

Parametric maximum closure on precedence forests

VALERIO DOSE, Sapienza Università di Roma, Italy

FABIO FURINI, Sapienza Università di Roma, Italy

MARCO LOCATELLI, Università di Parma, Italy

The *Maximum Closure Problem* asks for a maximum-weight closed subset of a precedence-constrained set of vertices; when vertex weights depend affinely on a scalar parameter λ , as in open-pit mine scheduling, the goal becomes computing the optimal closed set for every value of λ at once. We address this problem when the precedence graph is a directed forest. We present two algorithms (*Peel-and-Contract*, PaC, and its dual DPaC) that compute the full parametric solution – a canonical sequence of disjoint closure layers inducing the nested optimal closures – by iteratively aggregating vertices along precedence arcs, with time-complexity $O(n^2)$ for general forests and space-complexity $O(n)$; heap-based implementations (HPaC, DHPaC) have a worse time-complexity of $O(n^2 \log n)$, but on random instances their computing times empirically grow as $n \log n$. When the forest is restricted to in-trees or out-trees, specialized heap-based variants (HIPaC, HOPaC) achieve time-complexity $O(n \log n)$. We then present a further aggregation-based algorithm (*Rake-and-Compress*, RaC) with time- and space-complexity $O(n \log n)$ also for general forests, which matches the $\Omega(n \log n)$ lower bound for comparison-based algorithms. Computational experiments on random and structured forests with up to 10^5 vertices characterize the practical behavior of the proposed algorithms, identify the topologies on which HPaC and RaC are preferable, and compare HPaC with the public implementation of the fully parametric pseudoflow algorithm, the state of the art for arbitrary precedence graphs, which it outperforms on forests by one to more than three orders of magnitude.

CCS Concepts: • **Theory of computation** → **Graph algorithms analysis**; • **Mathematics of computing** → *Combinatorial optimization*.

Additional Key Words and Phrases: Parametric Maximum Closure Problem, Precedence forests, Combinatorial algorithms, Complexity, Computational study

1 Introduction

In this paper we are interested in the parametric *Maximum Closure Problem* (MCP) with affine costs. We first give a formal description of the problem. Let $G = (V, A)$ be a directed acyclic graph, where the vertex set $V := \{1, 2, \dots, n\}$ has $|V| = n$ vertices, the arc set A has $|A| = m$ arcs, and a directed arc $(i, j) \in A$ establishes that i precedes j , i.e., that inclusion of vertex i implies inclusion of vertex j . We denote with

$$\mathcal{S} = \{S \subseteq V : i \in S \Rightarrow j \in S, (i, j) \in A\},$$

the collection of subsets which fulfill the precedence constraints. It is easy to prove that $\mathcal{S}$ is a lattice when equipped with the intersection ($\cap$) and union ($\cup$) operations, i.e., $S_1, S_2 \in \mathcal{S}$ implies $S_1 \cap S_2, S_1 \cup S_2 \in \mathcal{S}$. To each vertex $i \in V$ we associate an integer profit p_i , possibly negative, and an integer weight $w_i > 0$. For a given subset S of vertices, we denote with $P(S) = \sum_{i \in S} p_i$ its overall profit, and with $W(S) = \sum_{i \in S} w_i$ its overall weight.

Authors' Contact Information: Valerio Dose, Sapienza Università di Roma, Dipartimento di Ingegneria Informatica, Automatica e Gestionale, Rome, Italy, valerio.dose@uniroma1.it; Fabio Furini, Sapienza Università di Roma, Dipartimento di Ingegneria Informatica, Automatica e Gestionale, Rome, Italy, fabio.furini@uniroma1.it; Marco Locatelli, Università di Parma, Dipartimento di Ingegneria e Architettura, Parma, Italy, marco.locatelli@unipr.it.

2026. Manuscript submitted to ACM

Manuscript submitted to ACM

1

We define function $\varphi: (-\infty, +\infty) \rightarrow \mathbb{R}$ as follows

$$\varphi(\lambda) := \max_{S \in \mathcal{S}} \{P(S) - \lambda W(S)\} = \max_{\mathbf{x} \in \{0,1\}^V} \left\{ \sum_{i \in V} (p_i - \lambda w_i) x_i : x_i - x_j \leq 0, (i, j) \in A \right\}. \quad (1)$$

The parametric MCP is the problem of computing the set of optimal solutions $X^*(\lambda)$ of problem (1) as a function of λ . Note that for each λ also the set $X^*(\lambda)$ is a lattice when equipped with the intersection ($\cap$) and union ($\cup$) operations, so that it admits a maximal member (the union of all optimal solutions) and a minimal member (the intersection of all optimal solutions). Function φ is a convex piece-wise linear function with breakpoints

$$\lambda_0 = +\infty > \lambda_1 > \lambda_2 > \dots > \lambda_k > \lambda_{k+1} = -\infty,$$

decreasing for $\lambda < \lambda_1$ and constant (equal to 0) for $\lambda \geq \lambda_1$. It is known (see, e.g., [19, Section 2.5]) that for a monotone sequence of λ values, the set of optimal solutions $X^*(\lambda)$ gives rise to a nested sequence of subsets of vertices $\mathcal{M}_0 \subset \mathcal{M}_1 \subset \mathcal{M}_2 \subset \dots \subset \mathcal{M}_k$, such that $\mathcal{M}_0 = \emptyset$, $\mathcal{M}_k = V$, and $\mathcal{M}_r \in X^*(\lambda)$ for $\lambda \in (\lambda_{r+1}, \lambda_r]$, $r \in \{1, 2, \dots, k\}$. We call this *nestedness property*. The breakpoints λ_r , with $r \in \{1, 2, \dots, k\}$, are λ values such that the set $X^*(\lambda)$ of optimal solutions of problem (1) is *not* a singleton. More precisely, the single optimal solution of problem (1) over the open interval $(\lambda_{i+1}, \lambda_i)$ is the set $\mathcal{M}_i$, while the single optimal solution over the open interval $(\lambda_i, \lambda_{i-1})$ is the set $\mathcal{M}_{i-1}$. For $\lambda = \lambda_i$ the set of optimal solutions is not a singleton and, moreover, as previously mentioned, it is a lattice whose maximal member is $\mathcal{M}_i$, while the minimal one is $\mathcal{M}_{i-1}$. The computation of the nested sequence $\mathcal{M}_1 \subset \mathcal{M}_2 \subset \dots \subset \mathcal{M}_k$ is also known in the literature as *parametrization* (see [19]). It also holds that the sequence of disjoint subsets of vertices defined as follows:

$$\mathcal{I}_1 = \mathcal{M}_1, \mathcal{I}_2 = \mathcal{M}_2 \setminus \mathcal{M}_1, \dots, \mathcal{I}_k = \mathcal{M}_k \setminus \mathcal{M}_{k-1},$$

is strictly decreasing with respect to the profit-to-weight ratio, i.e.:

$$\frac{P_r}{W_r} > \frac{P_{r+1}}{W_{r+1}}, \quad r \in \{1, 2, \dots, k-1\},$$

where

$$P_r := \sum_{i \in \mathcal{I}_r} p_i, \quad W_r := \sum_{i \in \mathcal{I}_r} w_i.$$

The following simple result establishes a relevant property of the subsets $\mathcal{I}_r$, $r \in \{1, 2, \dots, k\}$ (see, e.g., [31, Theorem 9]), stating that each $\mathcal{I}_r$ is a subset with maximum profit-to-weight ratio within the set of vertices which have not been included in the previous subsets $\mathcal{I}_1, \mathcal{I}_2, \dots, \mathcal{I}_{r-1}$.

PROPOSITION 1.1. *For each $r \in \{1, 2, \dots, k\}$, the subset of vertices $\mathcal{I}_r$ is the maximal optimal solution of problem (1) when $\lambda = \lambda_r$ and the set of vertices is restricted to $V \setminus \left[\bigcup_{j=1}^{r-1} \mathcal{I}_j \right]$.*

In what follows we call *closure layers* the subsets $\mathcal{I}_1, \mathcal{I}_2, \dots, \mathcal{I}_k$, and we call the sequence of such subsets an *optimal sequence of closure layers*. The breakpoints of φ are equal to the ratio of the different closure layers, i.e.,

$$\lambda_r = \frac{P_r}{W_r}, \quad r \in \{1, 2, \dots, k\}.$$

For each vertex $v \in V$, we introduce the following definition.

Definition 1.2. The *threshold* of a vertex v , denoted by θ_v , is the breakpoint λ_r such that $v \in \mathcal{M}_r$ but $v \notin \mathcal{M}_{r-1}$, or, equivalently, $v \in \mathcal{I}_r$. In other words, vertex v belongs to the optimal solution of problem (1) for $\lambda \in (\lambda_{r+1}, \lambda_r)$ but not to the one for $\lambda \in (\lambda_r, \lambda_{r-1})$.

We have that the r -th closure layer of the optimal sequence is made up of the vertices which share the same threshold, i.e.,

$$\mathcal{I}_r = \{v \in V : \theta_v = \lambda_r\}.$$

Fig. 1 illustrates the problem and its optimal solutions on a small running instance. We make the following simple observation about minimal time and space requirements for algorithms returning closures and breakpoints.

OBSERVATION 1. *Any comparison-based algorithm computing closures and the related breakpoints requires time $\Omega(n \log n)$. Indeed, even for empty forests, deciding whether the n vertex ratios are pairwise distinct is the element-distinctness problem, which requires $\Omega(n \log n)$ comparisons [7]; the bound therefore does not rely on requiring the breakpoints in sorted order. If an ordered set of breakpoints is required, the task also entails ordering these ratios. Moreover, the minimal space required by an algorithm for MCP, including the space needed for the input of the problem, is at least $\Omega(n)$ since this is the memory required for the profit and weights associated to vertices.*

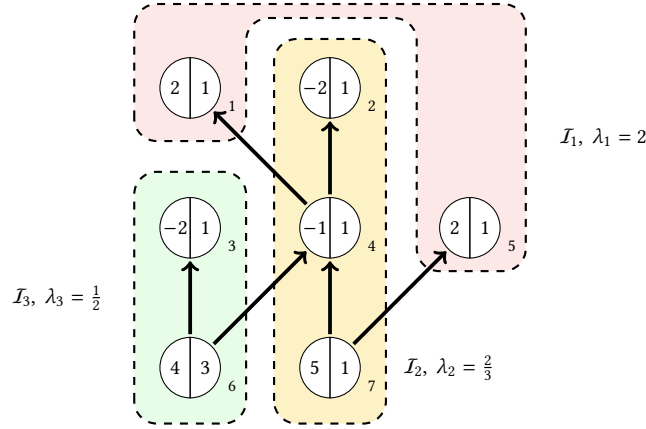


Fig. 1. The optimal solution of the parametric MCP on an example instance with $n = 7$ vertices and $m = 6$ precedence arcs. Each vertex shows $p_i | w_i$, with the number next to it the vertex index i . The dashed regions are the canonical closure-layer partition $\mathcal{I}_1, \mathcal{I}_2, \mathcal{I}_3$, in decreasing order of the ratio $\lambda_r = P(\mathcal{I}_r)/W(\mathcal{I}_r)$; the corresponding nested optimal solutions are $\mathcal{M}_1 = \mathcal{I}_1 = \{1, 5\}$, $\mathcal{M}_2 = \mathcal{I}_1 \cup \mathcal{I}_2 = \{1, 2, 4, 5, 7\}$, $\mathcal{M}_3 = \mathcal{I}_1 \cup \mathcal{I}_2 \cup \mathcal{I}_3 = V$.

1.1 Related work

The MCP was introduced by Lerchs and Grossmann [24] as a model for open-pit mine design. Picard [29] was the first to give the reduction of a general MCP instance to a minimum cut on an extended s - t network, a connection already recognized by Johnson [22] in the bipartite case. The characterization of the parametric solution through profit-to-weight ratios, i.e., the content of Proposition 1.1, goes back to Vallet [31], and Hochbaum and Chen [19] call *parameterization* the computation of the resulting nested family of closed sets.

To compute the function φ and its breakpoints, one can systematically search for its distinct affine components by solving problem (1) for different values of λ . Problem (1) for a fixed λ value is a standard MCP, which can be solved in polynomial time since the constraint matrix of (1) is totally unimodular, and efficient specialized algorithms have been developed. The related literature is extensive, we refer the interested reader to, e.g., [4, 5, 11, 17, 19, 24]. The strategy

based on the multiple solution of standard MCPs is applied, for instance, in the implementation of the algorithms developed by [10] for the Open-Pit Mine Production Scheduling Problem.

Alternative approaches to solve the parametric MCP leverage algorithms designed for the general parametric max-flow/min-cut problem over arbitrary precedence graphs. Gallo, Grigoriadis and Tarjan [16] compute all the breakpoints in $O(nm \log(n^2/m))$ time and list the maximum-ratio closure problem among the applications of their method; the parametric variant of the algorithm of Hochbaum [17] and the parametric pseudoflow algorithm [18] both run in $O(mn \log n)$ time, the latter being the method against which we benchmark in Section 5.5. More recently, Beines et al. [6] proposed an algorithm for the source–sink monotone parametric minimum cut problem which discovers the breakpoints in increasing order of the parameter, with an $O(n^2m)$ worst-case bound and a better practical behavior than previous implementations.

A second line of work originates in single-machine scheduling and linear ordering. Horn [20] computes a subtree of maximum profit-to-weight ratio on rooted trees, a problem solved in $O(n \log n)$ time by Adolphson and Hu [2]. Sidney [30] introduced the decomposition of an instance obtained by iteratively extracting a closed subset of extremal ratio, which in our terminology is the optimal sequence of closure layers, up to splitting some of the layers into pieces of equal ratio, and Margot, Queyranne and Wang [25] use parametric flow algorithms to compute the finest such decomposition. Lawler [23] proved that the underlying scheduling problem is NP-hard under general precedence constraints, and gave an $O(n \log n)$ algorithm for series-parallel precedence digraphs, together with an $O(m^2n \log n)$ procedure computing a single closed set of extremal ratio on an arbitrary digraph. A recent application of this framework comes from blockchain systems: Mollakhani, Wuille and Guo [28] formulate the mempool linearization problem faced by Bitcoin miners as the computation of such a decomposition of a transaction dependency graph.

Closest to the present work are the algorithms exploiting a tree-shaped precedence graph. Galil [15] uses mergeable heaps to compute, for every vertex of a rooted tree, the maximum-ratio subtree rooted at that vertex, in $O(n \log n)$ time overall. Carlson and Eppstein [9] address the parametric problem on rooted trees, solve it in $O(n \log n)$ time and prove that this bound is optimal by a reduction from sorting. Eppstein [13] introduced the name *parametric closure problem* and solved it for several classes of precedence graphs, in $O(n \log n)$ time for series-parallel ones and in $O(n^2 \log n)$ time for *polytrees*, i.e., directed trees, which represent the case considered in this paper (see also Figure 4(a) for an example of polytree, called here mixed-tree).

Finally, the technique of Section 3 borrows its vocabulary from parallel tree contraction [1, 26], where a tree is reduced by repeated *rake* and *compress* operations. The notion of a *cluster* and of its *boundary vertices*, on which our bottom-up phase is built, is the one used to maintain information on dynamic trees: it appears in Frederickson’s topology trees [14] and in the top trees of Alstrup et al. [3].

1.2 Statement of contributions

As previously mentioned, parametric flow algorithms are the state-of-the-art for arbitrary precedence graphs, with a time complexity of $O(mn \log n)$, where m is the number of arcs and n the number of vertices [18]. In this paper, we present new algorithms for computing the optimal sequence of closure layers when the precedence graph G is a *directed forest*, i.e., a disjoint union of directed trees. On forests, where $m = O(n)$, the time-complexity of parametric flow algorithms is $O(n^2 \log n)$, and the same bound is attained by the specialized algorithm of [13] for this class of precedence graphs; the best known time-complexity drops to $O(n \log n)$ only for the restricted classes of *in-trees* and *out-trees*, covered by the algorithms of [9] and [13]. Actually, it has been proven in [18] that over trees the standard MCP can be solved in $O(n)$. Equivalently, the evaluation of function φ for a fixed λ value, can be done in $O(n)$ time. This fact,

together with the $O(n)$ bound on the number of breakpoints of the function φ , suggests that an $O(n^2)$ algorithm for trees is possible, but, to the authors' knowledge, such an algorithm has not been presented in the literature yet.

Our first contribution in Section 2, is to provide two distinct algorithms based on vertex aggregation, which compute an optimal sequence of closure layers with time-complexity $O(n^2)$ and space-complexity $O(n)$ (which is optimal according to Observation 1) for general forests. The algorithms are based on an iterative aggregation of vertices. We also observe that a heap-based variant of these algorithms has a worse time-complexity but a better average behavior, with average computing times increasing as $n \log n$, as shown in Section 5. We further show that when the forest is composed of *out-trees* or *in-trees*, close variants of the proposed algorithms reduce the complexity to $O(n \log n)$.

Motivated by the practical behavior of the heap-based variant of the proposed algorithms, with computing times increasing as $n \log n$, we further present, in Section 3, a new algorithm (RaC), also based on a vertex aggregation strategy, with time- and space-complexity $O(n \log n)$ also in the case of general forests. Note that, according to Observation 1, this is an optimal time-complexity. The development of this algorithm involved the use of generative AI, as detailed at the beginning of Section 3.

We also performed extensive computational experiments which are discussed in Section 5. As previously mentioned, the heap-based variant of the algorithms presented in Section 2 works quite well in practice and, according to the computational experiments, performs better than RaC over random instances and over some special structured trees (binary and path trees), but much worse over other special structured trees (star trees), which is fully in accordance with the different time-complexity of the algorithms.

2 Aggregation-based algorithms for computing the optimal sequence of closure layers on directed forests

2.1 Notation and definitions

Definition 2.1. For each vertex $j \in V$, we denote with $N^+(j)$ the set of *successors* of j , while we denote with $N^-(j)$ the set of its *predecessors*. Similarly, for a subset M of vertices, we denote with $N^+(M) = \cup_{j \in M} N^+(j) \setminus M$ the set of successors of the subset M , and with $N^-(M) = \cup_{j \in M} N^-(j) \setminus M$ the set of its predecessors.

Definition 2.2. Let $G = (V, A)$ be a directed forest and let $j \in V$ be a vertex. The *preceding set* of vertex j is the set of vertices which can be reached from j with a directed path, possibly with length 0, which is the set

$$F_j = \{k \in V : \text{there exists a directed path from } j \text{ to } k\}.$$

Let $a = (i, j) \in A$ be an arc. The *minimal preceding set* of arc $a = (i, j)$ is the set of vertices which are in the preceding set of i but not in the preceding set of j , namely the set

$$F_a := F_i \setminus F_j.$$

Definition 2.3. Let $G = (V, A)$ be a directed forest and let $j \in V$ be a vertex. The *succeeding set* of vertex j is the set of vertices which can reach j with a directed path, possibly with length 0, which is the set

$$B_j = \{k \in V : \text{there exists a directed path from } k \text{ to } j\}.$$

Let $a = (i, j) \in A$ be an arc. The *minimal succeeding set* of arc $a = (i, j)$ is the set of vertices which are in the succeeding set of j but not in the succeeding set of i , namely the set

$$B_a := B_j \setminus B_i.$$

Definition 2.4. We indicate with V_{fin} the set of the *final vertices* of G , and with V_{init} the set of its *initial vertices*, which are defined respectively by

$$V_{\text{fin}} := \{j \in V : N^+(j) = \emptyset\}, \quad V_{\text{init}} := \{j \in V : N^-(j) = \emptyset\}.$$

Definition 2.5. Let $G = (V, A)$ be a forest and let $\mathcal{M}' \subset \mathcal{M} \subset V$ be a chain of subsets of vertices. The set of arcs connecting vertices in $\mathcal{M}$ is the set

$$A(\mathcal{M}) := \{(i, j) \in A : i, j \in \mathcal{M}\}.$$

The set of vertices in $\mathcal{M}$ which are connected with $\mathcal{M}'$ by an arc exiting $\mathcal{M}'$ is the set

$$\mathcal{M}^+(\mathcal{M}') := \{j \in \mathcal{M} \setminus \mathcal{M}' : \exists i \in \mathcal{M}', (i, j) \in A\},$$

and the set of vertices in $\mathcal{M}$ which are connected with $\mathcal{M}'$ by an arc entering $\mathcal{M}'$ is the set

$$\mathcal{M}^-(\mathcal{M}') := \{i \in \mathcal{M} \setminus \mathcal{M}' : \exists j \in \mathcal{M}', (i, j) \in A\}.$$

An illustration of preceding/succeeding sets associated to vertices and arcs, and of final/initial vertices, can be found in Fig. 2(a), while an illustration of the sets $\mathcal{M}^+(\mathcal{M}')$ and $\mathcal{M}^-(\mathcal{M}')$ is in Fig. 2(b).

2.2 Main algorithm to compute the optimal sequence of closure layers on a forest

First, we need to present a simple ratio property for disjoint sets, used later to compare unions of candidate closure layers. The proof is reported for completeness.

