PACE 2026 Solver Description: CherryPicker, a Heuristic Solver for the Maximum Agreement Forest Problem

Jakub Dziura
University of Warsaw, Poland

Tomáš Masařík
University of Warsaw, Poland

Abstract

This document contains a short description of the solver CherryPicker for the Maximum Agreement Forest (MAF) problem, submitted to the heuristic and lower-bound tracks of the PACE Challenge 2026. The heuristic solver combines a small set of standard reduction rules—singleton elimination, common-cherry reduction, and a forced-pendant-cut rule—with an iterated local search (ILS). For the lower-bound track, the solver uses a column generation method.

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization; Theory of computation → Design and analysis of algorithms; Mathematics of computing → Solvers

Keywords and phrases Maximum agreement forest (MAF), iterated local search (ILS), column generation, solution with lower-bound guarantees, LP relaxation, PACE Challenge

Category PACE Solver Description

Supplementary Material Code repository
URL: `https://github.com/jdziura/PACE2026_CherryPicker`

Acknowledgements This is a student submission to the PACE Challenge 2026 because the main contributor, Jakub Dziura, is a full-time student. The role of Tomáš Masařík was to supervise the project. The authors would like to thank Piotr Blinowski for valuable discussions in the initial stages of the project.

1 Introduction and Overview

The Rooted Maximum Agreement Forest (MAF) problem for two binary trees is defined as follows. The input is a pair of rooted trees (T_0, T_1) on the same set of n leaves with labels $\{0, 1, \ldots, n-1\}$. The task is to find the minimum number of edges $C \subseteq E(T_0) \cup E(T_1)$ such that $T_0 - C$ and $T_1 - C$ are isomorphic, respecting the labels on leaves after contractions of degree-two vertices. If $T_0 - C$ and $T_1 - C$ are isomorphic, we call the resulting object ($T_0 - C$ after degree-two contractions) an *agreement forest*. We refer to a not necessarily minimum set $C \subseteq E(T_0) \cup E(T_1)$ as an *edge cut*. We say that an edge cut is *valid* if it produces an agreement forest, and *partial* otherwise. We present algorithms that solve this problem heuristically and with a lower-bound guarantee as a submission to the PACE Challenge 2026 [5]. The convention is to count the number of components in the output forest, that is, $\frac{|C|}{2} + 1$. The minimum such number is denoted as OPT. We define a *cherry* in a forest F as a pair of leaves that share the same parent node in F. Given a pair of forests (F_0, F_1), we say that a pair of distinct leaves $a, b \in X$ is a *conflicting cherry* if it is a cherry in exactly one of F_0 or F_1. We would like to point out, unlike many common approaches in the literature that maintain an asymmetric (tree, forest) pair, a key feature of our algorithms is the full symmetry of operations applied to the trees. As reflected in the definitions, we operate on both T_0 and T_1 explicitly, allowing us to simultaneously consider conflicting cherries and

Algorithm 1 Heuristic algorithm pseudocode.

Input: Trees T_0, T_1
Output: A valid edge cut C

$C \leftarrow$ an initial solution obtained by starting with the partial edge cut $\emptyset$ and repeatedly applying Repair() and Reductions().
$C^\star \leftarrow C$
repeat
  $C \leftarrow \textsc{Perturb}(C)$
  while C is not a valid edge cut **do**
    $D \leftarrow \textsc{Repair}(C)$
    $C \leftarrow \textsc{Reductions}(C \cup D)$
  end while
  if $|C| \leq |C^\star|$ **then**
    $C^\star \leftarrow C$
  end if
until a stop signal has been received
return $C^\star$

reductions on either side.

1.1 Heuristic Algorithm Overview

The heuristic algorithm runs an iterated local search (ILS) loop until it receives a stop signal. Each subsequent *iteration* starts with a valid edge cut and tries to improve it with the operations described in detail in the subsequent sections. Within one iteration, our solver maintains a partial edge cut denoted by C' together with the corresponding pair of forests $(F_0 \coloneqq T_0 - C', F_1 \coloneqq T_1 - C')$. See Algorithm 1 for the pseudocode overview.

1. **Perturbation (Section 2.1).** A subset of edges from a valid edge cut is selected at the start of each iteration to form an initial partial edge cut for the repair loop.
2. **Repair (Section 2.2).** Starting from an arbitrary partial edge cut, the solver greedily finds a conflicting cherry and chooses which edges to cut next. This process repeats until a valid edge cut is produced.
3. **Reductions (Section 2.3).** Whenever an edge is added to a partial edge cut, we exhaustively apply a set of standard reduction rules [3, 7]: singleton elimination, common-cherry reduction, and forced pendant cuts.

