BOOLEAN DIMENSION AND TREE-WIDTH

STEFAN FELSNER, TAMÁS MÉSZÁROS, AND PIOTR MICEK

Abstract. The dimension of a partially ordered set P is an extensively studied parameter. Small dimension allows succinct encoding. Indeed if P has dimension d, then to know whether $x < y$ in P it is enough to check whether $x < y$ in each of the d linear extensions of a witnessing realizer. Focusing on the encoding aspect Nešetřil and Pudlák defined the boolean dimension so that P has boolean dimension at most d if it is possible to decide whether $x < y$ in P by looking at the relative position of x and y in only d permutations of the elements of P.

Our main result is that posets with cover graphs of bounded tree-width have bounded boolean dimension. This stays in contrast with the fact that there are posets with cover graphs of tree-width three and arbitrarily large dimension.

As a corollary, we obtain a labeling scheme of size $\log(n)$ for reachability queries on n-vertex digraphs with bounded tree-width. Our techniques seem to be very different from the usual approach for labeling schemes which is based on divide-and-conquer decompositions of the input graphs.

(S. Felsner) Institut für Mathematik, Technische Universität Berlin,

(T. Mészáros) Institut für Mathematik, Freie Universität Berlin,

(P. Micek) Theoretical Computer Science Department, Faculty of Mathematics and Computer Science, Jagiellonian University, Kraków and Freie Universität Berlin,

E-mail addresses: felsner@math.tu-berlin.de, meszaros.tamas@fu-berlin.de, piotr.micek@tcs.uj.edu.pl.

Key words and phrases. posets, tree-width, reachability, labeling scheme.

Piotr Micek was partially supported by the National Science Center of Poland, grant no. 2015/18/E/ST6/00299.

Tamás Mészáros was supported by the DRS POINT Fellowship Program at Freie Universität Berlin.

1. Introduction

Partially ordered sets, called *posets* for short, are combinatorial structures with applications in various mathematical fields, e.g. set theory, topology and algebra, and theoretical computer science. The most important measure of a poset's complexity is its dimension. The *dimension* $\dim(P)$ of a poset P is the least integer d such that points of P can be embedded into $\mathbb{R}^d$ in such a way that $x < y$ in P if and only if the point of x is below the point of y with respect to the product order of $\mathbb{R}^d$. Though this definition justifies the geometric intuition behind the notion of dimension usually we work with the following equivalent. A *realizer* of a poset P is a set $\{L_1, \ldots, L_d\}$ of linear extensions of P such that for every $x, y \in P$

$$x \leqslant y \text{ in } P \iff (x \leqslant y \text{ in } L_1) \wedge \cdots \wedge (x \leqslant y \text{ in } L_d),$$

and the dimension of P is the minimum size of its realizer.

This reveals the second nature of the dimension: Realizers provide a way of succinctly encoding posets. Indeed if a poset is given with a realizer witnessing dimension d, then a query of the form "is $x \leqslant y$?" can be answered by looking at the relative position of x and y in each of the d linear extensions of the realizer. This application motivates the following more powerful encoding of posets proposed by Nešetřil and Pudlák [8] in 1989.

The *boolean realizer* of a poset P is a set of permutations $\{L_1, \ldots, L_d\}$ of elements of P for which there exists a d-ary boolean formula ϕ such that

$$x \leqslant y \iff \phi((x \leqslant y \text{ in } L_1), \ldots, (x \leqslant y \text{ in } L_d)) = 1,$$

and the *boolean dimension* of P, denoted $\mathrm{bdim}(P)$, is the minimum size of its boolean realizer. Clearly, for every poset P we have

$$\mathrm{bdim}(P) \leqslant \dim(P).$$

The usual dimension of a poset on n elements may be linear in n. The so called, *standard example* S_n, for $n \geqslant 2$, is a poset on $2n$ elements $a_1, \ldots, a_n, b_1, \ldots, b_n$ with $a_i \leqslant b_j$ in S_n if and only if $i \neq j$, and with no other comparabilites (see Figure 1). It was already observed in the seminal paper by Dushnik and Miller [3] that $\dim(S_n) = n$. It is a nice little exercise to show that $\mathrm{bdim}(S_n) \leqslant 4$ for every n. In general, Nešetřil and Pudlák show that boolean dimension of posets on n elements is $O(\log n)$. They also provide an easy counting argument showing that there are posets on n elements with boolean dimension at least $c\, n \log n$.

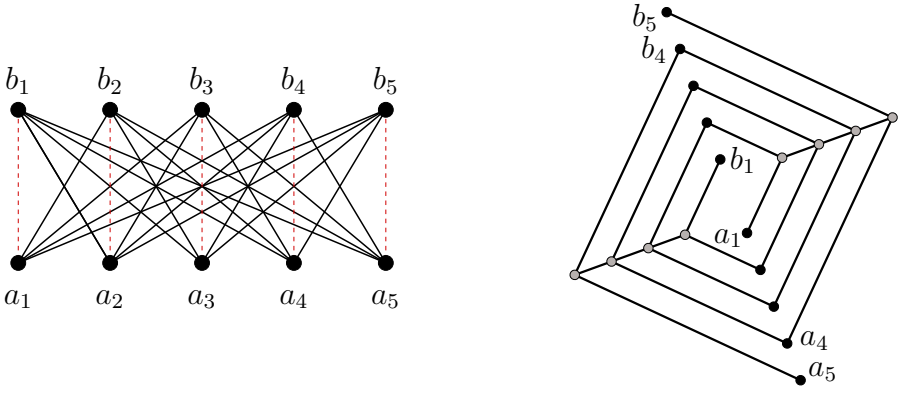

Figure 1. The standard example S_5 (left). Kelly's planar poset containing an induced S_5 (right).

Let P be a poset, the *cover graph* of P is the graph on the elements of P with edge set $\{xy \mid x < y \text{ in } P \text{ and there is no } z \text{ with } x < z < y \text{ in } P\}$. A poset is *planar* if it has a planar diagram, i.e., its cover graph has a non-crossing upward drawing in the plane. This means that every edge xy with $x < y$ is drawn as a curve that goes monotonically up from the point of x to the point of y. Somewhat unexpectedly planar posets have arbitrarily large dimension. Kelly [7] gave a construction that embeds a standard example as a subposet into a planar poset (see Figure 1). Another property of Kelly's construction is that the cover graphs of resulting posets have tree-width (and even path-width) at most 3. Still, the boolean dimension of standard examples and Kelly's construction is at most 4. There is a beautiful open problem posed in [8] that remains a challenge with essentially no progress over the years.

Problem 1 (Nešetřil and Pudlák (1989))**.** *Is the boolean dimension of planar posets bounded?*

Nešetřil and Pudlák suggested an approach for the negative resolution of this problem that involves an auxiliary Ramsey-type problem for planar posets. From the positive side, Brightwell and Franciosa [2] proved that spherical posets with a least element have bounded boolean dimension, contrary to ordinary dimension. Recently, Trotter and Walczak [11] studied the interplay of boolean dimension with yet another concept the *local dimension*. They propose constructions of families of posets where one of the parameters stays bounded while the other goes to infinity.