PROPOSITION 2.6. *Let $M_1, M_2 \subset V$ be nonempty. If M_1 and M_2 are disjoint, i.e., $M_1 \cap M_2 = \emptyset$, then it cannot hold that*

$$\frac{P(M_1 \cup M_2)}{W(M_1 \cup M_2)} > \frac{P(M_1)}{W(M_1)}, \frac{P(M_2)}{W(M_2)} \quad \text{or} \quad \frac{P(M_1 \cup M_2)}{W(M_1 \cup M_2)} < \frac{P(M_1)}{W(M_1)}, \frac{P(M_2)}{W(M_2)}. \quad (2)$$

Moreover,

$$\frac{P(M_1 \cup M_2)}{W(M_1 \cup M_2)} \geq (\leq) \frac{P(M_1)}{W(M_1)}, \frac{P(M_2)}{W(M_2)} \Rightarrow \frac{P(M_1 \cup M_2)}{W(M_1 \cup M_2)} = \frac{P(M_1)}{W(M_1)} = \frac{P(M_2)}{W(M_2)}. \quad (3)$$

PROOF. Let us assume by contradiction that the first inequality in (2) holds (the proof for the second inequality is analogous). Then, since $M_1 \cap M_2 = \emptyset$:

$$\frac{P(M_1 \cup M_2)}{W(M_1 \cup M_2)} = \frac{P(M_1) + P(M_2)}{W(M_1) + W(M_2)} > \frac{P(M_1)}{W(M_1)}, \frac{P(M_2)}{W(M_2)}.$$

But

$$\frac{P(M_1) + P(M_2)}{W(M_1) + W(M_2)} > \frac{P(M_1)}{W(M_1)}, \frac{P(M_2)}{W(M_2)} \Rightarrow \frac{P(M_2)}{W(M_2)} > \frac{P(M_1)}{W(M_1)} \text{ and } \frac{P(M_2)}{W(M_2)} < \frac{P(M_1)}{W(M_1)},$$

which is not possible. In a completely similar way we can also prove (3). $\square$

We also make a couple of remarks.

Remark 1. Let $\mathcal{T} = (\mathcal{I}_1, \mathcal{I}_2, \dots, \mathcal{I}_{k(\mathcal{T})})$ be the optimal sequence of closure layers. Then, for every vertex $j \in V$, all vertices in F_j are contained in closure layers of the sequence $\mathcal{T}$ that precede the closure layer containing j , or are

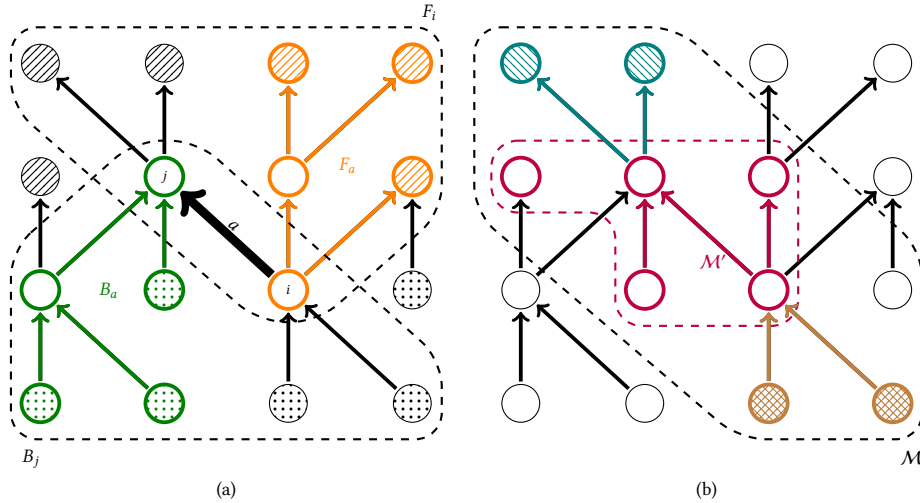


Fig. 2. (a) Representation of a directed tree. The arc $a = (i, j)$ is highlighted in bold. Two dashed lines enclose the sets F_i and B_j . The vertices in F_a , and their connecting arcs, are highlighted in orange, while the same is done in green for B_a . Final vertices (i.e., vertices v with $N^+(v) = \emptyset$) are filled with a north-east line pattern, while initial vertices (i.e., vertices v with $N^-(v) = \emptyset$) are filled with a dot pattern. (b) Directed graph with dashed set $\mathcal{M}$ and highlighted subset $\mathcal{M}'$ in purple. Vertices in $\mathcal{M}^+(\mathcal{M}')$ are highlighted in teal with a north-west line pattern, and the arcs from $\mathcal{M}'$ to $\mathcal{M}^+(\mathcal{M}')$ are highlighted in teal. Vertices in $\mathcal{M}^-(\mathcal{M}')$ are highlighted in brown with a crosshatch pattern, and the arcs from $\mathcal{M}^-(\mathcal{M}')$ to $\mathcal{M}'$ are highlighted in brown.

contained in the same closure layer. Furthermore, given an arc $a = (i, j)$, vertex i can be in the same closure layer as vertex j only if all vertices in F_a are contained in a closure layer of the sequence $\mathcal{T}$ that precedes the closure layer containing j , or are contained in the same closure layer.

Remark 2. Notice that $N^+(F_j) = \{k \in V \setminus F_j : \exists i \in F_j, (i, k) \in A\} = \emptyset$ for every $j \in V$ by Definition 2.2. Furthermore $N^+(F_{(i,j)} \setminus \{i\}) = \emptyset$ for every $(i, j) \in A$ since $F_{(i,j)} \setminus \{i\}$ is the disjoint union of the sets F_k with $k \in N^+(i) \setminus \{j\}$.

To show that our main algorithm outputs the optimal sequence of closure layers we need to prove three propositions.

PROPOSITION 2.7. *Suppose $G = (V, A)$ is a forest and let $\mathcal{M} \subset V$ be a subset of vertices. If $N^+(\mathcal{M}) = \emptyset$, then*

$$\mathcal{M} = \left(\bigsqcup_{a \in \tilde{A}} F_a \right) \sqcup \tilde{V}_{\text{fin}}$$

for some subset $\tilde{A} \subset A$ and $\tilde{V}_{\text{fin}} \subset V_{\text{fin}}$, i.e., $\mathcal{M}$ is the disjoint union of the minimal preceding sets associated to some arcs and some subset of final vertices.

PROOF. We can show this constructively. We start with $\tilde{A} = \emptyset$ and $\tilde{V}_{\text{fin}} = \emptyset$. Then, we add one final vertex in $V_{\text{fin}} \cap \mathcal{M}$ to $\tilde{V}_{\text{fin}}$ for every connected component of the graph $(\mathcal{M}, A(\mathcal{M}))$. Such final vertices exist since $N^+(\mathcal{M}) = \emptyset$ and G is a forest. Thus we obtain a first set $\mathcal{M}' \subset \mathcal{M}$ defined by

$$\mathcal{M}' = \tilde{V}_{\text{fin}}$$

and we have $N^+(\mathcal{M}') = \emptyset$. Now if $\mathcal{M}^-(\mathcal{M}') \neq \emptyset$ (i.e., $\mathcal{M} \neq \mathcal{M}'$), we consider any item $j \in \mathcal{M}'$ which is connected to some item in $\mathcal{M}^-(\mathcal{M}')$. Then we take any maximal backwards path $\pi_j \subset A(\mathcal{M}) \setminus A(\mathcal{M}')$ from j to some item $v(j)$

with $\mathcal{M}^-(\{v(j)\}) = \emptyset$, and we add the arcs in π_j to $\tilde{\mathcal{A}}$, updating $\mathcal{M}'$ accordingly. Since the new elements added to $\tilde{\mathcal{A}}$ are chosen in a backwards path in $\mathcal{A}(\mathcal{M}) \setminus \mathcal{A}(\mathcal{M}')$, we have $F_a \cap F_{a'} = \emptyset$, for every $a, a' \in \tilde{\mathcal{A}}$ with $a \neq a'$. Thus we obtain

$$\mathcal{M}' = \left(\bigsqcup_{a \in \tilde{\mathcal{A}}} F_a \right) \sqcup \tilde{V}_{\text{fin}}.$$

and we have enlarged the set $\mathcal{M}'$ to also include $F_{v(j)}$. Hence we have again $\mathcal{N}^+(\mathcal{M}') = \emptyset$.

Then, we repeat the last step until $\mathcal{M}' = \mathcal{M}$. □

An illustration of the outcome of the constructive procedure described in the proof of Proposition 2.7 can be found in Fig. 3.

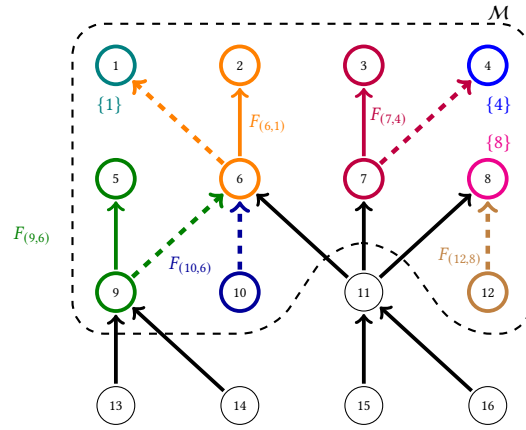


Fig. 3. Constructive decomposition of a set $\mathcal{M}$ with $\mathcal{N}^+(\mathcal{M}) = \emptyset$ according to the proof of Proposition 2.7. We start from one final vertex per connected component in $\mathcal{M}$, i.e., $\mathcal{M}' = \tilde{V}_{\text{fin}} = \{1, 4, 8\}$, and then iteratively add sets F_a associated with arcs a from $\mathcal{M}^-(\mathcal{M}')$ to $\mathcal{M}'$ (shown as dashed colored arcs), until $\mathcal{M}' = \mathcal{M}$. The resulting disjoint union is $\mathcal{M} = \{1, 4, 8\} \sqcup F_{(6,1)} \sqcup F_{(7,4)} \sqcup F_{(12,8)} \sqcup F_{(9,6)} \sqcup F_{(10,6)}$.

PROPOSITION 2.8. Suppose $G = (V, A)$ is a forest and let $a^* \in \arg \max_{a \in A} \frac{P(F_a)}{W(F_a)}$ with $a^* = (u, v)$. If $\frac{P(F_{a^*})}{W(F_{a^*})} \geq \frac{pf}{wf}$ for every $f \in V_{\text{fin}}$, then the two vertices u and v belong to the same closure layer in the optimal sequence.

PROOF. Let $\mathcal{T} = (\mathcal{I}_1, \mathcal{I}_2, \dots, \mathcal{I}_{k(\mathcal{T})})$ be the optimal sequence of closure layers and let $t(v) \in \{1, 2, \dots, k(\mathcal{T})\}$ be the index of the closure layer containing v , i.e., $v \in \mathcal{I}_{t(v)}$. Consider the set of vertices

$$\mathcal{M} := \bigcup_{r=1}^{t(v)} \mathcal{I}_r.$$

Notice that, in view of Remark 1, $\mathcal{N}^+(\mathcal{M}) = \emptyset$. By Proposition 2.7 we have

$$\mathcal{M} = \left(\bigsqcup_{a \in \tilde{\mathcal{A}}} F_a \right) \sqcup \tilde{V}_{\text{fin}} \tag{4}$$

for some $\tilde{\mathcal{A}} \subset A$ and $\tilde{V}_{\text{fin}} \subset V_{\text{fin}}$.

Suppose by contradiction that $u \notin \mathcal{I}_{t(v)}$ and thus, in view of Remark 1, $u \notin \mathcal{M}$ since $(u, v) \in A$. Hence, by Remark 2 we have $N^+(F_{a^*} \cap \mathcal{M}) = \emptyset$. Thus, by Proposition 2.7 we have

$$F_{a^*} \cap \mathcal{M} = \left(\bigsqcup_{a \in A^*} F_a \right) \sqcup V_{\text{fin}}^* \quad (5)$$

for some $A^* \subset A$ and $V_{\text{fin}}^* \subset V_{\text{fin}}$.

We have

$$\frac{P(F_{a^*} \setminus \mathcal{M})}{W(F_{a^*} \setminus \mathcal{M})} \geq \frac{P(F_{a^*})}{W(F_{a^*})} \geq \frac{P(\mathcal{M})}{W(\mathcal{M})} \geq \frac{P(\mathcal{I}_{t(v)})}{W(\mathcal{I}_{t(v)})}$$

where in the last inequality we used the fact that $\mathcal{T} = (\mathcal{I}_1, \mathcal{I}_2, \dots, \mathcal{I}_k(\mathcal{T}))$ is a decreasing sequence of closure layers. The second inequality follows from: (i) the fact that $\frac{P(F_{a^*})}{W(F_{a^*})} \geq \frac{P(F_a)}{W(F_a)}$ for every $a \in A$ and $\frac{P(F_{a^*})}{W(F_{a^*})} \geq \frac{P_f}{w_f}$ for every $f \in V_{\text{fin}}$; (ii) Eq. (4); (iii) Proposition 2.6. The first inequality follows again from the same properties of F_{a^*} , from Eq. (5), and from Proposition 2.6. This chain of inequalities leads to a contradiction. Indeed, $\frac{P(F_{a^*} \setminus \mathcal{M})}{W(F_{a^*} \setminus \mathcal{M})} \geq \frac{P(\mathcal{I}_{t(v)})}{W(\mathcal{I}_{t(v)})}$ and since $F_{a^*} \setminus \mathcal{M}$ and $\mathcal{I}_{t(v)}$ are disjoint, the ratio of their union is not lower than the ratio of $\mathcal{I}_{t(v)}$, by Proposition 2.6. Observing that $N^+(F_{a^*} \cup \mathcal{M}) = \emptyset$, we also have that $\mathcal{I}_{t(v)} \cup (F_{a^*} \setminus \mathcal{M})$ is a feasible closure layer after $\mathcal{I}_1, \mathcal{I}_2, \dots, \mathcal{I}_{t(v)-1}$. This contradicts the fact that $\mathcal{I}_{t(v)}$ is the maximal optimal solution of problem (1) when $\lambda = \lambda_{t(v)}$ and the set of vertices is restricted to $V \setminus \left[\bigcup_{j=1}^{t(v)-1} \mathcal{I}_j \right]$ (see Proposition 1.1). $\square$

Proposition 2.8 allows us, by collapsing the arc a^* , to reduce the problem of finding the optimal sequence of closure layers to a smaller graph.

PROPOSITION 2.9. *Suppose $G = (V, A)$ is a forest and let $g \in \mathcal{B}_{\text{fin}} := \arg \max_{f \in V_{\text{fin}}} \frac{P_f}{w_f}$. If $\frac{P_g}{w_g} > \frac{P(F_a)}{W(F_a)}$ for every $a \in A$, then $\mathcal{B}_{\text{fin}}$ is the first closure layer in the optimal sequence of closure layers.*

PROOF. In view of Remark 1, any candidate subset $\mathcal{M} \subset V$ to be the first closure layer in the optimal sequence must satisfy $N^+(\mathcal{M}) = \emptyset$. By Proposition 2.7 we have

$$\mathcal{M} = \left(\bigsqcup_{a \in \tilde{A}} F_a \right) \sqcup \tilde{V}_{\text{fin}}$$

for some subsets $\tilde{A} \subset A$, $\tilde{V}_{\text{fin}} \subset V_{\text{fin}}$. Since $g \in \mathcal{B}_{\text{fin}} = \arg \max_{f \in V_{\text{fin}}} \frac{P_f}{w_f}$ and $\frac{P_g}{w_g} > \frac{P(F_a)}{W(F_a)}$ for every $a \in A$, by Proposition 2.6 the ratio associated to the set $\mathcal{B}_{\text{fin}}$ is larger than or equal to the ratio associated to $\mathcal{M}$, with the equality possible only if $\tilde{A} = \emptyset$. This implies that $\mathcal{B}_{\text{fin}}$ is the maximal subset of V among those with largest ratio, i.e., $\mathcal{B}_{\text{fin}} = \mathcal{I}_1$. $\square$

Now we are ready to describe our first algorithm, based on aggregation of vertices. In Algorithm 1, called *Peel-and-Contract* (PaC) in what follows, the initialization (lines 3-7) creates a copy $(\bar{V}, \bar{A})$ of the graph and associates to each copied vertex j a set of vertices $R_j \subset V$ in the original graph; then, in the main cycle (lines 8-20) at each iteration we either find a new closure layer in the optimal sequence (line 15) as prescribed by Proposition 2.9, and we remove it from the graph $(\bar{V}, \bar{A})$, or we “contract” an arc joining two adjacent vertices according to Proposition 2.8 (line 18). Throughout the description of PaC and of its variants, the sets $N^+(\cdot)$, $N^-(\cdot)$, F_j , F_a and the totals $P(\cdot)$, $W(\cdot)$ are understood with respect to the current working graph $(\bar{V}, \bar{A})$ and to the current aggregated profits $\bar{p}_j$ and weights $\bar{w}_j$; we write $\bar{N}^+(\cdot)$ and $\bar{N}^-(\cdot)$ for the successors and predecessors in $\bar{A}$ when the distinction matters.

In particular, in procedure **FINDBESTWING** (Algorithm 2) we determine the arc $a^* = (u, v) \in \bar{A}$ whose associated set F_{a^*} is the one with the largest ratio. Then, in procedure **FINDBESTFINALVERTICES** (Algorithm 3) we determine the set $\mathcal{B}_{\text{fin}}$ of the final vertices of $\bar{V}$ with the largest ratio. Now, if the largest determined ratio is achieved only by

the final vertices in $\mathcal{B}_{\text{fin}}$, then the subset of vertices $\bigcup_{f \in \mathcal{B}_{\text{fin}}} R_f \subset V$ is the next closure layer in the optimal sequence by Proposition 2.9. Hence, procedure REMOVEFINALVERTICES (Algorithm 4) adds to the optimal sequence the new closure layer, and removes the vertices in $\mathcal{B}_{\text{fin}}$ together with the corresponding arcs from the graph $(\bar{V}, \bar{A})$. Conversely, if the largest determined ratio is achieved by (or also by) a set $F_{a^*} \subset \bar{V}$ associated to some arc $a^* \in \bar{A}$, procedure CONTRACTARC (Algorithm 5) removes arc $a^* = (u, v)$ from $\bar{A}$ according to Proposition 2.8, joining the two vertices u and v and the corresponding sets R_u, R_v of associated vertices in V . An illustration of a full run of PaC on a small instance is reported in Appendix A.

Algorithm 1 Algorithm PaC to compute the optimal sequence of closure layers on a forest

```

1: input  $V, A, V_{\text{fin}}, p, w$ 
2:  $\triangleright$  Initialization:
3:  $\bar{V} = V, \bar{A} = A, \bar{V}_{\text{fin}} = V_{\text{fin}}$ 
4: for  $j \in \bar{V}$  do
5:   Set  $\bar{p}_j = p_j, \bar{w}_j = w_j$  profit and weight for vertex  $j \in \bar{V}; R_j = \{j\}$ 
6: end for
7:  $k = 0$ 
8: while  $\bar{V} \neq \emptyset$  do
9:    $\triangleright$  Determine  $a^* = (u, v)$  from Proposition 2.8 and its ratio  $\frac{P(F_{a^*})}{W(F_{a^*})}$ :
10:   $(u, v) \leftarrow \text{FINDBESTWING}(\bar{V}, \bar{A})$ 
11:    $\triangleright$  Determine the set  $\mathcal{B}_{\text{fin}}$  from Proposition 2.9 and the ratio  $\frac{\bar{p}_g}{\bar{w}_g}$ :
12:   $(g, \mathcal{B}_{\text{fin}}) \leftarrow \text{FINDBESTFINALVERTICES}(\bar{V}_{\text{fin}})$ 
13:    $\triangleright$  If largest ratio is achieved by some final vertices (vertices in  $\mathcal{B}_{\text{fin}}$ ), we get a new closure layer:
14:   if  $\frac{\bar{p}_g}{\bar{w}_g} > \frac{P(F_{(u,v)})}{W(F_{(u,v)})}$  then
15:     REMOVEFINALVERTICES( $\mathcal{B}_{\text{fin}}$ )
16:   else
17:      $\triangleright$  If largest ratio is achieved by (or also by)  $F_{a^*}$  we “contract” the arc  $a^*$ :
18:     CONTRACTARC( $u, v$ )
19:   end if
20: end while
21: return  $\mathcal{T} = (I_1, I_2, \dots, I_k)$ 

```

Algorithm 2 Sub-procedure FINDBESTWING: determine arc $a^* = (u, v)$ with largest ratio $\frac{P(F_{a^*})}{W(F_{a^*})}$

```

1: procedure FINDBESTWING( $\bar{V}, \bar{A}$ )
2:   Compute  $P(F_j), W(F_j)$  for every  $j \in \bar{V}$ 
3:    $u = 0, v = 0, P(F_{(0,0)}) = -\infty, W(F_{(0,0)}) = 1$ 
4:   for  $a = (i, j) \in \bar{A}$  do
5:      $P(F_a) = P(F_i) - P(F_j), W(F_a) = W(F_i) - W(F_j)$ 
6:     if  $\frac{P(F_a)}{W(F_a)} > \frac{P(F_{(u,v)})}{W(F_{(u,v)})}$  then  $u = i, v = j$ 
7:     end if
8:   end for
9:   return  $(u, v)$ 
10: end procedure

```

THEOREM 2.10. PaC computes the optimal sequence of closure layers on a forest in $O(n^2)$ time, while space-complexity is $O(n)$.

PROOF. At each iteration, either Proposition 2.9 identifies the next closure layer and removes it, or Proposition 2.8 identifies an arc whose endpoints must belong to the same closure layer and contracts it. These operations preserve the optimal sequence of the original instance, after replacing each contracted vertex by the associated set R_j . Since each iteration decreases the cardinality of $\tilde{V}$ by at least one, there are at most n iterations. In a forest, completing the four procedures in Algorithms 2 to 5 can be done in $O(n)$ time. This is rather obvious for the last three procedures, each involving one or two for cycles with at most n iterations, and with a constant number of operations per iteration. Concerning the procedure in Algorithm 2, it also takes $O(n)$ by a recursive computation of the values $P(F_j)$ and $W(F_j)$ starting from the final vertices of the graph, up to the initial ones. Hence, the (at most) n iterations, with a computational cost per iteration equal to $O(n)$, require a number of operations which is $O(n^2)$.