A candidate solution updates the incumbent best solution whenever it is no worse than the current best. The initial valid edge cut is constructed by starting with $C' = \emptyset$ and running repair/reduce until an agreement forest is reached.

A detailed description of the heuristic algorithm is given in Section 2.

1.2 Lower-Bound Algorithm Overview

The lower-bound track provides numbers a and b such that the task is to provide a solution to MAF with at most $\lfloor a \cdot \mathrm{OPT} + b \rfloor$ components. The lower-bound algorithm can be seen as an extension of our heuristic algorithm. The only difference is that before running the ILS stage, we derive a *lower-bound certificate* k such that $k \leq \mathrm{OPT}$. Then we run the heuristic local search algorithm described in Section 1.1 until it reaches a number of components

at most $\lfloor a \cdot k + b \rfloor$. For the lower-bound certificate, we implement a solver based on the branch-and-price algorithm by Frohn, Kelk, and Vychytilova [1]. We adapt the block-based linear programming (LP) formulation by Olver, Schalekamp, Ster, Stougie, and Zuylen [4] to rooted trees. A detailed description of the lower-bound certificate generation is given in Section 3.

2 Heuristic Algorithm Description: Iterated Local Search

Here, we present a detailed description of the individual steps of our heuristic algorithm. The initial solution is found by a repair/reductions loop starting with an empty partial edge cut. See the pseudocode in Algorithm 1 for an overview.

2.1 Perturbation

Let C be a valid edge cut from the previous iteration. The solver draws, uniformly at random:

- a tree index $i \in \{0, 1\}$;
- an integer count $k \in [k_{\min}, k_{\max}]$;
- a seed edge $e \in C \cap E(T_i)$.

It then chooses a *discard* set $D \subseteq C \cap E(T_i)$ by one of the following two strategies.

- **Close-neighborhood** (default): up to k edges of $C \cap E(T_i)$ that are closest to e in T_i (found by breadth-first search).
- **Uniform** (small probability): each edge of $C \cap E(T_i)$ is discarded independently at random with probability $\min\{1, \frac{k}{|C \cap E(T_i)|}\}$.

This step outputs a partial edge cut $C' \coloneqq (C \setminus D) \cap E(T_i)$.

2.2 Repair

The repair phase can be viewed as a randomized modification of the FPT algorithm of Whidden, Beiko, and Zeh [7]. At each step, a single move is selected at random (with configurable weights) and applied immediately.

Throughout the algorithm, we maintain an updated set of cherries for both F_0 and F_1; see Section 4.1 for more details. First, the algorithm selects uniformly at random $i \in \{0, 1\}$. Second, it picks an arbitrary cherry (a, c) from F_i. If a and c are connected in F_{1-i}, we proceed as follows. Let $b_1, \ldots, b_s$ be all the pendant subtrees on the ac-path in F_{1-i}. We select one of the following three moves with probabilities described below.

1. **Cut a:** cut the edge above a in both forests.
2. **Cut c:** cut the edge above c in both forests.
3. **Cut the sibling path:** cut the edges above $b_1, \ldots, b_s$ in F_{1-i}.

If a and c are not connected in F_{1-i}, we uniformly sample from the first two cases only. The algorithm uses the following probabilities when move 3 is available. We choose move 3 with probability $p(s)$, where $p(2) = \frac{3}{5}$, $p(3) = \frac{1}{2}$, $p(4) = \frac{1}{3}$, and $p(s) = \frac{1}{5}$ for $s \geq 5$ (we favor small s). With the remaining probability, we choose move 1 or move 2 uniformly at random.

2.3 Reductions

Whenever edges are added to a partial edge cut, the following standard reduction rules [3, 7] are applied exhaustively.

1. **Contraction of degree-two vertices.**

2. **Singleton reduction:** If a leaf u is an isolated vertex (a singleton component) in F_i but not in F_{1-i}, then u must be separated in F_{1-i} as well. We cut the parent edge of u in F_{1-i}.
3. **Common-cherry reduction:** If leaves a and b are siblings in both F_i and F_{1-i}, the two forests agree on this pair, so it can be contracted without any cut. We merge a and b into a single leaf in both forests, adding no edge to C.
4. **Forced pendant cut:** Suppose leaves a and c are siblings in some forest F_i, while the ac-path in F_{1-i} contains exactly one pendant subtree rooted at b. Then we cut the edge above b.

3 Lower-Bound Solver

We describe the LP formulation for the MAF problem [1, 4] in Section 3.1. In Section 3.2, we list the improvements implemented to speed up solving the LP relaxation from Section 3.1. As discussed in the introduction, we finish by running the heuristic algorithm described in Section 2 to find a solution that satisfies the computed lower-bound threshold (or runs indefinitely if it is unable to compute such a solution).