The contribution of our paper is the following result.

Theorem 2. *Posets with cover graphs of bounded tree-width have bounded boolean dimension.*

The usual dimension is known to be at most 3 for posets with cover graphs being forests (Trotter, Moore [12]) and at most 1276 for posets with cover graphs of tree-width 2 (Joret et al. [5]). As mentioned before Kelly's examples have tree-width 3 and arbitrarily large dimension. This certifies that boolean realizers are capable to represent natural classes of posets that are out of reach in the default setting.

It is tempting to speculate, if a similar result to Theorem 2 hold for broader classes of sparse posets. Besides planar posets, it might be true even for posets whose cover graphs exclude a fixed graph as a (topological) minor, or even whose cover graphs belong to a fixed class with bounded expansion. This line resembles the series of papers where poset dimension is bounded in terms of poset height for posets with cover graphs that are planar [9], or have bounded tree-width [4], or exclude a fixed graph as a minor [14], or belong to a fixed class with bounded expansion [6].

A boolean realizer is very much related with a labeling scheme for reachability queries. Let G be a digraph. A *labeling* of G is a non-negative integer function L that assigns a *label* $L(v, G)$ to each vertex v of G. Formally we see labels as binary strings. A *reachability decoder* is a function f that given two labels λ_1, λ_2 returns a binary value $f(\lambda_1, \lambda_2)$. Now, a pair (L, f) is a *reachability labeling*, if for every u, v in G

$$\text{there is a path from } u \text{ to } v \text{ in } G \quad \iff \quad f(L(u, G), L(v, G)) = 1.$$

Let $|L(v, G)|$ be the *length* of $L(v, G)$ as a binary string. The *size* of a labeling scheme (L, f) for a graph G is $\max_{v \in G} |L(v, G)|$.

Bonichon et al. [1] developed a reachability labeling of size $O(\log n)$ for trees and caterpillars on n vertices. Thorup [10] presented a reachability labeling of size $O(\log^2 n)$ for planar digraphs on n vertices. It remains a challenge to verify the following.

Problem 3. *Is there a reachability labeling of size $O(\log n)$ for planar digraphs on n vertices?*

Our construction of a boolean realizer directly implies a labeling scheme.

Theorem 4. *For every $k \geqslant 1$, there is a reachability labeling of size $O(\log n)$ for digraphs on n vertices with tree-width at most k.*

2. Preliminaries

2.1. **Tree- and path-decompositions.** Let G be a graph. A *tree-decomposition* of G is a pair $(T, (B_t)_{t\in T})$ where T is a tree and $(B_t)_{t\in T}$ is a family of subsets of $V(G)$, satisfying:

(i) for each $v \in V(G)$ there exists $t \in V(T)$ with $v \in B_t$;
(ii) for every edge $uv \in E(G)$ there exists $t \in V(T)$ with $u, v \in B_t$;
(iii) for each $v \in V(G)$, if $v \in B_t \cap B_{t''}$ for some $t, t'' \in V(T)$, and t' lies on the path in T between t and t'', then $v \in B_{t'}$.

By property (iii) we have that for every vertex $v \in V(G)$ the vertices $t \in V(T)$ for which $v \in B_t$ form a subtree of T, called the *subtree of* v.

The quality of a tree-decomposition $(T, (B_t)_{t\in T})$ is usually measured by its width, i.e. the maximum of $|B_t| - 1$ over all $t \in V(T)$. Then the *tree-width* $\mathrm{tw}(G)$ of G is the minimum width of a tree-decomposition of G. The concept of *path-decomposition* and *path-width* can be defined similarly by restricting T above to be a path.

We will call a tree-decomposition *nice* if for every t, t' adjacent in T we have $|B_t - B_{t'}| \leqslant 1$, moreover we have equality only if the degree of both t and t' is at most two. Note that in case of path-decompositions the second assumption is automatically guaranteed. It is an easy observation that every graph has a nice tree-decomposition witnessing its tree-width.

Given a tree-decomposition $(T, (B_t)_{t\in T})$ of a graph G of width k, one can color the vertices of G with $k + 1$ colors in such a way that if two vertices $v, v' \in V(G)$ appear together in B_t for some $t \in V(T)$ then they have different color. We will call such a coloring of G with a fixed tree-decomposition the *ground-coloring* of G. (We will have more colorings within the proof.) When a ground coloring is fixed, for a vertex $t \in V(T)$ and $i \in [k + 1]$ we will call the vertex of ground-color i in B_t the *representative* of the ground-color i at t, and denote it by $\mathrm{repr}_i(t)$ (if such a vertex exists).

2.2. **Branching programs.** To show that a poset P has $\mathrm{bdim}(P) \leqslant d$ one should provide d permutations $\pi_1, \ldots, \pi_d$ of the elements of P and a boolean formula $\phi(\xi_1, \ldots, \xi_d)$. It will be convenient to describe the formula ϕ as a branching program.

We think of a branching program $\mathcal{B}$ as a rooted tree. The nodes of the tree are subprograms. The task of a subprogram at node N is to return a boolean value to its parent, this boolean value is called the *evaluation* of N. The actions taken by the subprogram depend on the values of some of the input variables $\xi_1, \ldots, \xi_d$ of $\mathcal{B}$. On the basis of these values the program at N may call some of its children for evaluation and ignore some other children. The

evaluation of N is a function of the values returned from the children and the values of some further input variables.

The evaluation $\mathrm{eval}(\mathcal{B})$ of the branching program $\mathcal{B}(b_1, \ldots, b_d)$ with inputs $b_1, \ldots, b_d$ is the evaluation of its root node.

A branching program $\mathcal{B}$ is said to represent a boolean formula $\phi(\xi_1, \ldots, \xi_d)$ if for all inputs $b_1, \ldots, b_d$ we have $\mathrm{eval}(\mathcal{B}) = \phi(b_1, \ldots, b_d)$. Our informal description of a branching program is not emphasizing the existence of a corresponding boolean formula, however, such a formula exists when the evaluation of each node N is ONLY depending on the input variables and the evaluations of the children.

Therefore we can prove $\mathrm{bdim}(P) \leqslant d$ by describing a branching program that answers queries '$(x < y)$?' using variables $\xi_1, \ldots, \xi_d$ which depend on x being left or right of y in the permutations $\pi_1, \ldots, \pi_d$ respectively.

Next we develop some tools that will be useful to define such a branching program.

2.3. **Tools for partitions.** The tools of this subsection are related to the localization of x and y in a partition.