Concerning space, we notice that at each iteration the algorithm keeps track of the different sets of aggregated vertices and of the related profits and weights, which requires $O(n)$ space, $\square$

Algorithm 3 Sub-procedure **FINDBESTFINALVERTICES**: determine set $\mathcal{B}_{\text{fin}}$ of final vertices with largest ratio $\frac{\hat{p}_g}{\hat{w}_g}$

```

1: procedure FINDBESTFINALVERTICES( $\tilde{V}_{\text{fin}}$ )
2:    $g = 0, \hat{p}_0 = -\infty, \hat{w}_0 = 1, \mathcal{B}_{\text{fin}} = \emptyset$ 
3:   for  $f \in \tilde{V}_{\text{fin}}$  do
4:     if  $\frac{\hat{p}_f}{\hat{w}_f} > \frac{\hat{p}_g}{\hat{w}_g}$  then  $g = f, \mathcal{B}_{\text{fin}} = \{f\}$ 
5:     else if  $\frac{\hat{p}_f}{\hat{w}_f} = \frac{\hat{p}_g}{\hat{w}_g}$  then  $\mathcal{B}_{\text{fin}} = \mathcal{B}_{\text{fin}} \cup \{f\}$ 
6:     end if
7:   end for
8:   return  $(g, \mathcal{B}_{\text{fin}})$ 
9: end procedure

```

Remark 3. We remark that the proposed approach does not suffer from numerical precision problems. Indeed, comparisons between ratios of integers can be obviously done by comparing integer numbers, while all other operations only involve integer values. This is a notable difference with respect, e.g., to the approach based on pseudo-flow computations. Indeed, such approach might be unable to recognize two distinct closure layers when the difference between their ratios is below a given numerical precision, as discussed in Section 5.

2.3 Dual Variant

PaC finds the closure layers in the optimal sequence in decreasing order with respect to their ratio. It is possible to devise an analogous variant of PaC, which finds the closure layers in the optimal sequence in increasing order through an iterative aggregation of vertices. In this variant, instead of computing the ratio of final vertices and preceding sets, we need to compute the ratio of *initial vertices* and *succeeding sets*, namely the sets V_{init}, B_j and B_a already introduced in Section 2.1.

We have the following three propositions, whose proofs are analogous to those of Propositions 2.7 to 2.9, and we omit them in this paper.

Algorithm 4 Sub-procedure REMOVEFINALVERTICES: create next closure layer from $\mathcal{B}_{\text{fin}}$ and update the graph

```

1: procedure REMOVEFINALVERTICES( $\mathcal{B}_{\text{fin}}$ )
2:    $k = k + 1$ ,  $\mathcal{I}_k = \bigcup_{f \in \mathcal{B}_{\text{fin}}} R_f$ ,  $U = \emptyset$ 
3:   for  $i \in \tilde{N}^-(\mathcal{B}_{\text{fin}})$  do
4:      $U = U \cup \{i\}$ 
5:   end for
6:   for  $(i, f) \in \tilde{A}$  with  $f \in \mathcal{B}_{\text{fin}}$  do
7:      $\tilde{A} = \tilde{A} \setminus \{(i, f)\}$ 
8:   end for
9:   for  $i \in U$  do
10:    if  $\tilde{N}^+(i) = \emptyset$  then  $\tilde{V}_{\text{fin}} = \tilde{V}_{\text{fin}} \cup \{i\}$ 
11:    end if
12:   end for
13:    $\tilde{V}_{\text{fin}} = \tilde{V}_{\text{fin}} \setminus \mathcal{B}_{\text{fin}}$ ,  $\tilde{V} = \tilde{V} \setminus \mathcal{B}_{\text{fin}}$ 
14: end procedure

```

Algorithm 5 Sub-procedure CONTRACTARC: contract arc $a^* = (u, v)$ and merge the associated vertex sets

```

1: procedure CONTRACTARC( $u, v$ )
2:   for  $i \in \tilde{N}^-(u)$  do  $\tilde{A} = \tilde{A} \setminus \{(i, u)\}$ ,  $\tilde{A} = \tilde{A} \cup \{(i, v)\}$ 
3:   end for
4:   for  $j \in \tilde{N}^+(u) \setminus \{v\}$  do  $\tilde{A} = \tilde{A} \setminus \{(u, j)\}$ ,  $\tilde{A} = \tilde{A} \cup \{(v, j)\}$ 
5:   if  $v \in \tilde{V}_{\text{fin}}$  then
6:      $\tilde{V}_{\text{fin}} = \tilde{V}_{\text{fin}} \setminus \{v\}$ 
7:   end if
8:   end for
9:    $R_v = R_u \cup R_v$ ,  $\tilde{p}_v = \tilde{p}_u + \tilde{p}_v$ ,  $\tilde{w}_v = \tilde{w}_u + \tilde{w}_v$ 
10:   $\tilde{A} = \tilde{A} \setminus \{(u, v)\}$ ,  $\tilde{V} = \tilde{V} \setminus \{u\}$ 
11: end procedure

```

PROPOSITION 2.11. Suppose $G = (V, A)$ is a forest and let $\mathcal{M} \subset V$ be a subset of vertices. If $\tilde{N}^-(\mathcal{M}) = \emptyset$, then

$$\mathcal{M} = \left(\bigsqcup_{a \in \tilde{A}} B_a \right) \sqcup \tilde{V}_{\text{init}}$$

for some subset $\tilde{A} \subset A$ and $\tilde{V}_{\text{init}} \subset V_{\text{init}}$, i.e., $\mathcal{M}$ is the disjoint union of the minimal succeeding sets associated to some arcs and some subset of initial vertices.

PROPOSITION 2.12. Suppose $G = (V, A)$ is a forest and let $a^* \in \arg \min_{a \in A} \frac{P(B_a)}{W(B_a)}$ with $a^* = (u, v)$. If $\frac{P(B_{a^*})}{W(B_{a^*})} \leq \frac{p_i}{w_i}$ for every $i \in V_{\text{init}}$, then the two vertices u and v belong to the same closure layer in the optimal sequence.

PROPOSITION 2.13. Suppose $G = (V, A)$ is a forest and let $g \in \mathcal{B}_{\text{init}} := \arg \min_{i \in V_{\text{init}}} \frac{p_i}{w_i}$. If $\frac{p_g}{w_g} < \frac{P(B_a)}{W(B_a)}$ for every $a \in A$, then $\mathcal{B}_{\text{init}}$ is the last closure layer in the optimal sequence of closure layers.

Thanks to these results, PaC can be converted to find the closure layers in the optimal sequence in increasing order with respect to their ratio. We refer to this algorithm as the *dual variant* of PaC, and denote it with DPaC. An illustration of a full run of this dual variant is reported in Appendix B.

We also have the same complexity of $\mathcal{O}(n)$ per iteration and $\mathcal{O}(n^2)$ total complexity, for which we give only the statement, since the proof is analogous to the one of Theorem 2.10.

THEOREM 2.14. *DPaC computes the optimal sequence of closure layers on a forest with time-complexity $O(n^2)$ and space-complexity $O(n)$.*

Remark 4. One can also combine the two approaches in a single iteration to halve the number of total iterations, but at the cost of increasing (doubling) the number of operations per iteration.

2.4 Improved variants with all out-trees or all in-trees

If we know that the forest graph G is composed of subtrees whose arcs all point in the same direction, we can employ a heap data structure in PaC and obtain a better total complexity of $O(n \log n)$. To describe this variant we first give the definition of out-tree and in-tree.

Definition 2.15. Let $G = (V, A)$ be a connected tree. We say that G is an *in-tree* if for every vertex $j \in V$ we have $|N^+(j)| \leq 1$. As a consequence, there is only one vertex j_0 with $N^+(j_0) = \emptyset$, which we call the *root* of the in-tree.

Definition 2.16. Let $G = (V, A)$ be a connected tree. We say that G is an *out-tree* if for every vertex $j \in V$ we have $|N^-(j)| \leq 1$. As a consequence, there is only one vertex j_0 with $N^-(j_0) = \emptyset$, which we call the *root* of the out-tree.

Fig. 4 illustrates these two restricted topologies against a general (mixed) tree, where neither restriction holds.

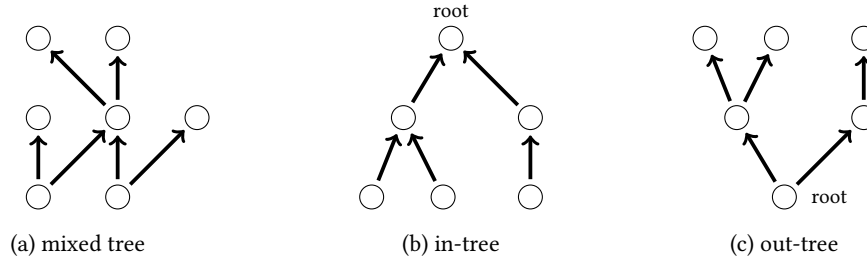


Fig. 4. Three directed trees (arrows point from a vertex to its prerequisite). (a) A general (mixed) tree: a vertex may have several predecessors and several successors at once, so arcs may point in any direction; it is neither an in-tree nor an out-tree. (b) An in-tree (Definition 2.15): every vertex has at most one successor. (c) An out-tree (Definition 2.16): every vertex has at most one predecessor. A forest is a disjoint union of such trees.

Algorithm 6, denoted in what follows with HIPaC, is devised for in-trees. In the initialization (lines 3-7) we create a copy of the set of vertices and we insert the elements in a heap with respect to their ratio. Then in the main cycle (lines 8-26), at each iteration we extract the vertex g at the top of the heap. If vertex g is a final vertex, we add it to the current (or to a new) closure layer (lines 11-17), we delete it from the graph and we update the heap accordingly. If vertex g is not a final vertex, we contract the only arc exiting g and we update the heap accordingly.

THEOREM 2.17. *If G is a forest of in-trees, HIPaC computes the optimal sequence of closure layers in $O(n \log n)$ time.*

PROOF. We assume that the heap is a Fibonacci one (see, e.g., [8, Section 1.1]). Building the heap requires $O(n)$ operations. Next, at each iteration of the main cycle, we first remove the element with largest value from the heap, which requires $O(\log n)$. Next, when we are contracting an arc, we need to update the value of a single element in the heap. This, again, requires $O(\log n)$ operations. Since the number of iterations is $O(n)$, we can conclude that the overall complexity of Algorithm 6 is $O(n \log n)$. $\square$

Algorithm 6 Improved variant of PaC, denoted with HIPaC, to compute the optimal sequence of closure layers on a forest of in-trees

```

1: input  $V, A, V_{\text{fin}}, p, w$ 
2:  $\triangleright$  Initialization:
3:  $\bar{V} = V, \bar{A} = A, \bar{V}_{\text{fin}} = V_{\text{fin}}$ 
4: for  $j \in \bar{V}$  do
5:   Set  $\bar{p}_j = p_j, \bar{w}_j = w_j$  profit and weight for vertex  $j \in \bar{V}; R_j = \{j\}$ 
6: end for
7:  $k = 0$ ; Build max-heap with values  $\frac{\bar{p}_j}{\bar{w}_j}$  for every  $j \in \bar{V}; r = +\infty$ 
8: while  $\bar{V} \neq \emptyset$  do
9:   Let  $g \in \bar{V}$  be the vertex at the top of the heap (i.e., one of the vertices with largest ratio)
10:   $\triangleright$  If top of the heap  $g$  is a final vertex, we add it to the next closure layer
11:  if  $g \in \bar{V}_{\text{fin}}$  then
12:    if  $\frac{\bar{p}_g}{\bar{w}_g} < r$  then  $k = k + 1, \bar{I}_k = R_g, r = \frac{\bar{p}_g}{\bar{w}_g}$ 
13:    else  $\bar{I}_k = \bar{I}_k \cup R_g$ 
14:    end if
15:    for  $i \in \bar{N}^-(g)$  do  $\bar{V}_{\text{fin}} = \bar{V}_{\text{fin}} \cup \{i\}, \bar{A} = \bar{A} \setminus \{(i, g)\}$ 
16:    end for
17:     $\bar{V}_{\text{fin}} = \bar{V}_{\text{fin}} \setminus \{g\}, \bar{V} = \bar{V} \setminus \{g\}$ ; remove  $g$  from heap and update heap
18:   $\triangleright$  If top of the heap  $g$  is not a final vertex, we “contract” the only arc  $(g, v)$  exiting  $g$ 
19:  else
20:    Let  $(g, v)$  be the only arc exiting  $g$ 
21:    for  $i \in \bar{N}^-(g)$  do  $\bar{A} = \bar{A} \setminus \{(i, g)\}, \bar{A} = \bar{A} \cup \{(i, v)\}$ 
22:    end for
23:     $R_v = R_g \cup R_v, \bar{p}_v = \bar{p}_g + \bar{p}_v, \bar{w}_v = \bar{w}_g + \bar{w}_v$   $\triangleright$  The union of sets is performed in  $O(1)$  operations by concatenating lists.
24:     $\bar{A} = \bar{A} \setminus \{(g, v)\}, \bar{V} = \bar{V} \setminus \{g\}$ ; update  $\frac{\bar{p}_v}{\bar{w}_v}$  in heap
25:  end if
26: end while
27: return  $\mathcal{T} = (\bar{I}_1, \bar{I}_2, \dots, \bar{I}_k)$ 

```

Remark 5. Notice that Algorithm 6 is doing essentially the same operations of PaC, except the recomputation, at each iteration in the main cycle, of the values $P(F_j), W(F_j)$ for every $j \in \bar{V}$ (line 2 in Algorithm 2). Indeed, when the graph is only composed of in-trees, for every arc $a = (i, j) \in A$ the set of vertices F_a is just the singleton $\{i\}$, as illustrated in Fig. 5(a). Furthermore, if at the end of an iteration of the main cycle we remove some final vertices, we do not have to update any of the values $P(F_a), W(F_a)$ with $a \in A$, since final vertices do not belong to sets F_a with $a \in A$, when there are only in-trees. If instead, at the end of an iteration of the main cycle we contract an arc $a = (i, j) \in A$, we only need to update the value of $F_{a'}$, where a' is the only arc exiting j , since in the graph there are only in-trees. Thus, exploiting the in-tree structure, we can employ a heap to extract the vertex with the largest ratio at each iteration, without unnecessary recomputation.

Remark 6. DPaC can also be improved using a heap-based data structure, in the case of a forest of out-trees. Indeed, on an out-tree it is the minimal succeeding set of an arc that collapses to a singleton, $B_a = \{j\}$ for $a = (i, j)$, as illustrated in Fig. 5(b), so that a single min-heap over the current vertices again replaces the two-way comparison. In this case we obtain an algorithm which computes the optimal sequence of closure layers in reverse order, i.e., starting from the closure layer with lowest ratio. The resulting algorithm is denoted with HOPaC. The total complexity is $O(n \log n)$ also in this case. We state this result without proof, since it is analogous to the one of Theorem 2.17.

THEOREM 2.18. *If G is a forest of out-trees, HOPaC computes the optimal sequence of closure layers in $O(n \log n)$ time.*

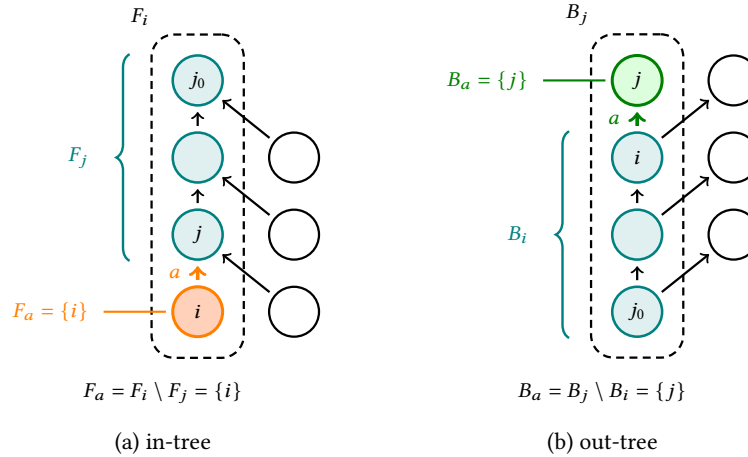


Fig. 5. Why the set associated with an arc collapses to a singleton on the two restricted topologies. **(a)** In an in-tree every vertex has at most one successor, so the preceding set F_i is the unique directed path from i to the root j_0 (dashed contour), and the side branches only ever *enter* that path. For $a = (i, j)$ the path F_j (teal) is exactly F_i deprived of its first vertex, hence $F_a = \{i\}$ (orange). **(b)** In an out-tree every vertex has at most one predecessor, so the succeeding set B_j is the unique directed path from the root j_0 up to j , and the side branches only ever *leave* it; hence $B_a = \{j\}$ (green). In both cases the collapse holds for every arc, and $a \mapsto \text{tail}$ (respectively, head) is a bijection between $\bar{A}$ and the non-root vertices: the $|\bar{A}|$ set ratios coincide with the vertex ratios $\bar{p}_v/\bar{w}_v$, so a single heap over $\bar{V}$ replaces the two-way comparison of PaC – a max-heap in the primal algorithm on in-trees, a min-heap in its dual on out-trees. On a general tree neither difference collapses; compare Fig. 2(a).

2.5 Heap-based variants

Remark 5 explains why a heap data structure cannot improve and, actually, worsens the time-complexity of PaC, when we have general trees. Indeed, if at the end of an iteration of the main cycle we are removing some final vertex $f \in \bar{V}_{\text{fin}}$, then it is necessary to update the values of $P(F_a)$, $W(F_a)$ for every $a = (i, j) \in \bar{A}$ with $i \in B_f$ and $j \notin B_f$, which could be a number of updates of the order of $O(n)$ in the worst case. Similarly, if at the end of an iteration of the main cycle we are contracting an arc $a^* \in \bar{A}$, then it is necessary to update the values of $P(F_a)$, $W(F_a)$ for every $a = (i, j)$ with $i \in B_{a^*}$ and $j \notin B_{a^*}$, which could be a number of updates of the order of $O(n)$ in the worst case. Thus, maintaining a heap data structure with the relevant ratios, in the case of general trees, would cost $O(n \log n)$ operations per iteration of the main cycle, in the worst case, which would worsen the total complexity to $O(n^2 \log n)$ to complete all iterations, instead of $O(n^2)$ if we just recompute all relevant ratios at each iteration. Nevertheless, using a heap in the main algorithm still provides a notable speedup in practice, as observed in Section 5. We will denote with HPaC the heap-based variant of PaC, and with DHPaC the heap-based variant of DPaC. One implementation detail is worth stating, since it is what makes the space bound $O(n)$ hold in practice for the heap-based variants as well: the candidate heaps of HPaC/DHPaC use lazy deletion, and are rebuilt from the live candidates whenever their size exceeds a constant multiple of the number of live candidates. This keeps the memory of the implementation $O(n)$ on every input – including some instances, such as the high-degree stars, where an unbounded lazy heap could use $O(n^2)$ memory.

3 Rake-and-Compress (RaC) algorithm for general trees

This section is dedicated to the description of the RaC algorithm. Following the ACM Policy on Authorship, we describe here the role played by generative AI in obtaining this algorithm. Motivated by the practical behavior of the heap-based algorithm of Section 2, whose computing times increase empirically as $n \log n$ despite its $O(n^2 \log n)$ worst-case complexity, the authors asked Claude (Anthropic) and ChatGPT (OpenAI) whether a suitable data structure could reduce that worst-case complexity to $O(n \log n)$ on general directed forests. The answer was negative, but both tools suggested a different approach, also based on the idea of contracting arcs, together with a sketch of an $O(n \log n)$ complexity argument for it. Starting from that suggestion, the authors reconstructed the pseudocode of the approach and completed the complexity proof, which had only been sketched, obtaining the algorithm described below and the results of Sections 3.3 and 3.4. No other conjecture, proof or scientific claim in this paper originates from a generative AI tool, and the authors take full responsibility for all of them. We restrict the attention to trees since the more general case of a forest can be handled by solving a separate problem over each subtree of the forest. We also assume $|V| \geq 2$: an isolated vertex v has threshold $\theta_v = p_v/w_v$ (see Definition 1.2) and this is directly added to the list of thresholds which is sorted at the end of the top-down phase in Section 3.2. We introduce the following notation. Let $G = (V, A)$ be a tree. For some $Y \subseteq V$ we denote with $T(Y) = (Y, A(Y))$, where $A(Y) = \{(i, j) \in A : i, j \in Y\}$, the subgraph induced by Y , which is a subforest of the tree G . We denote with $\mathcal{L}_G \subset V$ the set of leaves of the tree G , i.e., the set of vertices belonging to a single arc in A (vertices with degree equal to one). We denote with $\mathcal{D}_G^2 \subset V$ the set of vertices in V with degree equal to 2 (no matter if in- or out-degree). Now, we need to introduce the definition of *cluster*.

Definition 3.1. A cluster $C = (C, \partial C)$ is made up of a subset $C \subseteq V$ of vertices, whose induced subgraph $T(C)$ is a subtree of the tree G , together with its boundary $\partial C \subseteq C$, which is made up of a single vertex or by two vertices, and satisfies

$$\{i \in C : \exists j \in V \setminus C \text{ such that } (i, j) \in A \text{ or } (j, i) \in A\} \subseteq \partial C. \quad (6)$$

If $|\partial C| = 1$, the cluster is called a *1-cluster*, while if $|\partial C| = 2$, the cluster is called a *2-cluster*. We also introduce a single cluster with empty boundary and all vertices in its interior, namely, cluster $(V, \emptyset)$.

To each cluster $C = (C, \partial C)$ and to each possible subset of its boundary, defined by a state $\sigma \in \{0, 1\}^{\partial C}$, we associate two functions, f_C^σ and q_C^σ .