3.1 Linear Program Definition

We follow the notation given in [1]. Let X be the set of leaves. For $Y \subseteq X$, let $V_i[Y]$ denote the internal (non-leaf) vertices of the induced subtree of T_i for $i \in \{0, 1\}$. Furthermore, let $V[Y] = V_0[Y] \cup V_1[Y]$. Let $\mathcal{Y}$ be the (exponentially large) family of all sets of leaves that induce a connected component in a valid agreement forest. This set-partitioning formulation was shown in [4] to be a valid LP relaxation of the (integer) MAF problem. Consequently, the optimal value of its LP relaxation is a lower bound for MAF.

The relaxed *primal LP* formulation is:

$$
\begin{aligned}
\min \quad & \sum_{Y \in \mathcal{Y}} a_Y \\
\text{s.t.} \quad & \sum_{\substack{Y \in \mathcal{Y} \\ x \in Y}} a_Y \geq 1 && \forall x \in X \\
& \sum_{\substack{Y \in \mathcal{Y} \\ v \in V[Y]}} a_Y \leq 1 && \forall v \in V[X] \\
& a_Y \geq 0 && \forall Y \in \mathcal{Y}
\end{aligned}
$$

Now, we describe the corresponding *dual* program, where α_x is the *coverage dual* of leaf $x \in X$ and β_v is the *packing dual* of internal node v:

$$
\begin{aligned}
\max \quad & \sum_{x \in X} \alpha_x - \sum_{v \in V[X]} \beta_v \\
\text{s.t.} \quad & \sum_{x \in Y} \alpha_x - \sum_{v \in V[Y]} \beta_v \leq 1 && \forall Y \in \mathcal{Y} \qquad (1) \\
& \alpha_x \geq 0 && \forall x \in X \\
& \beta_v \geq 0 && \forall v \in V[X]
\end{aligned}
$$

We solve this LP with the column generation (CG) method: rather than enumerating $\mathcal{Y}$ explicitly, we maintain $\mathcal{Y}' \subseteq \mathcal{Y}$, which is initially the set of all single-leaf components. We repeatedly solve the *primal LP* using the HiGHS solver [2] over a small subset of columns $\mathcal{Y}'$ (not the whole $\mathcal{Y}$) and then search for a dual constraint indexed by Y (see Equation (1)) that is violated for the current dual solution (α, β). Such a set Y corresponds to a primal *column* (variable) of negative *reduced cost*, defined as $\mathrm{rc}(Y) = 1 - \sum_{x \in Y} \alpha_x + \sum_{v \in V[Y]} \beta_v$. We directly use an algorithm based on [1, Proposition 1] (the rWMAST dynamic program) that finds Y minimizing $\mathrm{rc}(Y)$ in time $\mathcal{O}(n^2)$, and we add Y to the current $\mathcal{Y}'$. We call this procedure *pricing*. We repeat until no dual constraint is violated, which means we have found an optimal solution to the relaxed LP.

3.2 Stabilization and Convergence

We employ several techniques and heuristics to accelerate convergence:

1. **Warm Starting:** Before column generation, we run a short ILS phase and collect the components present in a valid edge cut C. From C, we use components of $T_0 - C \sim T_1 - C$ whose size is at most 10 and add them to the initial $\mathcal{Y}'$.
2. **Wentges Smoothing [6]:** We stabilize the pricing duals using a standard method by Wentges [6]. That is, instead of pricing directly against the raw weights $w_i(u)$ at iteration i (i.e., α_x for a leaf x, or $-\beta_v$ for an internal node v), we price against smoothed weights $w'_i(u)$ obtained from the recurrence $w'_i(u) = \lambda\, w'_{i-1}(u) + (1 - \lambda)\, w_i(u)$, with $w'_0(u) \coloneqq w_0(u)$. Empirically, we set the initial smoothing weight to $\lambda = 0.775$ and decay it by a factor of 0.85 every time smoothed pricing fails to find an improving column (in which case we re-price once at the raw weights $w_i(u)$ before continuing).
3. **Early CG Breaking:** Recall that the input gives numbers a and b such that the task is to provide a solution to MAF with at most $\lfloor a \cdot \mathrm{OPT} + b \rfloor$ components. Suppose that at some point the primal LP objective value is obj, and let κ be the result of (not smoothed) pricing problem. Then, setting $\alpha'_x = \frac{\alpha_x}{\kappa}$ and $\beta'_v = \frac{\beta_v}{\kappa}$ is a valid solution to the relaxed dual with objective value $\frac{\mathrm{obj}}{\kappa}$. Let y denote the number of components in the best solution found during the warm-start phase. If $y \leq \lfloor \frac{a \cdot \mathrm{obj}}{\kappa} + b \rfloor$, then y already meets the lower-bound requirement and we terminate the column generation phase early.
4. **Large Component Penalty:** When rWMAST selects the columns to add to the LP after the pricing phase, we use a scaled reduced cost defined by the formula $\mathrm{rc}'(Y) = \frac{\mathrm{rc}(Y)}{\sqrt{|V[Y]|}}$. This prioritizes columns that describe small subtrees.
5. **Batched Column Management:** We employ the following heuristics directly related to column management:
   a. At each iteration, we remove from $\mathcal{Y}'$ all columns with reduced cost higher than 0.4.
   b. We add multiple columns in batches, all found by rWMAST, as follows: In the first iteration, we add at most 10 columns and linearly increase the per-batch capacity up to 400 columns by the tenth iteration, after which the capacity is kept constant. When adding a batch of columns, we skip any candidate block that would reuse a leaf or internal node already covered more than 8 times by blocks accepted earlier in the same batch. This avoids adding near-duplicate overlapping columns and consequently increases the diversity of the column pool added in one batch.