Tool 1 (Relative membership in a block)**.** *Let A be a set and let $\{C_1, \ldots, C_m\}$ be a partition of A. Then there are two permutations π_1, π_2 of A such that for every distinct $x, y \in A$*

$$\begin{array}{l} x < y \text{ both in } \pi_1 \text{ and } \pi_2 \\ \qquad \text{or} \qquad\qquad \iff \quad x \text{ and } y \text{ belong to different blocks of the partition .} \\ x > y \text{ both in } \pi_1 \text{ and } \pi_2 \end{array}$$

Proof. For $i \in [m]$ let σ_i and σ_i^* be an arbitrary permutation of the elements of C_i and its reversal respectively. Now $\pi_1 = \sigma_1\sigma_2\cdots\sigma_m$ and $\pi_2 = \sigma_1^*\sigma_2^*\cdots\sigma_m^*$ are suitable permutations. □

Tool 2 (Block identification)**.** *Let A be a set and let $\{C_1, \ldots, C_m\}$ be a partition of A. Then there are $3 \cdot \lceil \log m \rceil$ permutations of A such that for every $x, y \in A$ by looking at the order of x and y in this set of permutations the indices i and j such that $x \in C_i$ and $y \in C_j$ can be identified.*

Proof. Setting $r = \lceil \log m \rceil$, we can re-index the sets in the partition by subsets of $[r]$. After possibly adding empty sets to the partition, we may assume that the partition has the form $\{C_H \ : \ H \subseteq [r]\}$. This induces a dual set system $\{D_1, \ldots, D_r\}$ of A, where for each $i \in [r]$ we have $D_i = \bigcup_{i \in H \subseteq [r]} C_H$.

Now fix an arbitrary permutation σ of A. For a subset $B \subseteq A$ let $\sigma(B)$ and $\sigma^*(B)$ denote the restriction of σ to B and its reversal respectively, and let $\overline{B}$ be the complement of B in A. For each $i \in [r]$ set

$$\pi_1^i = \sigma(D_i)\sigma(\overline{D_i}), \quad \pi_2^i = \sigma^*(D_i)\sigma(\overline{D_i}), \quad \pi_3^i = \sigma(\overline{D_i})\sigma(D_i),$$

Suppose $x \in C_{H_x}$ and $y \in C_{H_y}$ for $H_x, H_y \subseteq [r]$. Now for fixed $i \in [r]$ look at the order of the pair (x, y) in the permutations $\pi_1^i, \pi_2^i, \pi_3^i$. Writing a 1 if x comes before y and a 0 otherwise we get $(1, 0, 1)$ or $(0, 1, 0)$ if both are in D_i; $(1, 1, 0)$ if only x is in D_i; $(0, 0, 1)$ if only y is in D_i; and $(0, 0, 0)$ or $(1, 1, 1)$ if neither is in D_i. Hence, boolean formulae can decide whether i is an element of H_x or not and whether i is an element of H_y or not. This shows that the $3r = 3\lceil \log m \rceil$ permutations identify H_x and H_y. □

For $m = 2$ as a direct corollary we obtain the following statement.

Tool 3 (Set membership)**.** *Let A be a set and let $C \subseteq A$. Then there are three permutations of A such that for every distinct $x, y \in A$ by looking at the order of x and y in these three permutations one can decide whether x (resp. y) belongs to C or not.*

2.4. **Red-Green detection on a path in a tree.** The tools in this subsection are more involved then the ones earlier. Here x and y are vertices of a rooted tree with some edges of the tree being colored with colors RED and GREEN. For the first question here we assume that x is an ancestor of y and ask whether the first colored edge on the path from x to y is RED or GREEN or there is no colored edge on this path.

Given a rooted tree T and a vertex v of T we denote the subtree of T rooted at v as T_v. For two vertices u, v we denote the path from u to v in T as $[u, v]$.

Tool 4. *Let T be a rooted tree with some edges colored either with* RED *or* GREEN*. Let x, y be two distinct vertices in T such that x is an ancestor of y, and let π be a permutation of vertices of T produced by Algorithm 1. Then*

$$y < x \text{ in } \pi \iff \text{the first colored edge on the path from } x \text{ to } y \text{ in } T \text{ is RED.}$$

Algorithm 1: RED-GREEN detection for x being an ancestor of y in T

```
Procedure process(v)
    C(v) = {u ∈ T_v | the path [v,u] has no colored edges}
    R(v) = {u ∈ T_v | the path [v,u] has a unique colored edge,
                      this edge is the last edge and it is RED }
    G(v) = {u ∈ T_v | the path [v,u] has a unique colored edge,
                      this edge is the last edge and it is GREEN }
    for u ∈ R(v) do
        process(u)
    L = list of all vertices in C(v) in a topological order in T
    append L to π
    for u ∈ G(v) do
        process(u)
π = empty
process(root(T))
return π
```

Proof. We start the proof with two simple observations.

- During the algorithm every vertex of T is a member of $C(v)$ for a unique $v \in T$.
- A call of process(v) appends all elements of T_v to π before returning.

Now consider two vertices x, y in T such that x is an ancestor of y in T. We distinguish three cases.

(i) The path $[x, y]$ has no colored edges.
(ii) The first colored edge (v, w) on the path $[x, y]$ is GREEN.

(iii) The first colored edge (v, w) on the path $[x, y]$ is RED.

Let z be the node of T for which $x \in C(z)$ during the algorithm.

Case (i) In this case $y \in C(z)$ and in the topological order L of the elements in $C(z)$ we have x before y, therefore, $x < y$ in π.

Case (ii) We append the list L containing x before calling process(w). However as process(w) is the one which puts y in the permutation, we again have $x < y$ in π.

Case (iii) We call process(w), and so put y in the permutation, before appending the list L containing x, so now we have $y < x$ in π. □

For the second question we assume that our rooted tree T is given with a planar upward drawing with lowest vertex being the root. This implies that at every vertex u there is a fixed left-to-right order of its subtrees. For two vertices u, v in T we say that u *is left of* v if there is a vertex w which has two children $u' \neq v'$ such that $u \in T_{u'}$ and $v \in T_{v'}$ and in the left-to-right order at w the subtree $T_{u'}$ is left of $T_{v'}$. If u, v, w are as above, then we call w the *meet* of u and v and denote it by $u \wedge v$. It is the last common vertex on the paths from the root of T to u and v.

Tool 5. *Let T be a rooted tree with some edges colored either with* RED *or* GREEN. *Let x, y be two distinct vertices in T such that x is left of y in T and let π be a permutation of vertices of T produced by Algorithm 2. Then*

$$y < x \text{ in } \pi \iff \text{the first colored edge on the path from } x \wedge y \text{ to } x \text{ in } T \text{ is RED.}$$

Algorithm 2: RED-GREEN detection for x left of y in T

```
1  Procedure process(v)
2      append v to π
3      for each w child of v in T taken in left-to-right-order do
4          if (v, w) is uncolored then
5              process(w)
6          if (v, w) is RED then
7              S.push(w)
8          if (v, w) is GREEN then
9              S.push(v)              // marking the beginning of the local stack of v
10             process(w)
11             while S.top() ≠ v do
12                 process(S.pop())
13             S.pop()                // taking off the marker
14 π = ∅
15 S = ∅
16 process(root(T))
17 while S ≠ ∅ do
18     process(S.pop())
19 return π
```

Proof. Again we begin with some simple observations.