Definition 3.2. Function f_C^σ is the equivalent of function φ but restricted to the vertices belonging to the interior $\text{int}(C) = C \setminus \partial C$ of the cluster and subject to the boundary conditions established by state σ . More precisely:

$$\begin{aligned} f_C^\sigma(\lambda) &= \max_{\mathbf{x} \in \{0,1\}^C} \sum_{i \in \text{int}(C)} (p_i - \lambda w_i) x_i \\ x_i - x_j &\leq 0, & (i, j) \in A, i, j \in C, \\ x_i &= \sigma_i, & i \in \partial C. \end{aligned}$$

Note that if the feasible region of the problem above is empty (e.g., if $\partial C = \{i, j\}$, $(i, j) \in A$, and $\sigma_i = 1, \sigma_j = 0$), then we set $f_C^\sigma(\lambda) = -\infty$ for all $\lambda \in (-\infty, +\infty)$. Functions f_C^σ are convex piece-wise linear functions characterized by a number of breakpoints not larger than $|\text{int}(C)|$.

Definition 3.3. Function q_C^σ is the equivalent of function φ restricted to the vertices belonging to the exterior $\text{ext}(C) = V \setminus C$ of the cluster, subject to the boundary conditions established by state σ , with the addition of the

contribution of the boundary itself. More precisely.

$$\begin{aligned}
 q_C^\sigma(\lambda) &= \max_{\mathbf{x} \in \{0,1\}^{\text{ext}(C) \cup \partial C}} \sum_{i \in \text{ext}(C) \cup \partial C} (p_i - \lambda w_i) x_i \\
 x_i - x_j &\leq 0, & (i, j) \in A, i, j \in \text{ext}(C) \cup \partial C, \\
 x_i &= \sigma_i, & i \in \partial C.
 \end{aligned}$$

Again, if the feasible region of the problem above is empty, then we set $q_C^\sigma(\lambda) = -\infty$ for all $\lambda \in (-\infty, +\infty)$.

We make the following key observation.

OBSERVATION 2. *Once the boundary state σ is fixed, the two problems defining f_C^σ and q_C^σ are fully separated, so that for each cluster C , exploiting (6), it holds that*

$$\varphi(\lambda) = \max_{\sigma \in \{0,1\}^{\partial C}} f_C^\sigma(\lambda) + q_C^\sigma(\lambda).$$

Functions f_C^σ are computed during the bottom-up phase of the proposed algorithm. As we will see, functions q_C^σ will turn out to be useful to explain how to compute the closure layers during the top-down phase, but they will not be explicitly computed.

3.1 Bottom-up phase

The bottom-up phase is inspired by the tree contraction strategy of Miller and Reif [26]. This phase is based on the execution of two operations called *compress* and *rake*.

Definition 3.4. Let $C = (C, \partial C)$ be a 2-cluster with $\partial C = \{i, j\}$. The *compress* operation $\text{Compress1}(C, i)$ receives in input C , together with its functions $f_C^{\sigma_i, \sigma_j}$, $\sigma_i, \sigma_j \in \{0, 1\}$, and vertex $i \in \partial C$. It returns in output the cluster $C' = (C, \partial C \setminus \{i\})$ with its related functions $f_{C'}^{\sigma_j}$, $\sigma_j \in \{0, 1\}$, defined as follows:

$$f_{C'}^{\sigma_j}(\lambda) = \max_{\sigma_i \in \{0,1\}} f_C^{\sigma_i, \sigma_j}(\lambda) + \sigma_i(p_i - \lambda w_i).$$

In a special case which occurs at the end of the bottom-up phase, the operation Compress1 is also applied to a 1-cluster C with boundary $\{i\}$. In that case the single function $f_{C'}^\emptyset$ is defined as follows:

$$f_{C'}^\emptyset(\lambda) = \max_{\sigma_i \in \{0,1\}} f_C^{\sigma_i}(\lambda) + \sigma_i(p_i - \lambda w_i).$$

Let $C_1 = (C_1, \partial C_1)$, $C_2 = (C_2, \partial C_2)$ be two 2-clusters with $\partial C_1 = \{i, j\}$ and $\partial C_2 = \{i, h\}$, i.e., they share vertex i in their boundaries. The *compress* operation $\text{Compress2}(C_1, C_2, i)$ receives in input the two clusters C_1 and C_2 , together with the vertex i such that $C_1 \cap C_2 = \{i\}$ (vertex i is the common vertex belonging to the boundaries of the two clusters), and the functions $f_{C_1}^{\sigma_i, \sigma_j}(\lambda)$, $\sigma_i, \sigma_j \in \{0, 1\}$ and $f_{C_2}^{\sigma_i, \sigma_h}(\lambda)$, $\sigma_i, \sigma_h \in \{0, 1\}$. It returns in output the cluster $C' = (C_1 \cup C_2, \{j, h\})$ with its related functions $f_{C'}^{\sigma_j, \sigma_h}$, $\sigma_j, \sigma_h \in \{0, 1\}$, defined as follows:

$$f_{C'}^{\sigma_j, \sigma_h}(\lambda) = \max_{\sigma_i \in \{0,1\}} f_{C_1}^{\sigma_i, \sigma_j}(\lambda) + f_{C_2}^{\sigma_i, \sigma_h}(\lambda) + \sigma_i(p_i - \lambda w_i). \quad (7)$$

LEMMA 3.5. *The number of breakpoints of functions $f_{C'}^{\sigma_j, \sigma_h}(\lambda)$, where cluster $C' = (C_1 \cup C_2, \{j, h\})$ has been obtained through the application of the Compress2 operation on the two clusters $C_1 = (C_1, \partial C_1)$, $C_2 = (C_2, \partial C_2)$ is not larger than $|\text{int}(C_1)| + |\text{int}(C_2)| + 1$ and the time to compute them is $O(|\text{int}(C_1)| + |\text{int}(C_2)| + 1)$. Instead, if the cluster $C' = (C, \partial C \setminus \{i\})$*

has been obtained through the application of the Compress1 operation on the cluster $C = (C, \partial C)$, the number of breakpoints of the functions $f_{C'}^{\sigma_j}(\lambda)$ is not larger than $|\text{int}(C)| + 1$ and the time to compute them is $O(|\text{int}(C)| + 1)$.

PROOF. We only prove the result for the Compress2 operation. We have that $f_{C'}^{\sigma_j, \sigma_h}$ is the parametric optimum over the lattice of closures of C' with fixed boundary, hence by nestedness it has at most $|\text{int}(C')| = |\text{int}(C_1)| + |\text{int}(C_2)| + 1$ breakpoints. To compute it, merge the breakpoint lists of $f_{C_1}^{0, \sigma_j} + f_{C_2}^{0, \sigma_h}$ and $f_{C_1}^{1, \sigma_j} + f_{C_2}^{1, \sigma_h} + (p_i - \lambda w_i)$ (each of size $O(|\text{int}(C_1)| + |\text{int}(C_2)|)$) and take the pointwise maximum in one linear scan; by nestedness the argmax over σ_i switches at most once, so the scan finds the single crossing $\bar{\lambda}$. The overall cost of this operation is $O(|\text{int}(C_1)| + |\text{int}(C_2)| + 1)$. $\square$

Definition 3.6. Let $C_0 = (C_0, \partial C_0)$ be a 2-cluster with $\partial C_0 = \{i, j\}$ and $C_s = (C_s, \partial C_s)$, $s \in \{1, 2, \dots, r\}$, be 1-clusters with $\partial C_s = \{i\}$ and such that $C_s \cap C_t = \{i\}$ for $s \neq t$, $s, t \in \{0, 1, \dots, r\}$ (i.e., all these clusters have pairwise disjoint interiors). The rake operation $\text{Rake}(C_0, \{C_1, C_2, \dots, C_r\})$ receives in input all those clusters together with the functions $f_{C_0}^{\sigma_i, \sigma_j}$, $\sigma_i, \sigma_j \in \{0, 1\}$ associated to cluster C_0 , and functions $f_{C_s}^{\sigma_i}$, $\sigma_i \in \{0, 1\}$, associated to clusters C_s , $s \in \{1, 2, \dots, r\}$. It returns in output the cluster $C' = (\cup_{s=0}^r C_s, \{i, j\})$ with its related functions $f_{C'}^{\sigma_i, \sigma_j}$, $\sigma_i, \sigma_j \in \{0, 1\}$. More precisely:

$$f_{C'}^{\sigma_i, \sigma_j}(\lambda) = f_{C_0}^{\sigma_i, \sigma_j}(\lambda) + \sum_{s=1}^r f_{C_s}^{\sigma_i}(\lambda).$$

The rake operation is also extended to a special case when also C_0 is such that $\partial C_0 = \{i\}$, i.e., it is a 1-cluster with the same boundary of all other clusters $C_1, C_2, \dots, C_r$. In this case we set

$$f_{C'}^{\sigma_i}(\lambda) = \sum_{s=0}^r f_{C_s}^{\sigma_i}(\lambda).$$

The following lemma gives the number of breakpoints for the functions associated to clusters obtained through rake operations (later on we will also discuss the number of operations to compute them).

LEMMA 3.7. The number of breakpoints for the functions $f_{C'}^{\sigma}$ of a cluster C' obtained through the application of a rake operation on clusters $C_s = (C_s, \partial C_s)$, $s \in \{0, 1, \dots, r\}$ is not larger than $\sum_{s=0}^r |\text{int}(C_s)|$.

PROOF. It is enough to observe that functions $f_{C'}^{\sigma}$ are obtained through sums of the functions associated to the clusters C_s , each of which has at most $|\text{int}(C_s)|$ breakpoints. $\square$

The bottom-up phase is detailed in Algorithm 7. In this algorithm for some vertex $i \in V$ the list $\ell_1(i)$ contains the indices of all 1-clusters where i is in the boundary, while $\ell_2(i)$ contains the indices of all 2-clusters where i is in the boundary. The subset $V' \subseteq V$, initialized with V itself, is the subset of vertices that at some iteration are in the boundary of at least one of the clusters or, equivalently, that have not been moved in the interior of one cluster yet. The set A' at some iteration is a set of undirected arcs that join vertices after all the compress operations. It is initialized with the set A of arcs of the tree G , ignoring direction of these arcs, but it is not necessarily a subset of A since some compress operations remove two adjacent arcs and substitute them with a new arc joining the two extremes of the adjacent arcs different from the common extreme. With a slight abuse of notation we keep writing (i, j) for the undirected arc of A' joining i and j , so that (i, j) and (j, i) denote the same element of A' .

Remark 7. It is important to point out what we keep in memory for each cluster C . We keep in memory each function f_C^{σ} together with its breakpoints $\lambda_1^C, \lambda_2^C, \dots, \lambda_t^C$ (not to be confused with the breakpoints $\lambda_1, \lambda_2, \dots, \lambda_k$ of φ) and, for each interval between two consecutive breakpoints $(\lambda_{i+1}^C, \lambda_i^C)$, $i \in \{0, 1, \dots, t\}$, the profit $P\left(S_{C, \sigma}^*(\lambda_{i+1}^C, \lambda_i^C)\right)$ and the

weight $W\left(S_{C,\sigma}^*(\lambda_{i+1}^C, \lambda_i^C)\right)$, where $S_{C,\sigma}^*(\lambda_{i+1}^C, \lambda_i^C)$ is the optimal subset of vertices over interval $(\lambda_{i+1}^C, \lambda_i^C)$ for cluster C with boundary conditions σ (with $\lambda_0^C = +\infty$ and $\lambda_{t+1}^C = -\infty$). We notice that we might also keep in memory the optimal subsets $S_{C,\sigma}^*(\lambda_{i+1}^C, \lambda_i^C)$ themselves. This would considerably simplify the identification of the optimal sequence of closure layers, since they correspond to the differences $S_{C^*,\emptyset}^*(\lambda_{i+1}, \lambda_i) \setminus S_{C^*,\emptyset}^*(\lambda_i, \lambda_{i-1})$, $i \in \{1, 2, \dots, k\}$, where k is the number of breakpoints of function φ . Indeed, it holds that $\mathcal{M}_i = S_{C^*,\emptyset}^*(\lambda_{i+1}, \lambda_i)$, $i \in \{1, 2, \dots, k\}$. However, this simplification comes at the cost of a memory occupation which rises to $\mathcal{O}(n^2)$. As we will see, we are able to reduce such cost through the top-down phase of the algorithm, which allows computing the closure layers with a computational cost of $\mathcal{O}(n \log n)$.

Time- and space-complexity $\mathcal{O}(n \log n)$ of the bottom-up phase are proved in Section 3.3.

3.2 Top-down phase

The top-down phase, detailed in Algorithm 8, allows computing the threshold θ_v for each vertex $v \in V$ (see Definition 1.2). First, we introduce, for each vertex v , the two functions $\hat{g}_x^v(\lambda)$, $x \in \{0, 1\}$:

$$\begin{aligned} \hat{g}_x^v(\lambda) &= \max_{\mathbf{x} \in \{0,1\}^V} \sum_{i \in V} (p_i - \lambda w_i) x_i \\ &\quad x_i - x_j \leq 0, \quad (i, j) \in A, \\ &\quad x_v = x, \end{aligned}$$

i.e., $\hat{g}_x^v(\lambda)$ is the optimal value of problem (1) when we further impose that $x_v = x$. Then, $\varphi(\lambda) = \max\{\hat{g}_0^v(\lambda), \hat{g}_1^v(\lambda)\}$ and the threshold θ_v is the value at which $\hat{g}_1^v(\lambda) - \hat{g}_0^v(\lambda)$ changes sign.

Next, for some $\theta_v \in \mathbb{R}$, let

$$\sigma_v(\lambda) = \begin{cases} 1 & \text{if } \lambda \leq \theta_v \\ 0 & \text{otherwise,} \end{cases}$$

be the indicator function of the interval $(-\infty, \theta_v]$.

For the root node of the tree of clusters $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ returned by the bottom-up phase, we compute functions $g_0^{i_K}$ and $g_1^{i_K}$ as follows:

$$\begin{aligned} g_0^{i_K}(\lambda) &= f_{C_K}^0(\lambda) \\ g_1^{i_K}(\lambda) &= f_{C_K}^1(\lambda) + (p_{i_K} - \lambda w_{i_K}). \end{aligned} \tag{8}$$

Then, we compute the single value θ_{i_K} at which $g_0^{i_K}(\theta_{i_K}) - g_1^{i_K}(\theta_{i_K})$ changes sign, i.e., we compute the threshold of vertex i_K .

Now, let us assume that by moving from the top of the tree of clusters $\mathcal{G}$, we reach the single cluster C' (see Observation 5) where a vertex b has been moved into the interior of that cluster after the execution of a Compress2 operation on two clusters C_1 and C_2 with boundaries $\partial C_1 = \{a, b\}$ and $\partial C_2 = \{b, c\}$. Due to the fact that we move from the top of the tree of clusters, the two thresholds θ_a and θ_c have been already computed when we reach such cluster. Then, for $x \in \{0, 1\}$, let us define

$$g_x^b(\lambda) = f_{C_1}^{\sigma_a(\lambda), x}(\lambda) + f_{C_2}^{x, \sigma_c(\lambda)}(\lambda) + x(p_b - \lambda w_b). \tag{9}$$

Algorithm 7 Bottom-up phase

```

1: input  $G = (V, A)$ ,  $\mathbf{p} \in \mathbb{Z}^V$ ,  $\mathbf{w} \in \mathbb{Z}_+^V$ 
2:  $\triangleright$  Initialization:
3:  $\kappa = 0$ ,  $\mathcal{A} = \emptyset$ 
4: Let  $T' = (V', A')$  be an undirected tree with  $V' = V$  and  $A' = A \triangleright T'$  is a tree initially obtained from  $G$  by removing arc orientation, but updated (contracted) during the execution of the algorithm
5: for  $i \in V$  do
6:    $\ell_1(i) = \ell_2(i) = \{\}$ 
7: end for
8: for  $(i, j) \in A$  do
9:   Let  $C_{\kappa+1} = (\{i, j\}, \{i, j\})$ 
10:  Set  $f_{C_{\kappa+1}}^{1,0}(\lambda) = -\infty$ ,  $f_{C_{\kappa+1}}^{0,0}(\lambda) = f_{C_{\kappa+1}}^{0,1}(\lambda) = f_{C_{\kappa+1}}^{1,1}(\lambda) = 0$ 
11:   $\kappa = \kappa + 1$ 
12:  Set  $\ell_2(i) = \ell_2(i) \cup \{\kappa\}$ ,  $\ell_2(j) = \ell_2(j) \cup \{\kappa\}$ .
13: end for
14: while  $|V'| > 1$  do  $\triangleright$  It must hold that  $|\ell_2(i)| = 1$  for each  $i \in \mathcal{L}_{T'}$  and  $|\ell_1(i)| = 0$  for each  $i \in \mathcal{L}_{T'} \cup \mathcal{D}_{T'}^2$ 
15:   if  $|V'| > 2$  then
16:     Compute a maximal independent set  $I$  in the subgraph of  $T'$  induced by  $\mathcal{L}_{T'} \cup \mathcal{D}_{T'}^2$ , and such that  $\mathcal{L}_{T'} \subseteq I \subseteq \mathcal{L}_{T'} \cup \mathcal{D}_{T'}^2$ 
17:     Set  $K_1 = \mathcal{L}_{T'}$  and  $K_2 = \mathcal{D}_{T'}^2$ 
18:   else
19:     Set  $I = \{i\}$  where  $i \in V'$  is chosen in an arbitrary way,  $K_1 = V'$ , and  $K_2 = \emptyset$ 
20:   end if
21:   for  $i \in K_1 \cap I$  do
22:     Let  $r \in \ell_2(i)$  and  $j \in \partial C_r \setminus \{i\}$ 
23:     Let  $C_{\kappa+1} = \text{Compress1}(C_r, i)$ 
24:     Set  $\kappa = \kappa + 1$ ,  $V' = V' \setminus \{i\}$ ,  $A' = A' \setminus \{(i, j)\}$ 
25:      $\ell_1(j) = \ell_1(j) \cup \{\kappa\}$ ,  $\ell_2(j) = \ell_2(j) \setminus \{r\}$ 
26:     Set  $\mathcal{A} = \mathcal{A} \cup \{(C_\kappa, C_r)\}$  with flag comp1 for the new arc
27:   end for
28:   for  $i \in K_2 \cap I$  do  $\triangleright$  It must hold that  $|\ell_2(i)| = 2$ 
29:     Let  $r, s \in \ell_2(i)$  with  $\partial C_r = \{i, j\}$  and  $\partial C_s = \{i, h\}$ ;
30:     Let  $C_{\kappa+1} = \text{Compress2}(C_r, C_s, i)$ 
31:     Set  $\kappa = \kappa + 1$ ,  $V' = V' \setminus \{i\}$ ,  $A' = A' \cup \{(j, h)\} \setminus \{(i, j), (i, h)\}$ 
32:      $\ell_2(j) = \ell_2(j) \cup \{\kappa\} \setminus \{r\}$ ,  $\ell_2(h) = \ell_2(h) \cup \{\kappa\} \setminus \{s\}$ 
33:     Set  $\mathcal{A} = \mathcal{A} \cup \{(C_\kappa, C_r), (C_\kappa, C_s)\}$  with flag comp2 for the new arcs
34:   end for
35:   for  $i \in \mathcal{L}_{T'} \cup \mathcal{D}_{T'}^2$  :  $\ell_1(i) \neq \emptyset$  do  $\triangleright T'$  is the updated tree after the compress operations
36:     Let  $r \in \ell_2(i)$  and  $j \in \partial C_r \setminus \{i\}$ ;  $\triangleright$  If  $|\ell_2(i)| > 1$ , the choice of  $r$  is arbitrary
37:     Set  $C_{\kappa+1} = \text{Rake}(C_r, \{C_h, h \in \ell_1(i)\})$ 
38:     Set  $\kappa = \kappa + 1$ 
39:      $\mathcal{A} = \mathcal{A} \cup \{(C_\kappa, C_r), (C_\kappa, C_h), h \in \ell_1(i)\}$ , with flag rake for all the new arcs
40:      $\ell_2(i) = \ell_2(i) \cup \{\kappa\} \setminus \{r\}$ ,  $\ell_1(i) = \emptyset$ ,  $\ell_2(j) = \ell_2(j) \cup \{\kappa\} \setminus \{r\}$   $\triangleright$  Note that the rake operation empties all sets  $\ell_1(i)$  for  $i \in \mathcal{L}_{T'} \cup \mathcal{D}_{T'}^2$ 
41:   end for
42:   if  $|V'| = 1$  then
43:     Let  $i \in V'$ 
44:     if  $\ell_2(i) = \emptyset$  and  $|\ell_1(i)| \geq 2$  then
45:       Set  $C_{\kappa+1} = \text{Rake}(\{C_h, h \in \ell_1(i)\})$ 
46:       Set  $\kappa = \kappa + 1$ 
47:        $\mathcal{A} = \mathcal{A} \cup \{(C_\kappa, C_h), h \in \ell_1(i)\}$ , with flag rake for all the new arcs
48:        $\ell_1(i) = \{\kappa\}$ 
49:     end if
50:   end if
51: end while  $\triangleright$  When we exit the while cycle,  $C_\kappa = (V, \{i\})$  for some  $i \in V$ 
52: Let  $i \in \partial C_\kappa$ 
53: Let  $C^* = \text{Compress1}(C_\kappa, i) = (V, \emptyset)$ 
54: Let  $\varphi(\lambda) = f_{C^*}^0(\lambda)$ 
55: Set  $\mathcal{V} = \{C_1, C_2, \dots, C_\kappa\}$ 
56: return  $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ ,  $\varphi, f_C^\sigma$ ,  $C \in \mathcal{V}$ ,  $\sigma \in \{0, 1\}^{\partial C}$ 

```

We prove the following lemma.

