cycles with at most four vertices). Then, create a MAXSAT instance where each cycle is a hard constraint requiring at least one of the edges in the cycle to be picked. Every edge also has its own soft constraint with the weight of the edge. We then solve this MAXSAT instance using the solver UWrMaxSat [4]. There are now two termination conditions: (1) after removing the edges from the MAXSAT solution, the resulting graph is acyclic, and (2) the cost of the MAXSAT solution is equal to our upper bound. In both cases, we have an optimal solution. In the first, the edges removed in our MAXSAT instance also make an optimal solution to the DFAS problem. In the second case, we know our upper-bound solution was optimal. Otherwise, the solver proceeds by temporarily removing the edges in the latest solution from the MAXSAT instance. Since this graph is not acyclic, we can find new cycles using a depth first search. We add these cycles to our MAXSAT instance and repeat until we hit one of the two termination conditions mentioned above.

---

#### References

- 1 Kenneth Langedal. KennethLangedal/PACE2024-UiB: pace-2024, June 2024. doi:10.5281/zenodo.11540761.
- 2 Eugene L. Lawler. A comment on minimum feedback arc sets. *IEEE Transactions on Circuit Theory*, 11(2):296–297, 1964.
- 3 Sungju Park and Sheldon B. Akers. An efficient method for finding a minimal feedback arc set in directed graphs. In *1992 IEEE International Symposium on Circuits and Systems*, volume 4, pages 1863–1866. IEEE, 1992.
- 4 Marek Piotrów. UWrMaxSat: Efficient solver for MaxSAT and pseudo-boolean problems. In *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 132–136. IEEE, 2020.