- For each vertex v in T there is a unique call of process(v), i.e., the resulting π is a permutation of the vertices of T.
- If u and v are vertices of T such that there are vertices u' and v' with $u \in T_{u'}$ and $v \in T_{v'}$ and u' is on the current stack when process(v') is called, then $v < u$ in π.
- If $[u, u']$ is a path of uncolored edges in T and v is a vertex such that u is an ancestor of v and u' is left of v, then $u' < v$ in π.
- If (v, w) is GREEN, then the local stack makes the procedure behave as if Algorithm 2 had been called for the tree T_w. In particular all the vertices of T_w appear in a consecutive block of π.

Let $m = x \wedge y$ be the meet of x and y, we distinguish three cases.

(i) The path $[m, x]$ has no colored edges.
(ii) The first colored edge (v, w) on the path $[m, x]$ is GREEN.
(iii) The first colored edge (v, w) on the path $[m, x]$ is RED.

Case (i) $x < y$ in π follows from the third item above.

Case (ii) From the third item above we obtain $v < y$ in π. The call of process(w) appends all vertices of T_w to π and the vertex x is among them so that $x < y$ in π.

Case (iii) w is put on the stack and remains on the stack until all children of m have been processed. If v' is the child of m with $y \in T_{v'}$, then w is on the stack when process(v') is called. The second item above shows that in this case $y < x$ in π. □

3. THE PROOF

Let P be a poset with $\mathrm{tw}(\mathrm{cover}(P)) \leqslant k$. Fix a nice tree-decomposition $(T, (B_t)_{t \in V(T)})$ of width at most k. We imagine the tree T to be rooted and being drawn upwards with lowest vertex the root. In particular at every vertex t there is a fixed left-to-right order of its subtrees. For $z \in P$ let $\mathrm{root}(z)$ denote the root (i.e. the lowest point) of the subtree of z in T. Since our tree-decomposition is nice we know that all the $\mathrm{root}(z)$ for $z \in P$ are distinct.

To prepare for the design of the branching program we next define a companion structure of the tree-decomposition.

3.1. **The accompanying dag.** Fix a *ground-coloring* $\mathrm{col} : P \to [k+1]$. For a vertex $t \in V(T)$ and an element $x \in P$ such that $\mathrm{root}(x)$ is an ancestor of t in T we define the vector $\mathrm{vec}(x, t)$ of length $k+1$ such that for $i \in \{1, \ldots, k+1\}$ the i^{th} coordinate is

$$\mathrm{vec}_i(x,t) = \begin{cases} < & \text{if } x < \mathrm{repr}_i(t) \text{ in } P \\ > & \text{if } x > \mathrm{repr}_i(t) \text{ in } P \\ \| & \text{if } x \| \mathrm{repr}_i(t) \text{ in } P \\ = & \text{if } x = \mathrm{repr}_i(t) \\ * & \text{if } \mathrm{repr}_i(t) \text{ is undefined} \end{cases}.$$

For a vertex $t \in V(T)$ we define the active set of t as

$$\mathrm{Active}(t) = \{\mathrm{vec}(x,t) \ : \ x \in P \text{ such that } \mathrm{root}(x) \text{ is an ancestor of } t\}.$$

Here we remark that above we allow $\mathrm{root}(x) = t$. Note that for every $t \in V(T)$ there are at most 5^{k+1} such possible vectors, so in particular $|\,\mathrm{Active}(t)| \leqslant 5^{k+1}$.

Next we define an auxiliary directed graph D accompanying our fixed tree-decomposition and ground-coloring, that will play a key role in the remaining argument. The vertex set of D is $\bigcup_{t\in V(T)}\{t\}\times \text{Active}(t)$. The edges in D are of the form: from $(t, \text{vec}(x,t))$ to $(t', \text{vec}(x,t')$ for every $t, t' \in T$ with t being a parent of t' in T and every $x \in P$ with $\text{root}(x)$ being an ancestor of t in T.

Lemma 5. *For every two distinct vertices $t, t' \in T$ such that t is an ancestor of t' in T and every $v \in \text{Active}(t)$ there is exactly one $v' \in \text{Active}(t')$ such that there is a path from (t, v) to (t', v') in D.*

Proof. Let's first start with two adjacent vertices t, t' in T with t being the parent of t'. First, we show that for every two elements $x, y \in P$ with $\text{root}(x)$ and $\text{root}(y)$ being ancestors of t in T, and $\text{vec}(x,t) = \text{vec}(y,t)$, we have

$$\text{vec}(x,t') = \text{vec}(y,t'). \tag{1}$$

Since our tree-decomposition is nice we either have $B_{t'} \subseteq B_t$ or there is exactly one element $z' \in P$ such that $z' \in B_{t'} - B_t$ (this clearly means that $\text{root}(z') = t'$).

If $B_{t'} \subseteq B_t$ then for every $z \in B_{t'}$, say of a ground-color i, we have $\text{repr}_i(t') = z = \text{repr}_i(t)$. Therefore, whenever $\text{repr}_i(t')$ is defined, we have $\text{vec}_i(x,t') = \text{vec}_i(x,t) = \text{vec}_i(y,t) = \text{vec}_i(y,t')$. Thus $\text{vec}(x,t') = \text{vec}(y,t')$ as required.

Otherwise let z' be the unique element in $B_{t'} - B_t$, say with $\text{col}(z') = j$. For every ground-color $i \neq j$, if $\text{repr}_i(t')$ exists then as before $\text{vec}_i(x,t') = \text{vec}_i(y,t')$. Now, suppose that $\text{vec}_j(x,t') = \text{`} < \text{`}$. This by definition means that $x < z'$ in P. Let us consider a cover chain of this relation in P. By the properties of our tree-decomposition this chain must contain an element z with $x \leqslant z < z'$ in P such that z is in both B_t and $B_{t'}$. Say $\text{col}(z) = i_0$. As clearly $i_0 \neq j$ we already know that $\text{vec}_{i_0}(x,t') = \text{vec}_{i_0}(y,t')$, hence we conclude $y \leqslant z < z'$ in P and therefore $\text{vec}_j(y,t') = \text{`} < \text{`}$ as required. Along the very same line we can prove that if $\text{vec}_j(x,t') = \text{`} > \text{`}$, then $\text{vec}_j(y,t') = \text{`} > \text{`}$. As $z' \neq x, y$ and $\text{repr}_{i_0}(t') = z'$ we know that $\text{vec}_j(x,t')$ and $\text{vec}_j(x,t')$ can take only three possible values `` < '', `` > '', `` || '' and as the argument was symmetric for x and y, this completes the proof of (1).

The lemma now follows with an easy induction on the distance between t and t' in T. We proved before the statement for t, t' at distance 1 in T. For larger distances, we pick the last vertex t'' before t' on a unique t-t' path in T. Now by the induction hypothesis there is a unique $v'' \in \text{Active}(t'')$ such that there is a path from (t, v) to (t'', v'') in D. And again there is a unique $v' \in \text{Active}(t')$ such that there is a path from (t'', v'') to (t', v') in D. This proves the lemma. □

For $z \in P$ let source-vec(z) denote the unique vertex $(\text{root}(z), v)$ in $\{\text{root}(z)\} \times \text{Active}(\text{root}(z))$ with no in-edge, i.e. v is the unique vector in $\text{Active}(\text{root}(z))$ which has a $'='$ in coordinate $\text{col}(z)$. Further let dest-vec$(< z)$ (resp. dest-vec$(> z)$) denote the set of all vertices $(\text{root}(z), v)$ of D with $v_{\text{col}(z)} = '<'$ (resp. $v_{\text{col}(z)} = '>'$).