LEMMA 3.8. *The threshold θ_b is the λ value at which $g_1^b(\lambda) - g_0^b(\lambda)$ changes sign.*

PROOF. We have that

$$\hat{g}_{\sigma_b(\lambda)}^b(\lambda) = g_{\sigma_b(\lambda)}^b(\lambda) + q_{C'}^{\sigma_a(\lambda), \sigma_c(\lambda)}(\lambda) = \varphi(\lambda),$$

while for $x = 1 - \sigma_b(\lambda)$ and $\lambda \neq \theta_b$ it holds that

$$g_x^b(\lambda) + q_{C'}^{\sigma_a(\lambda), \sigma_c(\lambda)}(\lambda) \leq \hat{g}_x^b(\lambda) < \varphi(\lambda).$$

The first inequality follows from the fact that

$$\begin{aligned}\hat{g}_x^b(\lambda) &= \max_{\sigma_a, \sigma_c \in \{0,1\}} \left[f_{C_1}^{\sigma_a, x}(\lambda) + f_{C_2}^{x, \sigma_c}(\lambda) + x(p_b - \lambda w_b) + q_{C'}^{\sigma_a, \sigma_c}(\lambda) \right] \geq \\ &\geq g_x^b(\lambda) + q_{C'}^{\sigma_a(\lambda), \sigma_c(\lambda)}(\lambda).\end{aligned}$$

Therefore, for $\lambda < \theta_b$ we have

$$\hat{g}_1^b(\lambda) = g_1^b(\lambda) + q_{C'}^{\sigma_a(\lambda), \sigma_c(\lambda)}(\lambda) = \varphi(\lambda) > \hat{g}_0^b(\lambda) \geq g_0^b(\lambda) + q_{C'}^{\sigma_a(\lambda), \sigma_c(\lambda)}(\lambda),$$

so that $g_1^b(\lambda) - g_0^b(\lambda) > 0$, while for $\lambda > \theta_b$

$$\hat{g}_0^b(\lambda) = g_0^b(\lambda) + q_{C'}^{\sigma_a(\lambda), \sigma_c(\lambda)}(\lambda) = \varphi(\lambda) > \hat{g}_1^b(\lambda) \geq g_1^b(\lambda) + q_{C'}^{\sigma_a(\lambda), \sigma_c(\lambda)}(\lambda),$$

so that $g_1^b(\lambda) - g_0^b(\lambda) < 0$. Therefore, the threshold θ_b is the λ value at which $g_1^b(\lambda) - g_0^b(\lambda)$ changes sign. $\square$

Assuming, w.l.o.g., that $\theta_a \leq \theta_c$, functions g_x^b , $x \in \{0, 1\}$, are convex piece-wise linear functions over each one of the three intervals $(-\infty, \theta_a]$, $(\theta_a, \theta_c]$, and $(\theta_c, +\infty)$. Over each interval, the number of breakpoints of functions g_x^b , $x \in \{0, 1\}$, is $O(|\text{int}(C_1)| + |\text{int}(C_2)| + 1)$. The value at which $g_1^b(\lambda) - g_0^b(\lambda)$ changes sign is searched for over the three intervals. Note that one should take into account that the sign might change at one extreme of the intervals, i.e., at θ_a or θ_c . In that case one should study what happens both in the interval $(-\infty, \theta_a]$ and in the interval $(\theta_a, \theta_c]$ (respectively, both in the interval $(\theta_a, \theta_c]$ and in the interval $(\theta_c, +\infty)$).

Next, let us assume that we reach the single cluster (see, again, Observation 5) where vertex b has been moved into the interior of such cluster through the execution of a Compress1 operation on a single cluster C_1 with boundary $\partial C_1 = \{a, b\}$. As before, the threshold θ_a has been already computed. For $x \in \{0, 1\}$, we define the two functions

$$g_x^b(\lambda) = f_{C_1}^{\sigma_a(\lambda), x}(\lambda) + x(p_b - \lambda w_b). \quad (10)$$

As before, we can prove that the threshold θ_b is the single λ value where $g_0^b(\theta_b) - g_1^b(\theta_b)$ changes sign. In this case, functions g_x^b , $x \in \{0, 1\}$, are convex piece-wise linear functions over each one of the two intervals $(-\infty, \theta_a]$ and $(\theta_a, +\infty)$. Over each interval, the number of breakpoints of functions g_x^b , $x \in \{0, 1\}$, is $O(|\text{int}(C_1)| + 1)$.

We remark that during the top-down phase, arcs with flag rake, which do not move vertices in the interiors of clusters, are traversed without doing anything.

Time- and space-complexity $O(n \log n)$ of the top-down phase are proved in Section 3.4.

Appendix C illustrates both phases of RaC on the running instance of Fig. 1, treating its arcs as undirected, as RaC does throughout this section.

3.3 Complexity of the bottom-up phase

To prove the complexity of the bottom-up phase we need some results. We first make a couple of observations. Note that we call *active* clusters those belonging to the sets ℓ_1, ℓ_2 , i.e., the set of active clusters is

$$C_{\text{act}} = \left\{ C_r : r \in \bigcup_{i \in V'} [\ell_1(i) \cup \ell_2(i)] \right\}.$$

OBSERVATION 3. *Active clusters have pair-wise disjoint interiors.*

Algorithm 8 Top-down phase

```

1: input the tree of clusters  $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ , its root  $C_\kappa$ , functions  $f_{C_i}^\sigma, i \in \{1, 2, \dots, \kappa\}, p \in \mathbb{Z}^V, w \in \mathbb{Z}_+^V$ 
2:  $\triangleright$  Initialization:
3:  $Q = \{C_\kappa\}$ 
4: Compute functions  $g_0^{i\kappa}$  and  $g_1^{i\kappa}$  through (8)
5: Compute the threshold  $\theta_{i\kappa}$  at which  $g_0^{i\kappa}(\lambda) - g_1^{i\kappa}(\lambda)$  changes sign
6: while  $Q \neq \emptyset$  do
7:   Set  $Q' = Q$ 
8:   for  $C_r \in Q'$  do
9:     if no outgoing arcs in  $\mathcal{A}$  from  $C_r$  then  $\triangleright C_r$  is a leaf of  $\mathcal{G}$ 
10:      Set  $Q = Q \setminus \{C_r\}$ 
11:     else if single arc in  $\mathcal{A}$  from  $C_r$  with flag comp1 then  $\triangleright C_r$  1-cluster with boundary  $\{i_r\}$ 
12:       Let  $(C_r, C_l) \in \mathcal{A}$ , with  $l < r$ , be the outgoing arc
13:       Let  $i_l \in \partial C_l \setminus \partial C_r$ 
14:       Compute functions  $g_0^{i_l}$  and  $g_1^{i_l}$  through (10) with  $C_1 = C_l$ 
15:       Compute the threshold  $\theta_{i_l}$  at which  $g_0^{i_l}(\lambda) - g_1^{i_l}(\lambda)$  changes sign
16:       Set  $Q = Q \cup \{C_l\} \setminus \{C_r\}$ 
17:     else if two arcs in  $\mathcal{A}$  from  $C_r$  with flag comp2 then  $\triangleright C_r$  2-cluster with boundary  $\{i_{l_1}, i_{l_2}\}$ 
18:       Let  $(C_r, C_{l_1}), (C_r, C_{l_2}) \in \mathcal{A}$ , with  $l_1, l_2 < r$ , be the outgoing arcs
19:       Let  $i_r \in \partial C_{l_1} \cap \partial C_{l_2}$ 
20:       Compute functions  $g_0^{i_r}$  and  $g_1^{i_r}$  through (9) with  $C_1 = C_{l_1}, C_2 = C_{l_2}$ 
21:       Compute the threshold  $\theta_{i_r}$  at which  $g_0^{i_r}(\lambda) - g_1^{i_r}(\lambda)$  changes sign
22:       Set  $Q = Q \cup \{C_{l_1}, C_{l_2}\} \setminus \{C_r\}$ 
23:     else arcs in  $\mathcal{A}$  from  $C_r$  with flag rake
24:       Let  $(C_r, C_p) \in \mathcal{A}$ , with  $l_p < r, p \in \{1, 2, \dots, P\}, P \geq 2$  (flag rake), be the outgoing arcs
25:       Set  $Q = Q \cup \{C_p\}_{p=1}^P \setminus \{C_r\}$ 
26:     end if
27:   end for
28: end while
29: Let  $\bar{\lambda}_j, j \in \{1, 2, \dots, t\}, t \leq |V|$ , be the  $t$  distinct threshold values  $\theta_v, v \in V$ , sorted in decreasing order.
30: Set  $I_j = \{v : \theta_v = \bar{\lambda}_j\}, j \in \{1, 2, \dots, t\}$ 
31: return  $\{I_1, I_2, \dots, I_t\}, \{\bar{\lambda}_1, \bar{\lambda}_2, \dots, \bar{\lambda}_t\}$ 

```

OBSERVATION 4. *Each vertex belongs to the interior of at most two clusters involved in rake operations executed at a given iteration. Indeed, a 2-cluster C with boundary $\{i, j\}$ might be involved in a first rake operation raking the clusters in $\ell_1(i)$. The resulting cluster, denoted with C' and still with boundary $\{i, j\}$, might be involved in a second rake operation raking the clusters in $\ell_1(j)$. The interiors of C and C' may partially overlap, so that a vertex may belong to the interior of at most two clusters involved in rake operations executed at a given iteration.*

Next, we prove that at the beginning of each iteration of the while cycle we have that $\ell_1(i) = \emptyset$ for all $i \in \mathcal{L}_{T'} \cup \mathcal{D}_{T'}^2$, while the sets $\ell_2(i)$ are related to the set of arcs A' .

LEMMA 3.9. *At the beginning of each iteration of the while cycle it holds that:*

- for each $i \in \mathcal{L}_{T'} \cup \mathcal{D}_{T'}^2$, $\ell_1(i) = \emptyset$;
- for each $i \in V'$ it holds that $r \in \ell_2(i)$, with $\partial C_r = \{i, j\}$ if and only if $(i, j) \in A'$;

As a consequence of the second point, it also holds that $|\ell_2(i)| = 1$ for $i \in \mathcal{L}_{T'}$ and $|\ell_2(i)| = 2$ for $i \in \mathcal{D}_{T'}^2$.

PROOF. The statement is true after the initialization phase. Next, at the beginning of each iteration $\ell_1(i) = \emptyset$ for $i \in \mathcal{L}_{T'} \cup \mathcal{D}_{T'}^2$, since all 1-clusters with i in the boundary are removed through rake operations at the end of the previous iteration. Concerning the second point, the sentence is certainly true after initialization. A Compress1 operation on a cluster with boundary $\{i, j\}$, removes such 2-cluster and replaces it with a 1-cluster, by also removing the arc (i, j) from A' . Since Compress2 operations generate new 2-clusters whose boundary is made up of vertices j, h joined by an arc (j, h) added to A' , while the vertices of the boundary of 2-clusters generated by rake operations are also connected by arcs in A' , this is also true across all the iterations. $\square$

The following result limits from above the number of active clusters at each iteration.

LEMMA 3.10. *At each iteration of the while cycle, the set of active clusters*

$$C_{\text{act}} = \left\{ C_r : r \in \bigcup_{i \in V'} [\ell_1(i) \cup \ell_2(i)] \right\}$$

contains at most $2n$ clusters.

PROOF. This is true after initialization, when the cardinality of $\bigcup_{i \in V'} [\ell_1(i) \cup \ell_2(i)]$ is equal to $|A'| = |A|$. As proved in Lemma 3.9, at the beginning of each iteration of the while cycle the sets $\ell_1(i)$ are empty for $i \in \mathcal{L}_{T'} \cup \mathcal{D}_{T'}^2$. Then, all clusters with $\ell_1(i) \neq \emptyset$ must be such that the degree of i in T' is larger than 2 and must have disjoint interiors in view of Observation 3. Moreover, again in view of Lemma 3.9, there is one 2-cluster associated to each arc in A' . Therefore:

$$|\bigcup_{i \in V'} [\ell_1(i) \cup \ell_2(i)]| \leq n + |A'| \leq 2n.$$

During the execution of an iteration the cardinality of $\bigcup_{i \in V'} [\ell_1(i) \cup \ell_2(i)]$ does not increase, so that it never exceeds $2n$. $\square$

Next lemma shows that at each iteration the number of vertices in V' is reduced by a constant factor and, consequently, that the number of iterations is $O(\log n)$.

LEMMA 3.11. *At each iteration all vertices $i \in I$ are removed from V' and the cardinality of V' at the end of the iteration is reduced to at most $(5/6)$ of its cardinality at the beginning of the iteration. Consequently, the number of iterations is $O(\log n)$.*

PROOF. The key observation is that in any tree the number of leaves is at least as large as the number of vertices with degree not lower than three. Consequently, $|\mathcal{L}_{T'}| + |\mathcal{D}_{T'}^2| \geq |V'|/2$. Moreover, the subgraph induced by $\mathcal{L}_{T'} \cup \mathcal{D}_{T'}^2$, has degree not larger than two and by a greedy algorithm one can compute an independent set with cardinality $\geq \frac{1}{3}(|\mathcal{L}_{T'}| + |\mathcal{D}_{T'}^2|) \geq |V'|/6$. Therefore, at each iteration the cardinality of V' is reduced by a factor $(5/6)$. Since we stop when $|V'| = 1$, the maximum number of iterations is t such that

$$(5/6)^t n \leq 1,$$

i.e., $t = \log_{\frac{6}{5}} n = O(\log n)$. $\square$

Note that each independent set can be computed by a greedy approach in time $O(|V'|)$, so that the overall cost for the computation of the independent sets is $O(\sum_{i=1}^t (5/6)^i n) = O(n)$.

Next, we make some observations.

OBSERVATION 5. *Each vertex v is moved in the interior of a cluster by a single Compress1 or Compress2 operation.*

The following observation is a consequence of Lemmas 3.5 and 3.7.

OBSERVATION 6. *Any compress or rake operation involving clusters $C_1, C_2, \dots, C_s$, $s \geq 1$, returns a new cluster C' and for all possible states σ (at most four, since the boundary of each cluster has cardinality at most equal to two), function $f_{C'}^\sigma$ has at most $1 + \sum_{j=1}^s |\text{int}(C_j)|$ breakpoints.*

As a consequence of Observation 6, we have the following corollary.

COROLLARY 3.12. *The total number of breakpoints associated to active clusters at each iteration is $O(n)$.*

PROOF. The total number of breakpoints associated to active clusters is

$$\sum_{C_r \in \mathcal{C}_{\text{act}}} \sum_{\sigma} [|\text{int}(C_r)| + 1].$$

Since the interiors of the active clusters involved in compress operations are disjoint, since the possible states σ are at most four, and since the number of active clusters is not larger than $2n$, then we have that the total number of breakpoints associated to all active clusters involved in compress operations (and the number of operations required to compute them) is $O(n)$. In a similar way we see that the total number of breakpoints associated to all active clusters involved in rake operations is $O(n)$. $\square$

Recalling Lemma 3.5, the cost of computing the breakpoints during a compress operation is $O(|\text{int}(C_1)| + |\text{int}(C_2)| + 1)$ for a Compress2 operation and $O(|\text{int}(C)| + 1)$ for a Compress1 operation. The proof of Corollary 3.12 shows that the overall cost of computing breakpoints generated by compress operations at each iteration is $O(n)$, so that the cost of computing them across all iterations is $O(n \log n)$.

Now we would like to establish the cost of rake operations. Let us denote with $r_1^j, r_2^j, \dots, r_{H_j}^j$ the rake operations performed at iteration j of the while cycle. Moreover, for a rake operation r_h^j and for each fixed boundary state, let $C_1^{r_h}, C_2^{r_h}, \dots, C_{\eta_{r_h}}^{r_h}$ be the clusters whose breakpoint lists are nonempty. Empty breakpoint lists require no merging work; in particular, the functions associated with an initial cluster are constant, with value either 0 or $-\infty$. If all breakpoint lists are empty, the merging cost is zero. Otherwise, let $L_s^{r_h}$ be the number of breakpoints in the list associated with cluster $C_s^{r_h}$ (it holds that $1 \leq L_s^{r_h} \leq |\text{int}(C_s^{r_h})|$), and let $S_{r_h}^j = \sum_{s=1}^{\eta_{r_h}} L_s^{r_h}$. Recalling Observation 4, at each iteration j the interiors of clusters involved in all rake operations are not necessarily disjoint but each vertex appears in at most two interiors. Then, $\sum_{h=1}^{H_j} S_{r_h}^j \leq 2n$. The Huffman's greedy algorithm (see [21, 27]) merges the list of breakpoints of the clusters involved in the rake operation r_h^j by merging at each iteration the two current shortest lists and thus building a so-called *merging tree*. This requires a number of operations equal to

$$\sum_{s=1}^{\eta_{r_h}} L_s^{r_h} d_s^{r_h} \leq \sum_{s=1}^{\eta_{r_h}} L_s^{r_h} \log_2 \frac{S_{r_h}^j}{L_s^{r_h}} + S_{r_h}^j, \quad (11)$$

where: $d_s^{r_h}$ is the depth of the list associated to cluster $C_s^{r_h}$ in the merging tree. Also notice that the greedy algorithm needs to build a priority queue whose cost is $O(\eta_{r_h} \log \eta_{r_h})$ for each rake operation. We notice that summing the values η_{r_h} across all rake operations, we have that such sum is $O(n)$ since each 1-cluster is involved in at most one rake operation and the overall number of 1-clusters is $O(n)$ (1-clusters are generated only through Compress1 operations and their number at each iteration is not larger than $|V'|$). Thus, summing $O(\eta_{r_h} \log \eta_{r_h})$ across all rake operations gives a cost $O(n \log n)$ for the priority queues.

Now, we need an upper bound for (11), which is obtained as follows. Let $\hat{d}_s = \left\lceil \log_2 (S_{r_h}^j / L_s^{r_h}) \right\rceil$. Then, $\sum_s 2^{-\hat{d}_s} \leq \sum_s L_s^{r_h} / S_{r_h}^j = 1$. In view of Kraft inequality (see, e.g., [12]) there exists a merging tree with depth $\hat{d}_s$. Since the greedy algorithm provides the merging tree with minimum cost, we have that:

$$\sum_s L_s^{r_h} d_s^{r_h} \leq \sum_s L_s^{r_h} \hat{d}_s \leq \sum_s L_s^{r_h} \log_2 \frac{S_{r_h}^j}{L_s^{r_h}} + S_{r_h}^j.$$

Now, let $v_s^{r_h} = |\text{int}(C_s^{r_h})| \geq L_s^{r_h}$ and $\Sigma_{r_h}^j = \sum_s v_s^{r_h} \geq S_{r_h}^j$. Then:

$$\sum_s L_s^{r_h} \log_2 \frac{S_{r_h}^j}{L_s^{r_h}} + S_{r_h}^j \leq \sum_s v_s^{r_h} \log_2 (\Sigma_{r_h}^j / v_s^{r_h}) + \Sigma_{r_h}^j. \quad (12)$$

Now, we are ready to prove the following result about the cost of rake operations.

LEMMA 3.13. *The overall cost of the rake operations executed during the bottom-up phase is $O(n \log n)$.*

PROOF. First, we notice that, similarly to $\sum_{h=1}^{H_j} S_{r_h}^j \leq 2n$, it also holds that $\sum_{h=1}^{H_j} \Sigma_{r_h}^j \leq 2n$. If we further sum across all iterations the resulting term is $O(n \log n)$, which is the overall contribution coming from the terms $\Sigma_{r_h}^j$ in the upper bound (12). Thus, we can restrict the attention to the sum $\sum_{s=1}^{\eta_{r_h}} v_s^{r_h} \log_2 (\Sigma_{r_h}^j / v_s^{r_h})$. We notice that such sum can be recovered if we assign a cost equal to $\log_2 (\Sigma_{r_h}^j / v_s^{r_h})$ to each vertex $v \in \text{int}(C_s^{r_h})$. Indeed, if we sum across all clusters involved in the rake operation and across all vertices belonging to the interiors of these clusters, we find exactly the sum $\sum_{s=1}^{\eta_{r_h}} v_s^{r_h} \log_2 (\Sigma_{r_h}^j / v_s^{r_h})$.

Now, for a given vertex v , let $r_h^{j_1}, r_h^{j_2}, \dots, r_h^{j_{z_v}}$ be the rake operations in which v is involved at the iterations $j_1 \leq j_2 \leq \dots \leq j_{z_v}$. Recall that two consecutive indices j_m and j_{m+1} can be equal, since, as previously observed, the same vertex may be involved in two rake operations of a given iteration, but not more than two. The key observation is the following. Let L^m be the value v associated to the single cluster containing v at iteration j_m , and let S^m be the value Σ produced by the rake operation involving the cluster containing v . It holds that $L^{m+1} \geq S^m$ for $m \in \{1, 2, \dots, z_v - 1\}$. Indeed, after the rake operation at iteration j_m , the resulting cluster has v -value at least S^m , and the inequality follows from the fact that the cardinality of the interiors of the clusters generated by these rake operations never decreases. As a consequence:

$$\sum_{m=1}^{z_v} \log_2 \frac{S^m}{L^m} = \sum_{m=1}^{z_v} (\log_2 S^m - \log_2 L^m) \leq \log_2 S^{z_v} - \log_2 L^1 \leq \log_2 n,$$

which is the overall cost associated to a single vertex v across all iterations. Now, to bound from above the cost of all rake operations, we just need to sum the upper bound $\log_2 n$ across all vertices v , which leads to the upper bound $O(n \log n)$. $\square$

Now, having observed that the overall cost of the compress and rake across all iterations of the while cycle is $O(n \log n)$, and since $O(n \log n)$ is also the sum of the cardinality of the interiors of all clusters in $\mathcal{V}$, generated during the bottom-up phase, we have the following complexity result.

PROPOSITION 3.14. *The time and space complexity of the bottom-up phase is $O(n \log n)$.*

3.4 Complexity of the top-down phase

We have the following result for the complexity of the top-down phase.