4 Practical Considerations

Here, we include separate issues that were useful in speeding up the performance of the solver but were not essential to the general description of the algorithm.

4.1 Iteration in Linear Time

The most important detail that allows the algorithm to traverse the search space sufficiently fast is performing each perturbation/repair iteration in linear time with respect to the number of leaves in the trees. The key observation is that whenever a cut, merge, or contraction is performed, there are at most $\mathcal{O}(1)$ new candidates for reductions and, similarly, at most $\mathcal{O}(1)$ new cherries introduced in the forests. We maintain a dynamic set of cherries and reduction candidates throughout the algorithm so that each basic tree operation takes amortized constant time.

4.2 Batched Iterations

The purpose of this part is to speed up computation in Section 2.1. Empirically, the best results on the testing instances are achieved by discarding $(k_{\min} := 5)$–$(k_{\max} := 50)$ edges from the current valid edge cut. For larger instances, this is only a small fraction of the tree size, and the algorithm wastes a lot of time on reapplying similar cuts (as $C \setminus D$ is very similar in many iterations when $|C| \gg |D|$) in each iteration. For this reason, we introduce *batched iterations*, which perform the repair phase (Section 2.2) several times on the same partial edge cut obtained by one perturbation. Efficient rollback is possible as we maintain a stack of the performed operations. Therefore, repeating m repairs on the forest obtained by discarding k edges from a valid edge cut takes roughly $\mathcal{O}(n + m \cdot k)$, compared to m full perturbation/repair iterations, which run in time $\mathcal{O}(m \cdot n)$. This approach slightly decreases the number of perturbations considered within a fixed time limit, but it greatly increases the chance of a successful repair that improves the current best solution. Empirically, we set the batch size to $m = 20$.

References

1 Martin Frohn, Steven Kelk, and Simona Vychytilova. A branch-&-price approach to the unrooted maximum agreement forest problem. *Operations Research Letters*, 63:107364, 2025. doi:10.1016/j.orl.2025.107364.

2 Q. Huangfu and J.A. Julian Hall. Parallelizing the dual revised simplex method. *Mathematical Programming Computation*, 10(1):119–142, 2018. doi:10.1007/s12532-017-0130-5.

3 David Mestel, Steven Chaplick, Steven Kelk, and Ruben Meuwese. Split-or-decompose: Improved FPT branching algorithms for maximum agreement forests. *Journal of Computer and System Sciences*, 160:103800, 2026. doi:10.1016/j.jcss.2026.103800.

4 Neil Olver, Frans Schalekamp, Suzanne van Der Ster, Leen Stougie, and Anke van Zuylen. A duality based 2-approximation algorithm for maximum agreement forest. *Mathematical Programming*, 198(1):811–853, 2023. doi:10.1007/s10107-022-01790-y.

5 PACE Challenge. Parameterized algorithms and computational experiments challenge, 2026. URL: https://pacechallenge.org/.

6 Paul Wentges. Weighted Dantzig–Wolfe decomposition for linear mixed-integer programming. *International Transactions in Operational Research*, 4(2):151–162, 1997. doi:10.1016/S0969-6016(97)00001-4.

7 Chris Whidden, Robert G. Beiko, and Norbert Zeh. Fixed-parameter algorithms for maximum agreement forests. *SIAM Journal on Computing*, 42(4):1431–1466, 2013. doi:10.1137/110845045.