By Lemma 5, for every $t \in T$, $v \in \text{Active}(t)$ and every descendant t' of t in T there is a unique directed path in D, denoted by $\text{path}((t,v), t')$, that starts at (t, v) and for every t'' on the path from t to t' intersects $\text{Active}(t'')$ in exactly one element. If $v = \text{vec}(x,t)$ for some element $z \in P$ with $\text{root}(z)$ an ancestor of t, then this element is $(t'', \text{vec}(z,t''))$. To simplify notation, if $(t,v) = \text{source-vec}(z)$ for some element $z \in P$, the instead of $\text{path}(\text{source-vec}(z), t')$ we will simply write $\text{path}(z, t')$.

Given a subgraph F of D we define the projection $\mathrm{proj}(F)$ of F to T as the subgraph of T obtained by restricting the elements of F to their first component. Formally, if a vertex (t, v), $v \in \mathrm{Active}(t)$ belongs to F, then t belongs to $\mathrm{proj}(F)$, and if $((t, v), (t', v'))$ is an edge of F, then (t, t') is an edge of $\mathrm{proj}(F)$.

Next we define a coloring of the vertices and edges of D with colors from $\Gamma = [5]^{k+1}$.

Algorithm 3: Γ-coloring of D

Initialization let $t_0, t_1, \ldots, t_m$ be the vertices of T in left-to-right topological order and color all vertices of D in $\mathrm{Active}(t_0)$ with distinct colors
for $i = 0, \ldots, m$ **do**
  color each edge of D leaving $\mathrm{Active}(t_i)$ with the color of its source vertex in $\mathrm{Active}(t_i)$
  for *each child* t *of* t_i **do**
    color each vertex of D in $\mathrm{Active}(t)$ according to the following rule
    if *the in-degree of the vertex is positive* **then**
      color it with the smallest of the colors of its incoming edges
    if *all vertices with positive in-degree have been colored and there exists a vertex with no incoming edge* **then**
      color it with a color different from all those used on $\mathrm{Active}(t)$ so far

For $z \in P$ let $\text{source-col}(z)$ denote the color of $\text{source-vec}(z)$ under this coloring. Further let $\text{dest-col}(< z)$ (resp. $\text{dest-col}(> z)$) be the set of colors of vertices in $\text{dest-vec}(< z)$ (resp. $\text{dest-vec}(> z)$).

Note that the colors (of the vertices) along any directed path in D are always non-increasing. We say that a color sequence $(\gamma_0 > \gamma_1 > \cdots > \gamma_\ell)$ is the *signature* of a directed path Q in D if $\{\gamma_0, \gamma_1, \ldots, \gamma_\ell\}$ is the set of colors of the vertices in Q. In fact this means that Q is a concatenation $Q_0 Q_1 \cdots Q_\ell$ of subpaths where all vertices in Q_i have color γ_i for every i.

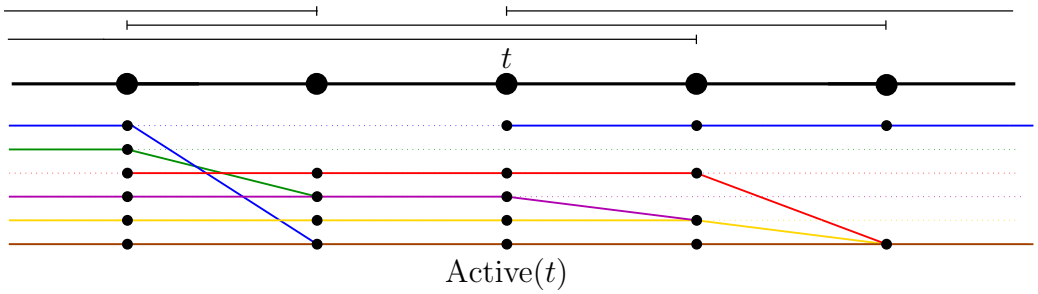

FIGURE 2. A sketch of a path-decomposition and an associated dag. Vectors of the same color are horizontally aligned such that colors increase vertically.

The following two lemmas are the key to the usage of the dag in the branching program. They provide necessary and sufficient conditions for two queried elements x and y of P to be in relation $x < y$ in P.

For an element (t, v) in D with $t \in V(T)$ and $v \in \mathrm{Active}(t)$, we define $\mathrm{tree}((t, v))$ of all elements of D that are reachable from (t, v) with a directed path in D. Indeed, as suggested by the notation, $\mathrm{tree}((t, v))$ is a directed subtree in D rooted at (t, v). By Lemma 5 $\mathrm{proj}(\mathrm{tree}((t, v)))$ is exactly T_t, i.e. the whole subtree rooted at t in T.

To simplify notation, if $(t, v) = \text{source-vec}(z)$ for some element $z \in P$ then we simply write $\text{tree}(z)$ for this subtree.

Lemma 6 (x is below y)**.** *Let x, y be two distinct elements in P such that* $\text{root}(x)$ *is an ancestor of* $\text{root}(y)$ *in T. Then*

$$x < y \text{ in } P \iff \text{tree}(\text{source-vec}(x)) \text{ contains a vector from } \text{dest-vec}(< y).$$

Proof. Note that $\text{tree}(\text{source-vec}(x))$ contains elements of the form $(t, \text{vec}(x, t))$ for all t below $\text{root}(x)$ in T. If $t = \text{root}(y)$, then $\text{vec}_{\text{col}(y)}(x, \text{root}(y)) \in \{<, >, \|\}$ indicates the comparability status of x and y. By definition this status is $'<'$ if and only if $(\text{root}(y), \text{vec}(x, \text{root}(y))) \in \text{dest-vec}(< y)$. □

Lemma 7 (x is left of y)**.** *Let x, y be two distinct elements in P such that* $\text{root}(x)$ *is left of* $\text{root}(y)$ *in T and let $m_{xy} = \text{root}(x) \wedge \text{root}(y)$. Then*

$$x < y \text{ in } P \iff \text{there is a vector } v \in \text{Active}(m_{xy}) \text{ such that } \text{tree}((m_{xy}, v)) \text{ contains}$$

- *a vector from* $\text{dest-vec}(> x)$ *and*
- *a vector from* $\text{dest-vec}(< y)$.

Proof. First we prove the backwards implication. Suppose that there is a vector $v \in \text{Active}(m_{xy})$ such that $\text{tree}((m_{xy}, v))$ intersects both $\text{dest-vec}(> x)$ and $\text{dest-vec}(< y)$. Let z be an element of P such that $\text{root}(z)$ is an ancestor of m_{xy} in T and $\text{vec}(z, m_{xy}) = v$. Note that all the elements in $\text{tree}((m_{xy}, v))$ are of the form $\text{vec}(z, t)$ for all t descendants of m_{xy} in T. This yields that $\text{vec}(z, \text{root}(x)) \in \text{dest-vec}(> x)$ and $\text{vec}(z, \text{root}(y)) \in \text{dest-vec}(> x)$, which implies $x < z < y$ in P as required.