PROPOSITION 3.15. *The top-down phase requires $O(n \log n)$ operations.*

PROOF. We observe that each vertex is compressed in the interior of a cluster exactly once (see Observation 5) and that each cluster (except the root node) has a single father in the tree of clusters $\mathcal{G}$. For each vertex b the evaluation of the change of sign for function $g_1^b(\lambda) - g_0^b(\lambda)$ requires an effort of $O(|\text{int}(C_1)| + |\text{int}(C_2)| + 1)$ if b has been moved in the interior of a cluster through a Compress2 operation. Indeed, detecting the change of sign requires a linear scan of

the list obtained by merging the breakpoints of the two clusters. Similarly, the cost is $O(|\text{int}(C_1)| + 1)$ if b has been moved in the interior of a cluster through a `Compress1` operation. By summing this effort across all clusters in $\mathcal{V}$, we have that this is $O(n \log n)$. Indeed, we proved that the sum of the cardinalities of the interiors of all active clusters at a given iteration is $O(n)$, so that by summing through all the $O(\log n)$ iterations, we have the $O(n \log n)$ result, which is also the cost of the top-down phase. $\square$

Note that the distinct threshold values $\bar{\lambda}_j$, $j \in \{1, 2, \dots, t\}$, obtained at the end of the top-down phase are also the breakpoints of function φ , so that $t = k$. Moreover, the thresholds are returned in decreasing order, which requires a further $O(n \log n)$ cost, which does not change the overall complexity of the algorithm.

4 A comparison between PaC/HPaC and RaC

From the algorithmic point of view PaC/HPaC and RaC share the common idea of progressively contracting the tree until it is reduced to a single vertex. In PaC/HPaC the contraction is a greedy one, where an arc is contracted into a single vertex when it is the current one with largest ratio (or, in the dual version, with smallest ratio). Instead, in RaC contractions are not made to decide which vertices should go into the same closure layer, but to iteratively compute functions f_C^σ associated to the clusters, until we are able to identify function φ . While contracting vertices, closure layers are not formed. The composition of closure layers is made in the following top-down phase.

In terms of time-complexity, RaC, whose time-complexity is $O(n \log n)$, is better than PaC, whose time-complexity is $O(n^2)$, and than HPaC, whose time-complexity even rises to $O(n^2 \log n)$. On the other hand, RaC has also space-complexity $O(n \log n)$, while PaC/HPaC have space-complexity $O(n)$.

The comparison between these approaches will be completed in the following experimental section (in particular, in Section 5.3). There we will also evaluate the practical performance of the approaches and observe that in practical cases HPaC usually performs a number of operations $\approx n \log n$, so that it is often highly competitive with respect to RaC, although the latter performs better on special structured instances such as star graphs.

5 Computational Results

All algorithms are implemented in C++ and run single-threaded. Every ratio comparison uses exact integer arithmetic, so the reported closure layers and thresholds are exact. We report median CPU times and median per-instance paired ratios. The complete experimental protocol, machine specification, validation checks, raw data, and additional analyses are available in the accompanying `parametric-closure-forests` repository (release `v0.4.0`; see Code and Data Availability). The C++ implementation of all the algorithms, the instance generators and the benchmark and analysis pipeline underlying this section were developed with the assistance of Claude Code (Anthropic), used interactively as a coding assistant; all the code was independently reviewed by the authors and validated against an exhaustive correctness oracle, cross-algorithm differential testing and sanitizer builds, as documented in the repository.

5.1 Benchmark instances

This subsection describes the generation of the benchmark instances used in the experiments that follow. Every instance combines two independently generated components, an affine-coefficient family and a topology, and is produced by a seeded, deterministic script, so that the whole benchmark can be regenerated exactly.

We begin with the affine coefficients. For each vertex i , weight w_i is always drawn strictly positive; profit p_i may be signed. Six families are used, chosen only for the shape of the resulting ratio p_i/w_i , never for any external classification:

- independent-positive: w_i, p_i drawn independently and uniformly from $\{1, 2, \dots, 1000\}$ – dispersed, positive ratios.
- independent-signed: w_i uniform on $\{1, 2, \dots, 1000\}$, p_i uniform on $\{-1000, -999, \dots, 1000\}$, independently – some vertices locally unfavorable.
- correlated: w_i uniform on $\{1, 2, \dots, 1000\}$, $p_i = w_i + \delta_i$ with δ_i uniform on $\{-50, -49, \dots, 50\}$ – ratios concentrated near 1.
- anti-correlated: w_i uniform on $\{1, 2, \dots, 1000\}$, $p_i = (1001 - w_i) + \delta_i$ with δ_i uniform on $\{-50, -49, \dots, 50\}$ – more heterogeneous ratios.
- near-ties: one target ratio num/den per instance, with num, den uniform on $\{1, 2, \dots, 20\}$; for each vertex i , $w_i = \text{den} \cdot t_i$ and $p_i = \text{num} \cdot t_i + j_i$, with t_i uniform on $\{1, 2, \dots, 50\}$ and jitter j_i uniform on $\{-2, -1, 1, 2\}$ – ratios near-equal but distinct, stressing exact rational comparisons between close thresholds.
- exact-ties: as near-ties, but with no jitter and with $G = \max(1, \lfloor n/20 \rfloor)$ target ratios num _{g} /den _{g} instead of one, each drawn uniformly on $\{1, 2, \dots, 20\}^2$; vertex i is assigned to group $g = i \bmod G$, so that the vertices sharing a ratio are spread over the whole graph rather than adjacent – exactly equal ratios, stressing the canonicalization rule that merges consecutive closure layers at an equal threshold.

We now turn to the topology of the precedence graph, which is generated independently of the coefficients. Six topologies are used. In the three named mixed, every edge is oriented in either direction with probability 1/2, so that the resulting directed graph is in general neither a forest of in-trees nor a forest of out-trees, i.e., it is the general case addressed by PaC and RaC.

- mixed-forest: a random forest with randomly oriented arcs. An undirected forest is built by connecting each vertex $v \in \{2, 3, \dots, n\}$ to a uniformly chosen predecessor $u < v$ independently with probability ϱ (the density), and each resulting edge is then oriented independently as (u, v) or (v, u) with probability 1/2 each; $\varrho \in \{0.3, 0.6, 0.9, 1.0\}$: the resulting forest has $\varrho(n-1)$ arcs and $n - \varrho(n-1)$ connected components in expectation, so $\varrho = 1.0$ gives a connected spanning tree and $\varrho = 0.3$ a forest of many small trees.
- in-forest: a random forest of in-trees. Vertices are scanned $u \in \{1, 2, \dots, n-1\}$ and, independently with probability ϱ , one arc (u, j) is added with j uniform on $\{u+1, u+2, \dots, n\}$, giving out-degree at most one everywhere.
- out-forest: a random forest of out-trees. Vertices are scanned $v \in \{2, 3, \dots, n\}$ and, independently with probability ϱ , one arc (u, v) is added with u uniform on $\{1, 2, \dots, v-1\}$, giving in-degree at most one everywhere.
- path-mixed, binary-mixed, star-mixed: three structured topologies, whose underlying undirected graph is, respectively, a path, a balanced binary tree and a star. In the implementation the vertices are labelled from 0 to $n-1$: the parent of $v > 0$ is $v-1$ on the path and $\lfloor (v-1)/2 \rfloor$ on the binary tree, while the center of a star is vertex 0. As for mixed-forest, orient every edge independently as (parent, v) or (v, parent) with probability 1/2.

The benchmark is obtained by combining each topology with each affine-coefficient family. Throughout this section, n denotes the number of vertices of an instance, and every reported value is aggregated over all the instances that share the same topology and the same value of n . For each of the three random topologies mixed-forest, in-forest and out-forest, which depend on the density ϱ , and for each value of n , we generate one instance for every combination of the four densities, the six coefficient families and ten random seeds, that is, $4 \times 6 \times 10 = 240$ instances per topology and per value of n . For each of the three structured topologies path-mixed, binary-mixed and star-mixed, which have

no density parameter, and for each value of n , we generate one instance for every combination of the six coefficient families and ten random seeds, that is, $6 \times 10 = 60$ instances per topology and per value of n .

5.2 Justification of the Heap-Based Variant

We first compare the direct-scan algorithms (PaC and its dual DPaC) with their heap-based counterparts (HPaC, DHPaC) on mixed-forest instances with $n \in \{100, 200, \dots, 1\,000\} \cup \{10\,000, 20\,000\}$, the sizes on which the direct-scan algorithms are affordable. Fig. 6 reports the median CPU times. On these instances we observed that the four algorithms are essentially equivalent up to $n \approx 300$ (at $n = 100$ the direct scan is in fact marginally faster, median PaC/HPaC = 0.91), and that the heap pays off from $n \approx 400$ on: the median ratio PaC/HPaC grows to 2.3 at $n = 1\,000$, 20.3 at $n = 10\,000$, and 42.9 at $n = 20\,000$, with the duals behaving analogously.

n	#inst	CPU time (ms)			
		PaC	HPaC	DPaC	DHPaC
100	240	0.1	0.1	0.1	0.1
200	240	0.2	0.2	0.2	0.2
300	240	0.3	0.3	0.3	0.3
400	240	0.5	0.3	0.5	0.3
500	240	0.7	0.4	0.7	0.4
600	240	0.9	0.5	1.0	0.5
700	240	1.2	0.6	1.3	0.6
800	240	1.8	0.8	1.6	0.7
900	240	2.2	0.9	2.0	0.8
1 000	240	2.2	0.9	2.4	0.9
10 000	240	315.1	12.7	417.5	13.2
20 000	240	1 425.2	28.0	1 897.3	29.8

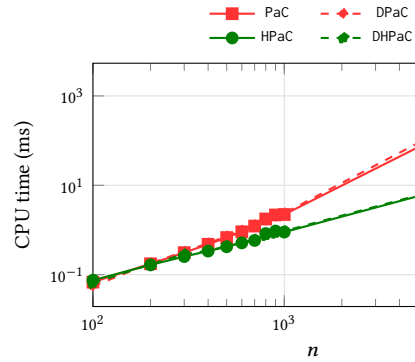


Fig. 6. Median CPU time of PaC, DPaC, HPaC and DHPaC on mixed-forest instances.

5.3 Scaling on Large Random Forests and Structured Instance Classes

Fig. 7 reports the median CPU times of HPaC, DHPaC and RaC on mixed-forest instances with n from 10 000 to 100 000 in steps of 10 000. HPaC solves the full parametric problem on forests with 10^5 vertices in about 0.2 s; DHPaC stays within 25% of it at every size (median paired ratio between 0.96 and 1.24), so the choice between the primal and the dual schedule is immaterial at this scale. On these instances the median paired ratio RaC/HPaC grows from 4.2 at $n = 10^4$ to 13.3 at $n = 10^5$. This aggregate ordering is unchanged across the tested densities and affine-coefficient families.

We next focus on three structured instance classes – paths, balanced binary trees, and stars, always with mixed arc orientations – which isolate the effect of the topology. Paths and binary trees behave like the random forests: the median ratio RaC/HPaC lies between 5.8 and 9.0 on paths and between 3.2 and 3.9 on binary trees, from $n = 100$ to $n = 100\,000$. Stars invert the picture, increasingly sharply as n grows (Fig. 8): RaC is already faster than HPaC at $n = 100$ (median ratio 0.75) and is about 250 times faster at $n = 20\,000$. The mechanism is structural: on a star, every peel or contraction touches the hub and invalidates the candidate ratios of a constant fraction of the remaining leaves, so any heap-based schedule pays $\Theta(n)$ maintenance per event, while RaC’s rake operations absorb all leaves in $O(\log n)$ rounds regardless of the hub’s update history. The direct scan is much less affected, since it performs no candidate maintenance at all: PaC stays about one order of magnitude below HPaC over the whole range, and RaC overtakes it between $n = 1\,000$ and $n = 2\,000$.

n	#inst	CPU time (ms)		
		HPaC	DHPaC	RaC
10 000	240	12.7	13.2	50.5
20 000	240	28.0	29.8	131.5
30 000	240	44.9	47.2	262.8
40 000	240	62.9	66.9	399.7
50 000	240	84.3	92.3	543.9
60 000	240	96.1	117.9	652.5
70 000	240	117.4	141.7	837.0
80 000	240	141.1	161.5	1 027.5
90 000	240	164.9	198.1	1 304.7
100 000	240	203.9	214.6	1 709.4

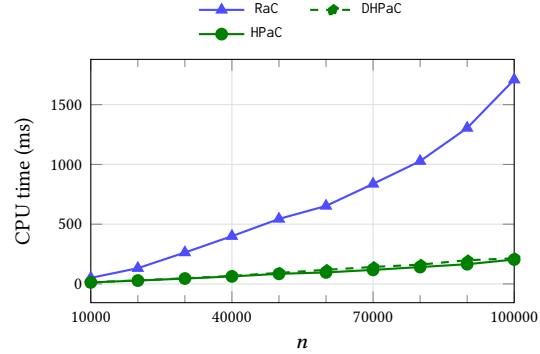


Fig. 7. Median CPU time of HPaC, DHPaC and RaC on large mixed-forest instances.

n	#inst	CPU time (ms)		
		PaC	HPaC	RaC
100	60	0.1	0.6	0.4
200	60	0.2	2.0	0.8
500	60	1.1	12.6	2.1
1 000	60	4.5	62.0	5.2
2 000	60	17.6	231.2	9.7
5 000	60	123.8	1 427.6	24.8
10 000	60	643.1	6 898.8	63.3
20 000	60	2 890.8	26 127.1	122.1

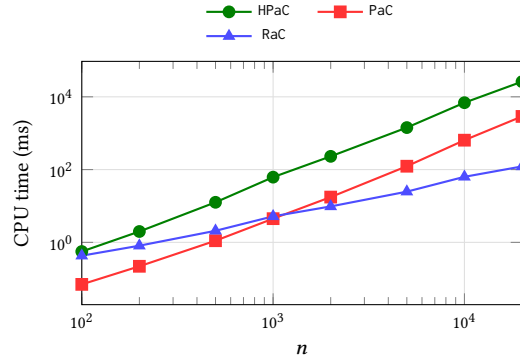


Fig. 8. Median CPU time of PaC, HPaC and RaC on star-mixed instances.

5.4 Specialized Variants for In-Trees and Out-Trees

We evaluate the specialized $O(n \log n)$ heap variants on the orientation each is built for: HIPaC on in-forest and HOPaC on out-forest instances (4 800 instances per orientation: 20 sizes, four densities, six families, ten seeds), always against the general HPaC on the same instances. Fig. 9 reports the median times for $n \in \{10\,000, 20\,000, \dots, 100\,000\}$. On these instances the specialized variants take a stable fraction of HPaC's time – median per-size ratios between 0.52 and 0.75 for HIPaC and between 0.57 and 0.78 for HOPaC – with no erosion of the advantage up to $n = 10^5$. The gain reflects Remark 5: on a single orientation the minimal preceding (succeeding) set of every arc collapses to a singleton, so one heap value per vertex suffices and no closure-sum propagation is ever needed. On the same oriented instances RaC loses ground as n grows, with median RaC/HPaC rising from about 2.8 at $n \leq 1\,000$ to 12.8 at $n = 10^5$ on in-forests, and similarly on out-forests.

5.5 Comparison with the Fully Parametric Pseudoflow Algorithm

Finally, we compare HPaC with the public implementation of the *fully parametric* pseudoflow algorithm of [18] (FPHPF in what follows; version 2, May 2025, see Code and Data Availability), which applies to arbitrary precedence graphs. As already mentioned, its complexity is $O(mn \log n)$ on a graph with n vertices and m arcs; since a forest has $m \leq n - 1$, on

n	#inst	in-forest (ms)		out-forest (ms)	
		HIPaC	HPaC	HOPaC	HPaC
10 000	240	6.5	14.2	7.6	13.7
20 000	240	14.4	29.5	16.5	31.9
30 000	240	25.0	49.6	27.8	51.4
40 000	240	35.2	73.8	40.3	73.6
50 000	240	45.1	95.3	54.7	95.3
60 000	240	52.8	109.4	61.5	106.1
70 000	240	66.0	127.8	75.5	135.6
80 000	240	78.7	168.4	88.4	164.4
90 000	240	86.5	203.1	106.0	185.7
100 000	240	114.2	237.6	132.1	229.5

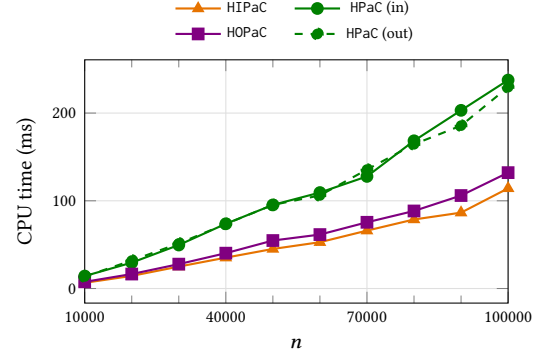


Fig. 9. Median CPU time of the specialized variants HIPaC and HOPaC against the general HPaC, on in-forest and out-forest instances.

the instances considered here that bound reads $O(n^2 \log n)$. Each instance is encoded through the standard maximum-closure/minimum-cut reduction, with source- and sink-adjacent arc capacities affine in the parameter; the range is $[-\max_i |p_i| - 1, \max_i |p_i| + 1]$, which contains every vertex ratio since $w_i \geq 1$. From the output of FPHPF we take only the partition of the vertices into closure layers, and we recompute every breakpoint exactly as $\lambda_r = P(\mathcal{I}_r)/W(\mathcal{I}_r)$, so that no floating-point value produced by FPHPF enters the comparison. Its internal solve timer (input parsing excluded) is compared against HPaC's in-process time for the full parametric sweep on the same instance; both are medians of 11 repetitions, measured in the same session. The comparison runs on all 2 400 mixed-forest instances with $n \in \{100, 200, \dots, 1\,000\}$.

n	#inst	CPU time (ms)		
		HPaC	FPHPF	ratio
100	240	0.15	1.98	9.2
200	240	0.37	5.85	14.2
300	240	0.52	9.62	17.0
400	240	0.65	12.09	18.0
500	240	0.71	14.11	18.6
600	240	0.71	15.96	20.5
700	240	0.76	18.12	22.6
800	240	0.90	23.79	21.7
900	240	1.00	29.16	24.9
1 000	240	1.07	35.04	27.6

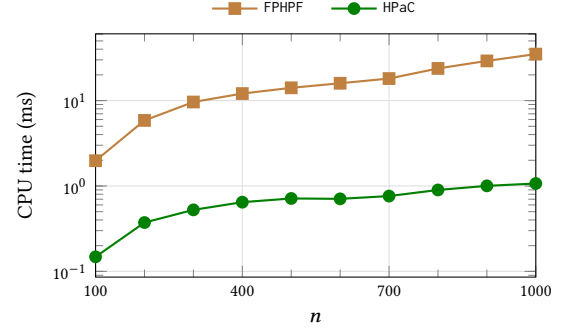


Fig. 10. Median CPU time of HPaC and of the fully parametric pseudoflow implementation FPHPF on mixed-forest instances, both solving each instance from scratch; “ratio” is the median over the instances of the paired ratio FPHPF/HPaC.

On these instances the two methods return the same closure-layer partition and the same breakpoints on every instance, and HPaC is faster on every instance. The median paired ratio FPHPF/HPaC grows from 9.2 at $n = 100$ to 27.6 at $n = 1\,000$ (Fig. 10), with an overall median of 19.5. The gap keeps widening with size: on the larger mixed-forest instances the median ratio is about 250 at $n = 10^4$, 550 at $n = 2 \cdot 10^4$, 1 300 at $n = 5 \cdot 10^4$ and 4 000 at $n = 10^5$, where a single FPHPF run takes about 15 minutes against 0.23 s for HPaC; the complete measurements are reported in the repository.

From $n = 10^4$ on, FPHPF also stops returning the exact layer sequence on some instances: it merges consecutive closure layers on 1 of 24 instances at $n = 10^4$, on 2 of 24 at $n = 2 \cdot 10^4$, on 2 of 6 at $n = 5 \cdot 10^4$, and on the $n = 10^5$ instance tested. The cause is a fixed absolute tolerance in the implementation, which brackets every computed intersection of two linear pieces by $\pm 10^{-7}$ in the parameter, so that two breakpoints closer than about $2 \cdot 10^{-7}$ cannot both be found; the thresholds involved are perfectly distinct as floating-point numbers (in the first such case they are 2299/2226 and 2362/2287, whose difference is $1/5\,090\,862$). On the 2 400 instances with $n \leq 1\,000$ the smallest gap between consecutive breakpoints is $2.21 \cdot 10^{-7}$, just above that threshold, and indeed no merge occurs; as n grows the breakpoints become denser and the merges become systematic. This is the numerical-precision phenomenon of Remark 3; the algorithms of this paper are unaffected, since they only compare integers.

For the sake of correctness, although the above results show that HPaC outperforms FPHPF, we stress once again that FPHPF applies to arbitrary precedence graphs, while our algorithms are specific to forests.

6 Conclusion

In this paper we studied the parametric MCP with affine costs when the precedence graph is a forest. We presented two aggregation-based algorithms, PaC and DPaC, that compute the complete sequence of closure layers in $O(n^2)$ time and $O(n)$ space on general forests. On the restricted classes of in-trees and out-trees, their specialized heap-based variants HIPaC and HOPaC run in $O(n \log n)$ time. For general forests, the heap-based implementations HPaC and DHPaC retain an $O(n^2 \log n)$ worst-case bound. We also presented RaC, which computes the same parametric solution on general forests in $O(n \log n)$ time and space, which matches the $\Omega(n \log n)$ lower bound for comparison-based algorithms.

The computational study complements these bounds. On the tested random forests, paths, and balanced binary trees, HPaC is faster than RaC; on stars, RaC becomes increasingly faster as the instance size grows. The specialized variants HIPaC and HOPaC consistently improve on the general heap-based implementation on their respective oriented classes. Finally, we compared HPaC with the public implementation of the fully parametric pseudoflow algorithm of [18]: HPaC is faster on every tested instance, by a margin growing with n , and, relying only on exact integer comparisons, it is not affected by the numerical tolerance which makes that implementation merge close layers on the larger instances. This comparison is of course limited to forests, whereas the pseudoflow method applies to arbitrary precedence graphs.