Now assume that $x < y$ in P. There is a chain $x = z_0 < \cdots < z_\ell = y$ with all $z_i < z_{i+1}$ being cover relations in P. By the properties of the tree-decomposition we know that the union of subtrees of $z_1, \ldots, z_\ell$ form a subtree of T. In particular there must be z_i such that $z_i \in B_{m_{xy}}$ Since $\text{root}(z_i)$ is an ancestor of m_{xy} the vectors $\text{vec}(z_i, \text{root}(x))$ and $\text{vec}(z_i, \text{root}(y))$ are defined. From $x < z_i < y$ it follows that $\text{tree}(z_i)$ contains vectors from $\text{dest-vec}(> x)$ and $\text{dest-vec}(< y)$. Let $v = \text{vec}(z_i, m_{xy})$ and note that $\text{tree}((m_{x,y}, v))$ is a subtree of $\text{tree}(\text{source-vec}(z_i))$ that still intersects both $\text{dest-vec}(> x)$ and $\text{dest-vec}(< y)$. This completes the proof. □

3.2. **The initial nodes of the branching process.** The colored dag D provides the structure needed to devise a branching program $\mathcal{B}$ and a set of permutations $\pi_1, \ldots, \pi_d$ (with d being a function of k only) such that for every query '$(x < y)$?', when we set $b_i = [x < y \text{ in } \pi_i]$ then the evaluation of $\mathcal{B}$ with the variables being set to $b_1, \ldots, b_d$ is 1 if and only if $x < y$ in P. As discussed before this will prove $\text{bdim}(P) \leqslant d$.

The root node N_{root} of $\mathcal{B}$ first of all quickly separates the case $x = y$. This can be done with two initial permutations where one is reversal of the other. If $x \leqslant y$ in both permutations then it must be that $x = y$ and N_{root} outputs 1. Therefore, in the following we assume that the queried elements are distinct.

So now the root node N_{root} is used to branch into the appropriate subprograms depending on the relative position of $\text{root}(x)$ and $\text{root}(y)$ in the tree T.

Since we assume a nice tree-decomposition we know that $\text{root}(x) \neq \text{root}(y)$, hence their relative position is one of the following four:

- *x is below y*,
  i.e., root(x) is an ancestor of root(y).
- *x is above y*,
  i.e., root(y) is an ancestor of root(x).
- *x is left of y*,
  i.e., at their meet node t the subtree of root(x) is left of the subtree of root(y).
- *x is right of y*,
  i.e., at their meet node t the subtree of root(x) is right of the subtree of root(y).

To decide on the relative position it is enough to look at the relative position of root(x) and root(y) in the left-first-search and the right-first-search order of the vertices of T.

Note that vertices of T are not quite elements of P, hence, for example the left-first-search order of the vertices of T is not a valid permutation for the branching program. However, restricting the left-first-search order of the vertices of T to those vertices which are of the form root(z) for some $z \in P$ we obtain a permutation of P which can be used to decide on the relative position of x and y.

In the remaining part of this paper, just as above, instead of constructing permutations of P we frequently construct permutations of the vertices of T and understand without saying that the corresponding permutation of P is the induced permutation of the vertices which are of the form root(z) for some $z \in P$.

Below, in Subsection 3.4 and 3.5 we describe the branching (sub)programs for the cases where x is below y and where x is left of y. The other two cases are symmetric to these and can thus be ommitted. As a warmup, in Subsection 3.3 we deal with the special case where the tree-decomposition is in fact a path-decomposition, i.e., T is a path.

3.3. **The case where T is a path.** Throughout this subsection T is a path. We think of this path drawn horizontally with the root being its leftmost vertex, i.e., the edges of the underlying dag are directed left-to-right. We also assume that for the vertices queried we have root(x) left of root(y) on T. This instance can be identified with a single permutation, and the other case when root(x) is right of root(y) is symmetric.

One advantage of being in the path case is that for every $z \in P$ there is a unique directed path, denoted by path(z), in D that starts in source-vec(z). This path contains the vertices $(t, \mathrm{vec}(z,t))$ for all t to the right of root(z) in T.

A direct corollary of Lemma 6 in this case is the following.

Corollary 8. *Let x, y be different elements in P such that* root(x) *is left of* root(y) *in T. Then $x < y$ in P if and only if there exists a color sequence $(\gamma_0 > \gamma_1 > \cdots > \gamma_\ell)$ such that $\gamma_0 =$* source-col(x)*, $\gamma_\ell \in$* dest-col($< y$) *and the portion of* path(x) *from* source-vec(x) *to* dest-vec($< y$) *has signature $(\gamma_0, \gamma_1, \ldots, \gamma_\ell)$.*

Note that the set of possible signatures has cardinality at most $2^{5^{k+1}}$. This is a function of k, hence, we can color elements z of P with the signature of path(z). The first task of the branching program is to identify the color, i.e., the signature, of x. This can be handled with the Block Identification Tool (Tool 2) using at most $3 \cdot \lceil \log 2^{5^{k+1}} \rceil = 3 \cdot 5^{k+1}$ permutations. We will have a separate subprogram N_S for every possible signature S.

Consider now a fixed signature $S = (s_0 > s_1 > s_2 > \cdots > s_\lambda)$. In the following we describe how to build the subprogram N_S that allows to decide for the comparability status of x, y when S is the signature of path(x).

Let Z_S be the set of all $z \in P$ such that S is the signature of path(z). For $z \in Z_S$ and $i \leqslant \lambda$ we write path$_i(z)$ to denote the the maximal prefix of path(z) with signature $(s_0, s_1, \ldots, s_i)$.

Define $\mathcal{F}_0$ to be the family of projections of all maximal subpaths of D entirely of color s_0 to T. Note that the paths of $\mathcal{F}_0$ are pairwise disjoint. In fact $\mathcal{F}_0$ is the collection of projections of the initial parts of path(z) for all z with source-col$(z) = s_0$.

Starting from $\mathcal{F}_0 = \mathcal{F}_0^\emptyset$ we inductively construct families $\mathcal{F}_i^\alpha$ of paths of T where $1 \leqslant i \leqslant \lambda$ and $\alpha = \alpha_1, \ldots, \alpha_i$ is a binary sequence. The construction is such that the two familes $\mathcal{F}_{i+1}^{\alpha,0}$ and $\mathcal{F}_{i+1}^{\alpha,1}$ depend only on $\mathcal{F}_i^\alpha$.

The following properties of the families are essential:

- For each i and α the paths in $\mathcal{F}_i^\alpha$ are pairwise disjoint.
- For each path Q in $\mathcal{F}_i^\alpha$ there is an element $z \in Z_S$ such that proj(path$_i(z)$) is a postfix of Q. We call z a *witness* for Q and let $W(Q)$ be the set of all witnesses of Q.
- For all $z \in Z_S$ and all i there is some α such that there is a path Q in $\mathcal{F}_i^\alpha$ which has proj(path$_i(x)$) as a postfix.

Given $\mathcal{F}_i^\alpha$ we now show how to construct $\mathcal{F}_{i+1}^{\alpha,0}$ and $\mathcal{F}_{i+1}^{\alpha,1}$. This is done in two steps. First we extend each path Q of $\mathcal{F}_i^\alpha$ to a larger path Q^+. In the second step, after possibly removing some of the paths, we break the family of extended paths into two subfamilies such that the paths in each of these subfamilies are disjoint.

For each such Q consider any witness z and define Q^+ as $Q \cup$ proj(path$_{i+1}(z)$). Now whenever there are two paths Q_1^+, Q_2^+ with the same right endpoint, then remove the shorter from the collection and note that every $z \in W(Q_1) \cup W(Q_2)$ is a witness for the larger one. Let $\mathcal{F}_i^{\alpha+}$ be the family of the remaining extended paths.

To partition $\mathcal{F}_i^{\alpha+}$ into two disjoint subfamilies we first sort the paths in $\mathcal{F}_i^{\alpha+}$ by increasing right endpoints. Each $Q^+ \in \mathcal{F}_i^{\alpha+}$ is composed of a path from the disjoint family $\mathcal{F}_i^\alpha$ and the projection of a path of color s_{i+1} in D. Note that these latter paths of color s_{i+1} are also pairwise disjoint. Therefore a path $Q^+ \in \mathcal{F}_i^{\alpha+}$ can only intersect with its predecessor and its successor in the order by right endpoints, see Figure 3. Now let $\mathcal{F}_{i+1}^{\alpha,0}$ consist of the path at even positions in the sorted order and let $\mathcal{F}_{i+1}^{\alpha,1}$ consist of those at odd positions. Both families are disjoint.

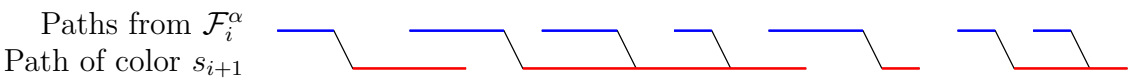

FIGURE 3. Some elements of $\mathcal{F}_i^{\alpha+}$.

By construction each path in a family $\mathcal{F}_i^\alpha$ has a nonempty set of witnesses and since each $z \in Z_S$ is the witness for a path in $\mathcal{F}_0$ it is then necessarily also a witness of a path in $\mathcal{F}_i^\alpha$ for all i and some α. This shows that the construction results in families of paths with the three desired properties.

For each $z \in Z_S$ there is a unique binary sequence $\alpha(z)$ of length $\lambda \leqslant 5^{k+1}$ such that for each prefix $\alpha_i(z)$ of length i of this sequence it holds that z is a witness for some path in

$\mathcal{F}_i^{\alpha_i(z)}$. With the Block Identification Tool (Tool 2) we can pass the query for the pair x, y to a subprogram $N_{S,\alpha}$ with $\alpha = \alpha(x)$. As the total number of possible binary sequences is 2^λ this can be done by using $3 \cdot \lceil \log 2^\lambda \rceil = 3\lambda \leqslant 3 \cdot 5^{k+1}$ permutations.

$N_{S,\alpha}$ has a child $N_{S,\alpha,\ell}$ for each $1 \leqslant \ell \leqslant \lambda$ and it evaluates to 1 if and only if one of its children evaluates to 1. The node $N_{S,\alpha,\ell}$ checks whether the right end point of $\mathrm{proj}(\mathrm{path}_{\ell-1}(x))$ is to the left of $\mathrm{root}(y)$ and $\mathrm{proj}(\mathrm{path}_\ell(x))$ contains $\mathrm{root}(y)$. If one of these test fails then $N_{S,\alpha,\ell}$ evaluates to 0. If both tests are passed we know that the vertex $(\mathrm{root}(y), v)$ of D contained in $\mathrm{path}(x)$ has color s_ℓ. To see whether $x < y$ we thus only have to check whether $s_\ell \in \text{dest-col}(< y)$. This is application of the Set Membership Tool (Tool 3), and can be done using three permutations. If all

What remains is to describe how to decide whether the right endpoint of $\mathrm{proj}(\mathrm{path}_\ell(x))$ is to the left or to the right of $\mathrm{root}(y)$. Recall that $\mathrm{root}(x)$ is left of $\mathrm{root}(y)$ and by construction of the path families $\mathcal{F}_\ell^{\alpha_\ell(x)}$ contains some Q witnessed by x, i.e., there is a path Q_x which contains $\mathrm{proj}(\mathrm{path}_\ell(x))$ as a postfix. Therfore it is enough to check whether $\mathrm{root}(y) \in Q_x$. This can be done using the first version of the Red-Green Detection Tool (Tool 4). Color the edges on the right of the right endpoints of paths in $\mathcal{F}_\ell^{\alpha_\ell(x)}$ Red. By Proposition 4 the positions of x and y in a single permutation of P allows to decide whether there is a Red edge on the path from $\mathrm{root}(x)$ to $\mathrm{root}(y)$ on T.

This completes the description of the branching structure for the special case where T is a path.

3.4. **The case when x is below y in T.** Recall that when $\mathrm{root}(z)$ is an ancestor of t in T with $\mathrm{path}(z,t)$ we denote the directed path in D which starts at source-vec(z) and ends in some vertex from $\{t\} \times \mathrm{Active}(t)$. Clearly $\mathrm{path}(x, \mathrm{root}(y))$ is a path in $\mathrm{tree}(x)$ and $\mathrm{tree}(x)$ contains a vector from dest-vec$(< y)$ if and only if $\mathrm{path}(x, \mathrm{root}(y))$ contains such a vector. Therefore, we have the following direct corollary of Lemma 6.

Corollary 9. *Let x, y be different elements in P such that x is below y in T. Then $x < y$ in P if and only if there exists a color sequence $(\gamma_0 > \gamma_1 > \cdots > \gamma_\ell)$ such that $\gamma_0 = \text{source-col}(x)$, $\gamma_\ell \in \text{dest-col}(< y)$ and $\mathrm{path}(x, \mathrm{root}(y))$ has signature $(\gamma_0, \gamma_1, \ldots, \gamma_\ell)$.*

Corollaries 9 and 8 look almost identical at the first glance, however, the path and also the signature now depend on y in a much stronger way. Therefore, the task of identifying the relevant signature for a pair x, y is a harder one in this case.

The branching program for this case starts with a major parallel node $N_{x \text{ below } y}$ that has as a child N_S for every S a possible signature of a path in D. Recall that the set of possible signatures has cardinality at most $2^{5^{k+1}}$. Again the output of $N_{x \text{ below } y}$ is simply an alternative of outputs of all N_S.

Now we fix a legal signature $S = (s_0 > s_1 > s_2 > \cdots > s_\lambda)$ and we proceed with a description of the node N_S. The node N_S is designed in a way that N_S outputs a 1 if and only if the signature of $\mathrm{path}(x, \mathrm{root}(y))$ is S and $s_\lambda \in \text{dest-col}(< y)$.