Several directions are left open by this work. A first one concerns the class of precedence graphs. The clusters used by RaC have a boundary of at most two vertices, which is what keeps the number of boundary states, and hence the work per cluster, constant. It is natural to ask whether the same machinery, applied to the decomposition tree of a series-parallel digraph, recovers the $O(n \log n)$ bound known for that class [13, 23], and how far it can be pushed towards richer classes: precedence graphs of bounded treewidth, where a boundary of k vertices leads to 2^k boundary states, or graphs that differ from a forest by a bounded number of arcs.

A second direction concerns the problem itself. Both PaC and RaC rely on precedence constraints being hard: in the minimum-cut formulation of the closure problem, the arcs encoding them have infinite capacity, so that contracting one of them never contributes to the objective. Extending the approach to parametric minimum-cut problems on tree-shaped networks with finite arc capacities, where a violated precedence is paid for rather than forbidden, would require the aggregation steps of PaC and the cluster functions of RaC to keep track of the capacities of the arcs involved, and would make the present algorithms applicable to a genuinely larger class of parametric problems.

Code and Data Availability

The public GitHub repository <https://github.com/fabiofurini/parametric-closure-forests> contains the implementation, instance generators, full experimental protocol, complete computational details, and the raw and processed benchmark data underlying Section 5. The archived release v0.4.0 (commit 9423cd4) is self-contained and includes the full instance archive (instances.zip) and the raw and processed benchmark data (results.zip), together with their SHA-256 checksums. The pseudoflow implementation compared against in Section 5.5 is the fully parametric HPF solver of the Hochbaum group, version 2 (May 2025), available at <https://github.com/hochbaumGroup/pseudoflow-parametric-cut-v2> (commit fbd480f); its source is not redistributed in our repository but downloaded from the authors' repository by a script pinned to that commit, and the only local changes are two output format strings (timing and parameter values printed with more digits), recorded as a patch.

References

- [1] Karl Abrahamson, Norm Dadoun, David G. Kirkpatrick, and Teresa Przytycka. 1989. A Simple Parallel Tree Contraction Algorithm. *Journal of Algorithms* 10, 2 (1989), 287–302. [https://doi.org/10.1016/0196-6774\(89\)90017-5](https://doi.org/10.1016/0196-6774(89)90017-5)
- [2] D. Adolphson and T. C. Hu. 1973. Optimal linear ordering. *SIAM J. Appl. Math.* 25, 3 (1973), 403–423. <https://doi.org/10.1137/0125042>
- [3] Stephen Alstrup, Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. 2005. Maintaining Information in Fully Dynamic Trees with Top Trees. *ACM Transactions on Algorithms* 1, 2 (2005), 243–264. <https://doi.org/10.1145/1103963.1103966> Preprint: arXiv:cs/0310065.
- [4] Michel L Balinski. 1970. On a selection problem. *Management Science* 17, 3 (1970), 230–231. <https://doi.org/10.1287/mnsc.17.3.230>
- [5] F Barahona and D. Jensen. 1998. Plant location with minimum inventory. *Mathematical Programming* 83, 1 (1998), 101–111. <https://doi.org/10.1007/BF02680552>
- [6] Arne Beines, Michael Kaibel, Philip Mayer, Petra Mutzel, and Jonas Sauer. 2025. A Simpler Approach for Monotone Parametric Minimum Cut: Finding the Breakpoints in Order. In *Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX 2025)*. SIAM, Philadelphia, PA, 29–41. <https://doi.org/10.1137/1.9781611978339.3>
- [7] Michael Ben-Or. 1983. Lower bounds for algebraic computation trees. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing (STOC)*. ACM, New York, NY, 80–86. <https://doi.org/10.1145/800061.808735>
- [8] Gerth Stølting Brodal, George Lagogiannis, and Robert E. Tarjan. 2025. Strict Fibonacci Heaps. *ACM Transactions on Algorithms* 21, 2, Article 15 (Jan. 2025), 18 pages. <https://doi.org/10.1145/3707692>
- [9] Josiah Carlson and David Eppstein. 2006. The Weighted Maximum-Mean Subtree and Other Bicriterion Subtree Problems. In *Algorithm Theory – SWAT 2006 (Lecture Notes in Computer Science, Vol. 4059)*. Springer, Berlin, Heidelberg, 397–408. https://doi.org/10.1007/11785293_37 Preprint: arXiv:cs/0503023.
- [10] Renaud Chicoisne, Daniel Espinoza, Marcos Goycoolea, Eduardo Moreno, and Enrique Rubio. 2012. A new algorithm for the open-pit mine production scheduling problem. *Operations Research* 60, 3 (2012), 517–528. <https://doi.org/10.1287/opre.1120.1050>
- [11] Denis Cornaz, Fabio Furini, Mathieu Lacroix, Enrico Malaguti, A Ridha Mahjoub, and Sébastien Martin. 2019. The vertex k-cut problem. *Discrete Optimization* 31 (2019), 8–28. <https://doi.org/10.1016/j.disopt.2018.07.003>
- [12] Thomas M. Cover and Joy A. Thomas. 2006. *Elements of Information Theory* (2nd ed.). Wiley-Interscience, Hoboken, NJ.
- [13] David Eppstein. 2017. The Parametric Closure Problem. *ACM Transactions on Algorithms* 14, 1, Article 2 (Dec. 2017), 22 pages. <https://doi.org/10.1145/3147212>
- [14] Greg N. Frederickson. 1997. Ambivalent Data Structures for Dynamic 2-Edge-Connectivity and k Smallest Spanning Trees. *SIAM J. Comput.* 26, 2 (1997), 484–538. <https://doi.org/10.1137/S0097539792226825>
- [15] Zvi Galil. 1980. Applications of Efficient Mergeable Heaps for Optimization Problems on Trees. *Acta Informatica* 13, 1 (1980), 53–58. <https://doi.org/10.1007/BF00288535>
- [16] G. Gallo, M. D. Grigoriadis, and R. E. Tarjan. 1989. A fast parametric maximum flow algorithm and applications. *SIAM J. Comput.* 18, 1 (1989), 30–55. <https://doi.org/10.1137/0218003>
- [17] Dorit S Hochbaum. 2001. A new—old algorithm for minimum-cut and maximum-flow in closure graphs. *Networks* 37, 4 (2001), 171–193. <https://doi.org/10.1002/net.1012>
- [18] Dorit S Hochbaum. 2008. The pseudoflow algorithm: A new algorithm for the maximum-flow problem. *Operations Research* 56, 4 (2008), 992–1009. <https://doi.org/10.1287/opre.1080.0524>
- [19] Dorit S Hochbaum and Anna Chen. 2000. Performance analysis and best implementations of old and new algorithms for the open-pit mining problem. *Operations Research* 48, 6 (2000), 894–914. <https://doi.org/10.1287/opre.48.6.894.12392>
- [20] W. A. Horn. 1972. Single-machine job sequencing with treelike precedence ordering and linear delay penalties. *SIAM J. Appl. Math.* 23, 2 (1972), 189–202. <https://doi.org/10.1137/0123021>

- [21] David A Huffman. 1952. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE* 40, 9 (1952), 1098–1101. <https://doi.org/10.1109/JRPROC.1952.273898>
- [22] Thys Brentwood Johnson. 1968. *Optimum open pit mine production scheduling*. Ph.D. Dissertation. University of California, Berkeley.
- [23] E. L. Lawler. 1978. Sequencing jobs to minimize total weighted completion time subject to precedence constraints. *Annals of Discrete Mathematics* 2 (1978), 75–90. [https://doi.org/10.1016/S0167-5060\(08\)70323-6](https://doi.org/10.1016/S0167-5060(08)70323-6)
- [24] Helmut Lerchs and Ingo Grossmann. 1965. Optimum design of open-pit mines. *CIM Bulletin* 58 (1965), 47–54.
- [25] François Margot, Maurice Queyranne, and Yaoguang Wang. 2003. Decompositions, Network Flows, and a Precedence Constrained Single-Machine Scheduling Problem. *Operations Research* 51, 6 (2003), 981–992. <https://doi.org/10.1287/opre.51.6.981.24912>
- [26] Gary L. Miller and John H. Reif. 1989. Parallel Tree Contraction Part 1: Fundamentals. *Advances in Computing Research* 5 (1989), 47–72. <https://api.semanticscholar.org/CorpusID:5274022>
- [27] Alistair Moffat. 2019. Huffman coding. *Comput. Surveys* 52, 4 (2019), 1–35. <https://doi.org/10.1145/3342555>
- [28] Arman Mollakhani, Pieter Wuille, and Dongning Guo. 2026. Bitcoin Mempool Linearization. arXiv:2607.23787 [cs.DS] <https://arxiv.org/abs/2607.23787>
- [29] Jean-Claude Picard. 1976. Maximal Closure of a Graph and Applications to Combinatorial Problems. *Management Science* 22, 11 (1976), 1268–1272. <https://doi.org/10.1287/mnsc.22.11.1268>
- [30] J. B. Sidney. 1975. Decomposition algorithms for single-machine sequencing with precedence relations and deferral costs. *Operations Research* 23, 2 (1975), 283–298. <https://doi.org/10.1287/opre.23.2.283>
- [31] R. Vallet. 1976. Optimisation mathématique de l’exploitation d’une mine à ciel ouvert ou le problème de l’enveloppe. *Annales des Mines de Belgique* 2 (1976), 113–135.

Appendices

The three appendices report complete step-by-step executions of the three algorithms on the running instance of Fig. 1, with the quantities compared at every iteration shown explicitly.

A Execution of PaC

At every iteration, PaC computes the largest ratio among the minimal preceding sets associated with the current arcs,

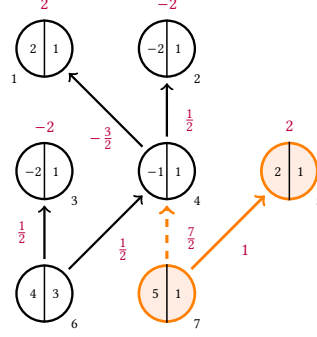
$$\mu_A = \max_{a \in \bar{A}} \frac{P(F_a)}{W(F_a)},$$

and the largest ratio among the current final vertices,

$$\mu_{\text{fin}} = \max_{f \in \bar{V}_{\text{fin}}} \frac{\bar{p}_f}{\bar{w}_f}.$$

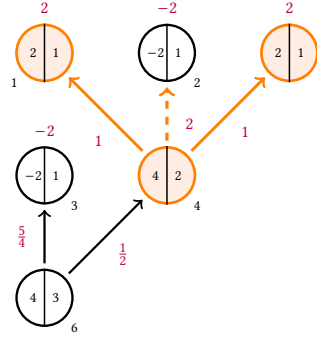
Let $a^* = (u, v)$ attain μ_A and let $\mathcal{B}_{\text{fin}}$ contain all final vertices attaining μ_{fin} . If $\mu_{\text{fin}} > \mu_A$, the sets represented by the vertices in $\mathcal{B}_{\text{fin}}$ form the next closure layer and are removed. If $\mu_A \geq \mu_{\text{fin}}$, arc a^* is contracted and its endpoints are represented by a single aggregated vertex. Thus, equality is resolved by contraction. In the diagrams, purple labels are the ratios compared at the current iteration; orange identifies the selected wing F_{a^*} and its boundary arc a^* ; teal identifies the selected final vertices.

Step 1 – first comparison and contraction of (7, 4). Initially the final vertices are $\{1, 2, 3, 5\}$, and the largest final-vertex ratio is $\mu_{\text{fin}} = 2$, attained by $\mathcal{B}_{\text{fin}} = \{1, 5\}$. The largest wing ratio is attained by $a^* = (7, 4)$: its minimal preceding set is $F_{(7,4)} = \{7, 5\}$ and has ratio $7/2$. Since $7/2 > 2$, the algorithm contracts $(7, 4)$. Vertex 4 becomes the representative of the merged set $R_4 = \{4, 7\}$, with $\bar{p}_4 = 4$ and $\bar{w}_4 = 2$.



(a) Initial PaC iteration. The selected wing is $F_{(7,4)} = \{7, 5\}$, with ratio $7/2$; the best final vertices have ratio 2.

Step 2 – a tie and the second contraction. After contracting $(7, 4)$, the current representative 4 stores $R_4 = \{4, 7\}$. The best wing is now associated with $a^* = (4, 2)$: $F_{(4,2)} = \{1, 4, 5\}$ has ratio 2. The best final vertices are still $\mathcal{B}_{\text{fin}} = \{1, 5\}$, also with ratio 2. Because the algorithm contracts whenever $\mu_A \geq \mu_{\text{fin}}$, the tie is resolved by contracting $(4, 2)$. Vertex 2 becomes the representative of $R_2 = \{2, 4, 7\}$, with $\bar{p}_2 = 2$ and $\bar{w}_2 = 3$.

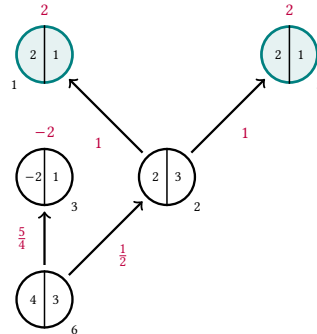


(b) After the first contraction. The best wing and the best final vertices both have ratio 2, so PaC contracts $(4, 2)$.

Step 3 – the first peel and the first closure layer. After contracting $(4, 2)$, vertex 2 represents $R_2 = \{2, 4, 7\}$. The current final vertices are $\{1, 3, 5\}$; the best ones are again $\mathcal{B}_{\text{fin}} = \{1, 5\}$ with ratio 2. The best wing is $F_{(6,3)}$, whose ratio is $5/4$. Hence $\mu_{\text{fin}} > \mu_A$, and the final vertices are peeled. Since $R_1 = \{1\}$ and $R_5 = \{5\}$, the first closure layer is

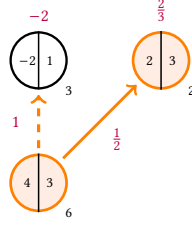
$$\mathcal{I}_1 = R_1 \cup R_5 = \{1, 5\}, \quad \lambda_1 = 2.$$

Removing vertices 1 and 5 leaves the three-vertex tree shown in the next step.



(c) After the second contraction. The best final vertices $\{1, 5\}$ have ratio 2, larger than the best wing ratio $5/4$, and therefore form $\mathcal{I}_1$.

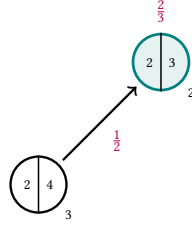
Step 4 – contraction of (6, 3). After peeling $\mathcal{I}_1$, the remaining graph contains representatives 2, 3, and 6. Vertex 2 is the best final vertex, with ratio $2/3$. The best wing is associated with $a^* = (6, 3)$ and is $F_{(6,3)} = \{6, 2\}$, with ratio 1. Since $1 > 2/3$, PaC contracts (6, 3). Vertex 3 becomes the representative of $R_3 = \{3, 6\}$, with $\bar{p}_3 = 2$ and $\bar{w}_3 = 4$.



(d) After peeling $\mathcal{I}_1$. The wing $F_{(6,3)} = \{6, 2\}$ has ratio 1, larger than the best final-vertex ratio $2/3$.

Step 5 – the second peel. After contracting (6, 3), only representatives 2 and 3 remain, with the single arc (3, 2). The wing $F_{(3,2)} = \{3\}$ has ratio $1/2$, while the final representative 2 has ratio $2/3$. The final vertex therefore wins the comparison and is peeled. Since it represents $R_2 = \{2, 4, 7\}$, the second closure layer is

$$\mathcal{I}_2 = \{2, 4, 7\}, \quad \lambda_2 = \frac{2}{3}.$$

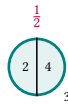


(e) After contracting (6, 3). The final representative 2 has ratio $2/3$, larger than the only wing ratio $1/2$, and therefore yields $\mathcal{I}_2$.

Step 6 – the last closure layer. After removing representative 2, the graph consists only of representative 3. There are no arcs, so the only possible move is to peel this final vertex. It represents $R_3 = \{3, 6\}$ and has ratio $1/2$. Hence

$$\mathcal{I}_3 = \{3, 6\}, \quad \lambda_3 = \frac{1}{2}.$$

The graph is now empty and PaC terminates.

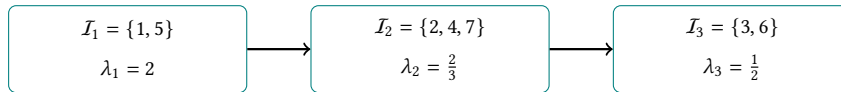


(f) Final iteration. Representative 3 is the only remaining final vertex and yields $\mathcal{I}_3 = \{3, 6\}$.

Result. The contractions have only aggregated vertices that are eventually assigned to the same closure layer; the peel operations identify the layers themselves. The complete optimal sequence returned by PaC is therefore

$$\mathcal{I}_1 = \{1, 5\}, \quad \mathcal{I}_2 = \{2, 4, 7\}, \quad \mathcal{I}_3 = \{3, 6\},$$

with strictly decreasing ratios 2, $2/3$, and $1/2$, respectively.

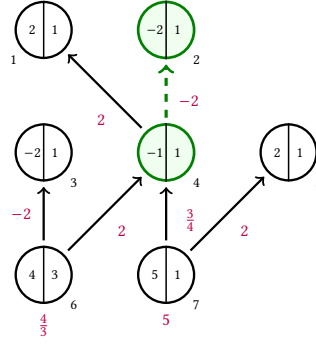


(g) Closure layers returned by PaC, in decreasing order of their ratios.

B Execution of DPaC

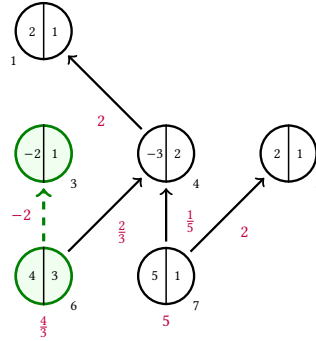
The dual variant, denoted here by DPaC, performs the symmetric construction and returns the closure layers in increasing order of their ratios. At each iteration it compares the minimum ratio of the minimal succeeding sets B_a associated with the current arcs with the minimum ratio of the current initial vertices. A strict win by the initial vertices produces the next layer from the bottom; otherwise the selected arc is contracted, so equality is again resolved by contraction.

Step 1 – contract (4, 2). Initially, the minimum succeeding-set ratio equals -2 and is attained by two arcs, (4, 2) and (6, 3); the tie is broken arbitrarily and we let $a^* = (4, 2)$. The minimum ratio among the initial vertices is $4/3$, attained by vertex 6. Hence DPaC contracts (4, 2), and the representative 4 stores $R_4 = \{2, 4\}$, with $\bar{p}_4 = -3$ and $\bar{w}_4 = 2$.



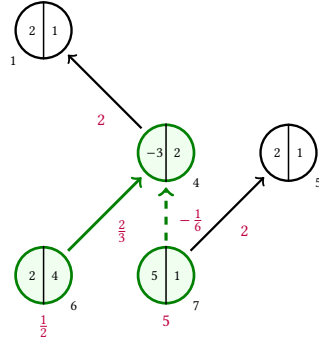
(a) Initial dual iteration. The minimum succeeding-set ratio is -2 , attained by $B_{(4,2)}$ and $B_{(6,3)}$; the minimum initial-vertex ratio is $4/3$.

Step 2 – contract (6, 3). After the first contraction, the minimum succeeding-set ratio is again -2 , now attained by the only remaining arc that achieves it, $a^* = (6, 3)$, while vertex 6 is still the minimum-ratio initial vertex with ratio $4/3$. Thus (6, 3) is contracted and representative 6 stores $R_6 = \{3, 6\}$, with $\bar{p}_6 = 2$ and $\bar{w}_6 = 4$.



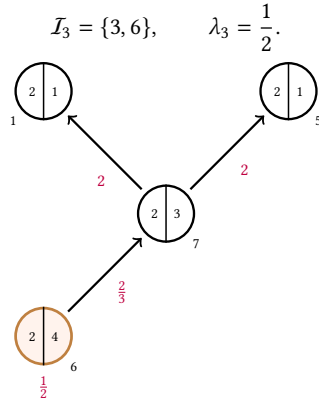
(b) After contracting (4, 2). The minimum succeeding-set ratio is attained by (6, 3), so the second operation is again a contraction.

Step 3 – contract (7, 4). With $R_4 = \{2, 4\}$ and $R_6 = \{3, 6\}$, the minimum succeeding-set ratio is $-1/6$, attained by $a^* = (7, 4)$, whereas the minimum initial-vertex ratio is now $1/2$, attained by representative 6. Hence (7, 4) is contracted and representative 7 stores $R_7 = \{2, 4, 7\}$, with $\bar{p}_7 = 2$ and $\bar{w}_7 = 3$.



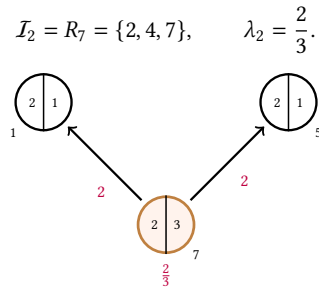
(c) After the first two contractions. The selected succeeding set is associated with $(7, 4)$ and has ratio $-1/6$.

Step 4 – the first layer found by the dual algorithm. After contracting $(7, 4)$, representatives 1, 5, 6, and 7 remain. The minimum initial-vertex ratio is $1/2$, attained by representative 6, while the minimum succeeding-set ratio is $2/3$. The initial vertex therefore wins strictly and is peeled. Since $R_6 = \{3, 6\}$, the dual algorithm first identifies the lowest-ratio closure layer



(d) The initial representative 6 has ratio $1/2$, below the best succeeding-set ratio $2/3$, and yields $\mathcal{I}_3$.

Step 5 – the second layer. After removing representative 6, the minimum initial-vertex ratio is $2/3$, attained by representative 7, while the minimum succeeding-set ratio is 2. Therefore representative 7 is peeled and yields



(e) Representative 7 is now the minimum-ratio initial vertex and yields $\mathcal{I}_2$.

Step 6 – the last layer. Only vertices 1 and 5 remain. Both are initial vertices, both have ratio 2, and there are no arcs. Hence the last layer returned by the dual run is

$$\mathcal{I}_1 = \{1, 5\}, \quad \lambda_1 = 2.$$

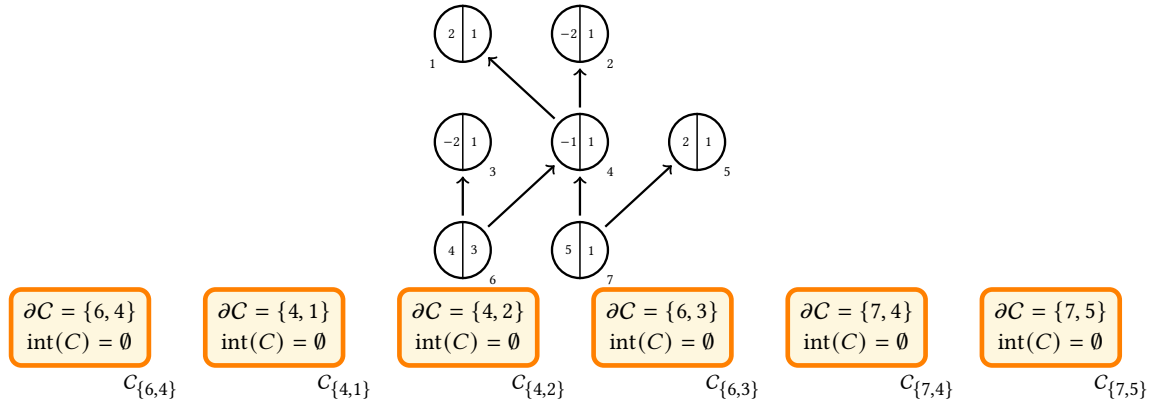
Thus DPAC has recovered the same partition as PaC, but in the order $\mathcal{I}_3, \mathcal{I}_2, \mathcal{I}_1$.



C Execution of RaC

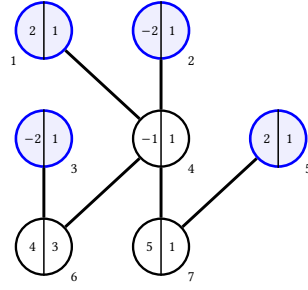
Panels (a)–(n) show the complete mechanics of the Rake-and-Compress algorithm step by step. The original precedence graph is directed. Only when deciding which vertices are leaves, which have degree two, and which vertices can be contracted simultaneously, RaC works on the underlying undirected tree. The original arc orientations are retained in the feasibility conditions and in all cluster functions f_C^σ .

Step 0 – original directed instance and initialization of elementary clusters. The running instance has the six precedence arcs $(4, 1)$, $(4, 2)$, $(6, 3)$, $(6, 4)$, $(7, 4)$, $(7, 5)$. At initialization, every arc (i, j) creates an elementary 2-cluster $(\{i, j\}, \{i, j\})$ with empty interior. Because the interior is empty, its finite cluster functions are initially zero; the boundary state that violates the directed precedence constraint is assigned value $-\infty$. These elementary 2-clusters are the leaves of the cluster tree that will be built bottom-up.



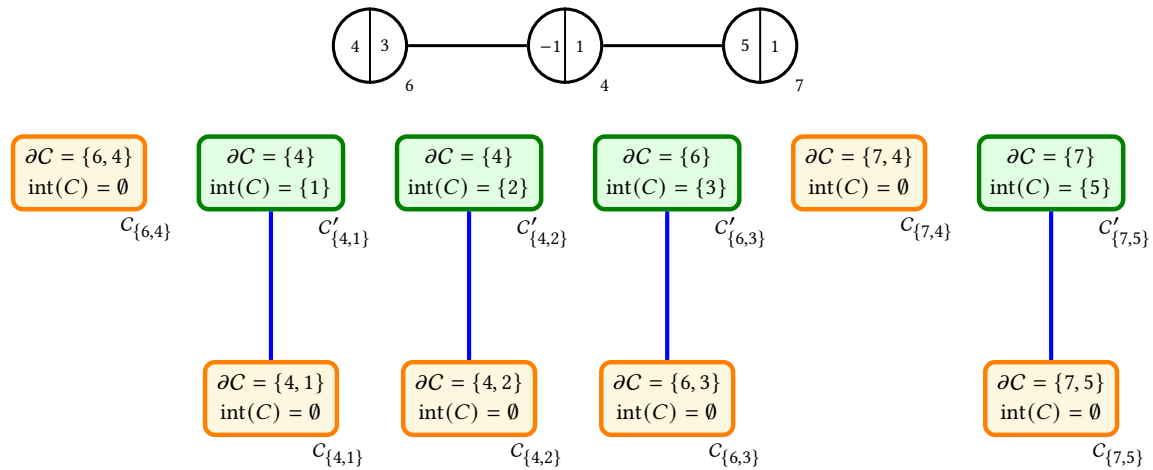
For example, the elementary cluster associated with arc $(4, 1)$ has boundary $\{4, 1\}$ and empty interior. The state $x_4 = 1, x_1 = 0$ is infeasible because the precedence constraint requires $x_4 \leq x_1$. The same initialization is performed independently for all six arcs.

Step 1 – choose the contraction set in the first round. For the contraction logic, arc directions are now ignored. In the current tree T' , the leaves are $\mathcal{L}_{T'} = \{1, 2, 3, 5\}$ and the degree-two vertices are $\mathcal{D}_{T'}^2 = \{6, 7\}$. The algorithm chooses a maximal independent set I_1 satisfying $\mathcal{L}_{T'} \subseteq I_1 \subseteq \mathcal{L}_{T'} \cup \mathcal{D}_{T'}^2$. All four leaves must therefore be selected. Vertex 6 cannot be added because it is adjacent to selected leaf 3, and vertex 7 cannot be added because it is adjacent to selected leaf 5. Hence the maximal independent set is forced: $I_1 = \{1, 2, 3, 5\}$.

(b) First-round maximal independent set $I_1 = \{1, 2, 3, 5\}$.

Step 2 – apply Compress1 to the four selected leaves. Every vertex in I_1 is a leaf, hence every contraction in this round is a Compress1. If a leaf i belongs to a 2-cluster whose other boundary vertex is j , Compress1 moves i from the boundary to the interior and leaves j as the unique boundary vertex. At the level of functions, the state of i is eliminated by taking the maximum over $\sigma_i \in \{0, 1\}$ and adding the contribution $\sigma_i(p_i - \lambda w_i)$.

In the running instance this creates four 1-clusters: one with interior $\{1\}$ and boundary $\{4\}$, one with interior $\{2\}$ and boundary $\{4\}$, one with interior $\{3\}$ and boundary $\{6\}$, and one with interior $\{5\}$ and boundary $\{7\}$. Vertices 1, 2, 3, 5 are therefore removed from the active tree. The active tree becomes the path $6 - 4 - 7$.

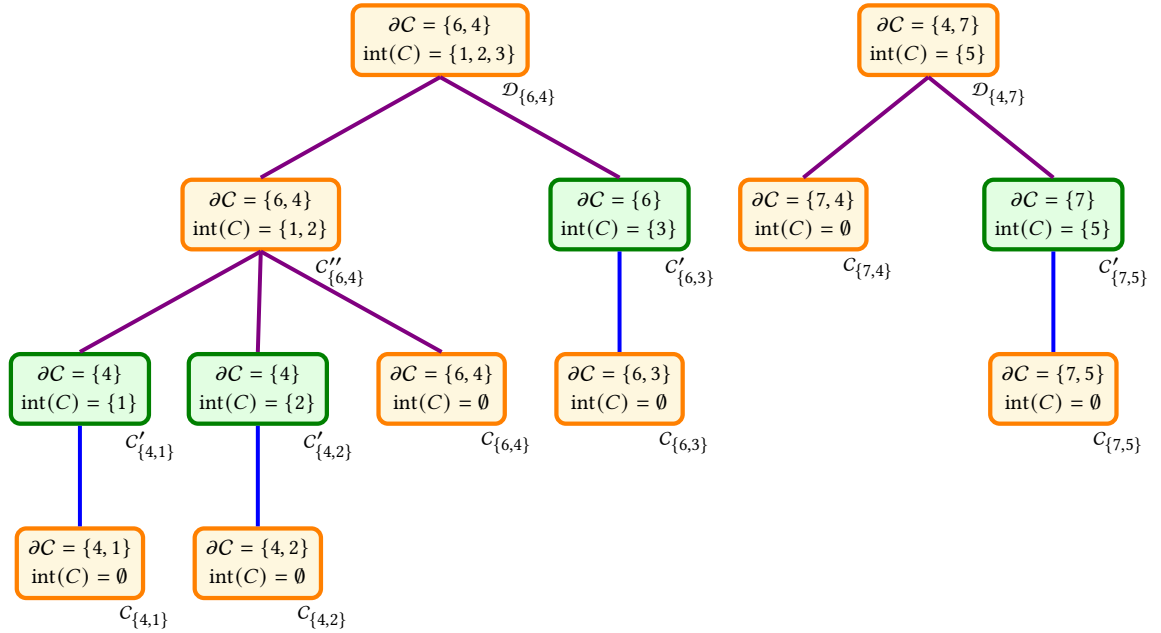
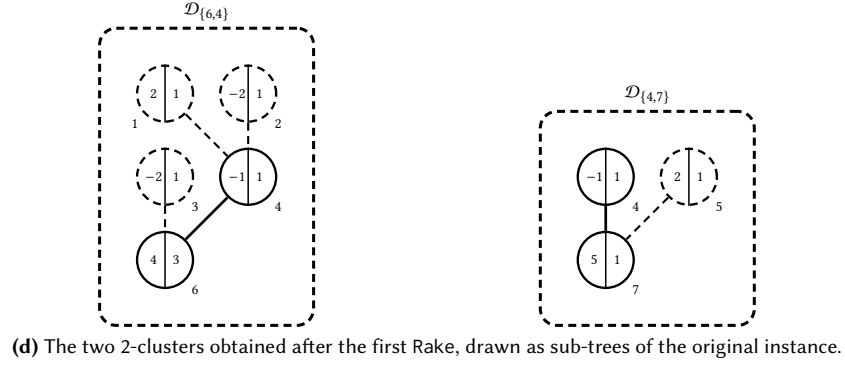


(c) Active tree, and cluster graph after the four first-round Compress1 operations. Each new 1-cluster (green box) is connected by a blue line to the elementary 2-cluster (yellow box) it was compressed from, with the leaf moved from boundary to interior and a prime added to the cluster's name; $C_{\{6,4\}}$ and $C_{\{7,4\}}$ are untouched this round and simply carry over to the next step.

Step 3 – rake the four 1-clusters into the surviving edge clusters. After the compressions, the active tree is $6 - 4 - 7$, but the information associated with the removed leaves is stored in the four 1-clusters. A Rake operation attaches such 1-clusters to a surviving cluster sharing the same boundary vertex. At vertex 4 there are two incident surviving 2-clusters; the algorithm may choose either one when assigning the 1-clusters with boundary $\{4\}$. We follow the same legal choice as in the manuscript illustration: the clusters containing vertices 1 and 2 are raked into the $\{6, 4\}$ side.

On the $\{6, 4\}$ side this is actually carried out as two Rake operations in sequence, one for each boundary vertex that still holds pending 1-clusters. The first Rake is triggered by vertex 4, which is the boundary of the two singleton-boundary clusters $C'_{\{4,1\}}$ and $C'_{\{4,2\}}$: it attaches both of them to $C_{\{6,4\}}$, producing the intermediate 2-cluster $C''_{\{6,4\}}$ with boundary $\{6, 4\}$ and interior $\{1, 2\}$. The second Rake is then triggered by vertex 6, which is the boundary of the singleton-boundary cluster $C'_{\{6,3\}}$: it attaches $C'_{\{6,3\}}$ to $C''_{\{6,4\}}$, only at this point producing the 2-cluster $\mathcal{D}_{\{6,4\}}$ of the next figure, with boundary $\{6, 4\}$ and interior $\{1, 2, 3\}$. On the $\{4, 7\}$ side a single Rake, triggered by vertex 7, attaches $C'_{\{7,5\}}$ directly to $C_{\{7,4\}}$.

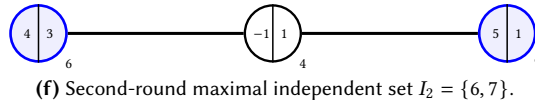
The result is a 2-cluster $\mathcal{D}_{\{6,4\}}$ with boundary $\{6, 4\}$ and interior $\{1, 2, 3\}$, and a 2-cluster $\mathcal{D}_{\{4,7\}}$ with boundary $\{4, 7\}$ and interior $\{5\}$. Importantly, the active tree is still the path $6 - 4 - 7$; Rake changes cluster information but does not remove an active vertex.



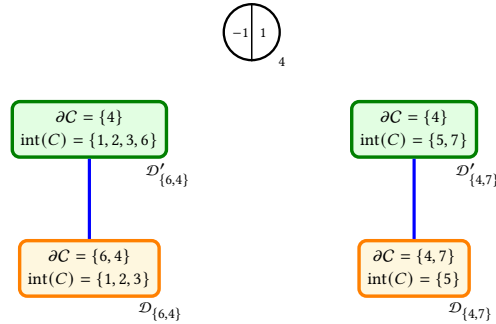
This concludes the first iteration of the main cycle in lines 14–51 of Algorithm 7. The next iteration begins with the active tree $6 - 4 - 7$ and the two 2-clusters $\mathcal{D}_{\{6,4\}}$ and $\mathcal{D}_{\{4,7\}}$ at the top of Figure (e).

Step 4 – choose the contraction set in the second round. The active tree is now the three-vertex path $6 - 4 - 7$. Its leaves are 6 and 7, while vertex 4 has degree two. Since the maximal independent set must contain all leaves, both 6 and 7 are selected. Vertex 4 cannot then be added because it is adjacent to both selected leaves. Consequently $I_2 = \{6, 7\}$.

This also explains why Compress2 is never executed in this particular running example. In a general round, a degree-two vertex belonging to the maximal independent set would be processed by Compress2, combining its two incident 2-clusters and moving that degree-two vertex to the interior. Here, no degree-two vertex is ever selected.

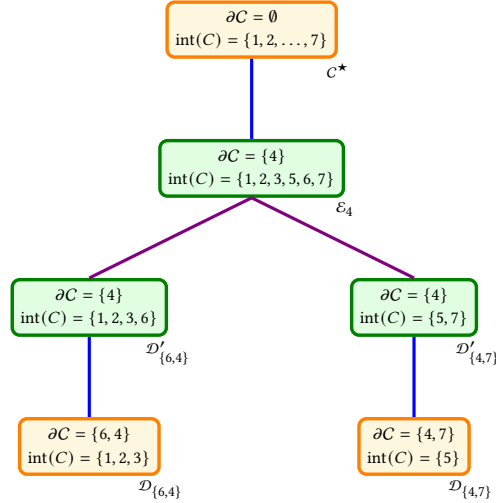


Step 5 – apply Compress1 to vertices 6 and 7. The two selected vertices are leaves of the active path, so each is processed by Compress1. Compressing 6 in $\mathcal{D}_{\{6,4\}}$ produces the 1-cluster $\mathcal{D}'_{\{6,4\}}$ with boundary $\{4\}$ and interior $\{1, 2, 3, 6\}$. Compressing 7 in $\mathcal{D}_{\{4,7\}}$ produces the 1-cluster $\mathcal{D}'_{\{4,7\}}$ with boundary $\{4\}$ and interior $\{5, 7\}$. Only vertex 4 remains active.



Step 6 – form the root of the cluster tree and then compute φ . Both $\mathcal{D}'_{\{6,4\}}$ and $\mathcal{D}'_{\{4,7\}}$ are 1-clusters with the same singleton boundary $\{4\}$, so this is precisely the special case at the end of Definition 3.6. Applying this special Rake merges $\mathcal{D}'_{\{6,4\}}$ and $\mathcal{D}'_{\{4,7\}}$ into the 1-cluster $\mathcal{E}_4 = (V, \{4\})$, the root of the cluster tree $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ stored by the bottom-up phase.

This concludes the second and last iteration of the main cycle in lines 14–51 of Algorithm 7: now $|V'| = 1$, the while loop terminates, and execution moves to line 53, where one final Compress1 eliminates the last boundary vertex 4 and takes $\mathcal{E}_4$ to $C^* = (V, \emptyset)$, with $\varphi(\lambda) = f_{C^*}^0(\lambda)$. It is important to distinguish the root $\mathcal{E}_4$, which is stored as part of $\mathcal{G}$, from the boundary-free cluster C^* , which only serves to obtain φ .



(h) Panel (g) completed with the special Rake of Definition 3.6 merging $\mathcal{D}'_{\{6,4\}}$ and $\mathcal{D}'_{\{4,7\}}$ into the root $\mathcal{E}_4 = (V, \{4\})$, immediately followed by the final Compress1 that eliminates boundary vertex 4 and yields $C^* = (V, \emptyset)$, ending the bottom-up phase.

Step 7 – start the top-down phase and recover the threshold of vertex 4. The top-down phase starts from $\mathcal{E}_4 = (V, \{4\})$, not from C^* . The two possible states of the root boundary vertex define

$$g_0^4(\lambda) = f_{\mathcal{E}_4}^0(\lambda), \quad g_1^4(\lambda) = f_{\mathcal{E}_4}^1(\lambda) + (p_4 - \lambda w_4).$$

The threshold θ_4 is the unique value at which $g_1^4(\lambda) - g_0^4(\lambda)$ changes sign. For this instance, $\theta_4 = 2/3$.

Step 8 – traverse the special-rake arcs. The root $\mathcal{E}_4$ was created by a Rake. Rake arcs do not correspond to a vertex being moved into the interior of a cluster, so the top-down phase does not compute a new threshold while traversing them. It simply descends from $\mathcal{E}_4$ to its two children $\mathcal{D}'_{\{6,4\}}$ and $\mathcal{D}'_{\{4,7\}}$. The already known threshold $\theta_4 = 2/3$ remains available as a boundary condition for the next computations.

Step 9 – reverse the second-round Compress1 operations. Cluster $\mathcal{D}'_{\{6,4\}}$ was obtained by compressing vertex 6 from $\mathcal{D}_{\{6,4\}}$, while $\mathcal{D}'_{\{4,7\}}$ was obtained by compressing vertex 7 from $\mathcal{D}_{\{4,7\}}$. When a Compress1 arc is traversed top-down, the threshold of the surviving boundary vertex is already known. For vertex 6 this known boundary is vertex 4, so the state $\sigma_4(\lambda)$ is fixed and the algorithm compares

$$g_x^6(\lambda) = f_{\mathcal{D}_{\{6,4\}}}^{\sigma_4(\lambda), x}(\lambda) + x(p_6 - \lambda w_6), \quad x \in \{0, 1\}.$$

The sign change gives $\theta_6 = 1/2$. The analogous computation for vertex 7 gives $\theta_7 = 2/3$.

Step 10 – continue downward to the first-round clusters: vertices 1 and 2. Inside $\mathcal{D}_{\{6,4\}}$, rake arcs are first traversed without computation until the two 1-clusters created by Compress1(1) and Compress1(2) are reached. In both cases the surviving boundary vertex is 4, whose threshold is already known. Reversing the corresponding Compress1 arcs and comparing the two state functions yields $\theta_1 = 2$ and $\theta_2 = 2/3$.

Step 11 – recover the thresholds of vertices 3 and 5. The remaining first-round 1-clusters have boundaries $\{6\}$ and $\{7\}$. Their boundary thresholds have already been recovered in Step 9. Reversing Compress1(3) with the state of vertex 6 fixed by $\theta_6 = 1/2$ gives $\theta_3 = 1/2$. Reversing Compress1(5) with the state of vertex 7 fixed by $\theta_7 = 2/3$ gives $\theta_5 = 2$.

Step 12 – group equal thresholds to obtain the closure layers. At this point a threshold has been recovered for every vertex. The values are

$$\theta_1 = 2, \quad \theta_2 = \frac{2}{3}, \quad \theta_3 = \frac{1}{2}, \quad \theta_4 = \frac{2}{3}, \quad \theta_5 = 2, \quad \theta_6 = \frac{1}{2}, \quad \theta_7 = \frac{2}{3}.$$

Only now are vertices grouped into closure layers. Equal thresholds define the same layer, and the distinct threshold values are ordered decreasingly. Hence

$$\mathcal{I}_1 = \{1, 5\}, \quad \mathcal{I}_2 = \{2, 4, 7\}, \quad \mathcal{I}_3 = \{3, 6\},$$

with layer thresholds 2, $2/3$, and $1/2$, respectively. This last step emphasizes the conceptual distinction between the two phases: the bottom-up phase constructs cluster functions, whereas the top-down phase recovers vertex thresholds; the closure-layer partition is obtained from those thresholds only at the end.

v	1	2	3	4	5	6	7
θ_v	2	$\frac{2}{3}$	$\frac{1}{2}$	$\frac{2}{3}$	2	$\frac{1}{2}$	$\frac{2}{3}$

$\mathcal{I}_1 = \{1, 5\}$ $\lambda = 2$	>	$\mathcal{I}_2 = \{2, 4, 7\}$ $\lambda = \frac{2}{3}$	>	$\mathcal{I}_3 = \{3, 6\}$ $\lambda = \frac{1}{2}$
---	---	--	---	---

(n) Recovered thresholds and induced closure-layer partition.