The subprogram for N_S starts with a verification if $s_0 = \text{source-col}(x)$. This is done with the Set Membership Tool (Tool 3) using 3 permutations. The node N_S returns a 0 if $s_0 \neq \text{source-col}(x)$ and otherwise it continues with the next step.

At this point we know that s_0 is the first color of the signature of $\mathrm{path}(x, \mathrm{root}(y))$. We are going to proceed step by step verifying if consecutive colors of this signature agrees with S.

This check, similarly as in the case of T being a path, is based on families of disjoint subtrees of T which are defined inductively.

We start with some helpful notation. First let (t, v) be a vertex of color γ in D. We will denote by tree $((t, v), \gamma)$ the directed subtree of D which is rooted at (t, v) and contains all vertices of color γ in D which can be reached from (t, v) on a directed path of color γ. Let $T(t, \gamma)$ be proj(tree $((t, v), \gamma)$).

As a basis for the inductive construction we have a family $\mathcal{F}(\gamma)$ for each $\gamma \in [5^{k+1}]$. This family is defined to be

$$\mathcal{F}(\gamma) = \{T(\text{root}(z), \gamma) \mid \text{for every } z \in P \text{ with source-col}(z) = \gamma\}$$

It comes from the basic properties of the dag that trees in $\mathcal{F}(\gamma)$ are pairwise disjoint.

Claim. *The family $\mathcal{F}(\gamma)$ is a family of pairwise disjoint subtrees of T, for every $\gamma \in [5^{k+1}]$.*

Proof. Suppose in contrary that there are distinct $Q, Q' \in \mathcal{F}(\gamma)$ and $t \in Q \cap Q'$. Let $\text{root}(Q) = \text{root}(z)$ and $\text{root}(Q') = \text{root}(z')$ for some elements $z, z' \in P$. By the construction we have source-col$(z) = \gamma =$ source-col(z'). Let (t, v) be the vertex in D colored with γ. Since t is in Q and Q' there are directed paths in D from source-vec(z) to (t, v) and from source-vec(z') to (t, v). In particular $\text{root}(z)$ and $\text{root}(z')$ must be both ancestors of t in T. This implies that one of $\text{root}(z)$ and $\text{root}(z')$ is an ancestor of the other in T, say $\text{root}(z)$ is an ancestor $\text{root}(z')$. Now the directed path from source-vec(z) to (t, v) in D has all vertices colored with γ. Since $\text{root}(z')$ is on the directed path from $\text{root}(z)$ to t in T the path must go through a vertex in $\{\text{root}(z')\} \times \text{Active}(\text{root}(z'))$ colored with γ, which is source-vec(z'). This is a contradiction as source vertices have no incoming edges in D. □

Starting from $\mathcal{F}_0^{\emptyset} = \mathcal{F}(s_0)$ we inductively construct families $\mathcal{F}_i^{\alpha}$ of trees of T where $1 \leqslant i \leqslant \lambda$ and $\alpha = \alpha_1, \ldots, \alpha_i$ is a ternary sequence, i.e., $\alpha_j \in \{0, 1, 2\}$ for all j. An essential property of the families is that:

For each i and α the trees in $\mathcal{F}_i^{\alpha}$ are pairwise disjoint.

If α is a ternary sequence of length i (with $0 \leqslant i < \lambda$), then we define $\mathcal{F}_{i+1}^{\alpha,0} = \mathcal{F}(s_{i+1})$. To construct the two other familes $\mathcal{F}_{i+1}^{\alpha,1}$ and $\mathcal{F}_{i+1}^{\alpha,2}$, which depend on $\mathcal{F}_i^{\alpha}$, we use an intermediate family $\mathcal{F}_i^{\alpha,+}$. The family $\mathcal{F}_i^{\alpha,+}$ is produced by Algorithm 4. For the description of this algorithm we need additional notation. For $\gamma \in [5^{k+1}]$ we call a vertex t in T a *γ-break* if vertices in $\{t\} \times \text{Active}(t)$ in D are not colored with γ. For an edge (t, t') of T we say that γ *merges into* β at (t, t') if

(i) $\gamma > \beta$,
(ii) there is a γ-colored vertex $(t, v) \in \text{Active}(t)$ and a β-colored vertex $(t', v') \in \text{Active}(t')$,
(iii) and there is an edge from (t, v) to (t', v') in D.

The following claim shows that $\mathcal{F}_i^{\alpha,+}$ can be split into two families $\mathcal{F}_{i+1}^{\alpha,1}$ and $\mathcal{F}_{i+1}^{\alpha,2}$ such that each of them consists of pairwise disjoint trees.

Claim. *For every $i \geqslant 0$ and $\alpha \in \{0, 1, 2\}^i$, assuming that the family $\mathcal{F}_i^{\alpha}$ consists of pairwise disjoint subtrees of T, the family $\mathcal{F}_i^{\alpha,+}$, produced by Algorithm 4, can be split into two blocks each consisting of pairwise disjoint subtrees of T.*

Algorithm 4: Construction of a family $\mathcal{F}_i^{\alpha,+}$ given $\mathcal{F}_i^{\alpha}$

$\mathcal{F}_i^{\alpha,+} \leftarrow \emptyset$
for *each* Q *in* $\mathcal{F}_i^{\alpha}$ **do**
  $Q^+ \leftarrow \emptyset, \quad r \leftarrow \mathrm{root}(Q)$
  for *each* t *leaf of* Q *and edge* (t,t') *such that*
    (a) $[r,t]$ *contains a* s_{i+1}*-break*
    (b) s_i *merges into* s_{i+1} *at* (t,t') **do**
    $Q^+ \leftarrow Q^+ \cup [r,t'] \cup T(t', s_{i+1})$
  if $Q^+ \neq \emptyset$ **then**
    add Q^+ to $\mathcal{F}_i^{\alpha,+}$
return $\mathcal{F}_i^{\alpha,+}$

Proof. Consider the family $\mathcal{F}_i^{\alpha,+}$ produced by Algorithm 4. Every tree Q^+ added into $\mathcal{F}_i^{\alpha,+}$ comes from some tree Q in $\mathcal{F}_i^{\alpha,+}$. We split each such Q^+ into two sections. The *primal section* of Q^+ is the part that comes from its origin Q, i.e., $Q^+ \cap Q$, and the *extended section* of Q^+ is the extended part, i.e., $Q^+ \setminus Q$.

We show that for every two distinct trees $Q_1^+, Q_2^+ \in \mathcal{F}_i^{\alpha,+}$ both their primal and extended sections are disjoint respectively.

First, we argue about the primal sections. The primal sections of Q_1^+ and Q_2^+ are contained in their origins Q_1 and Q_2, respectively. But since $Q_1, Q_2 \in \mathcal{F}_i^{\alpha}$ they are disjoint by assumption.

Now consider the extended sections R_1, R_2 of Q_1^+, Q_2^+, respectively. In order to get a contradiction, suppose that $t \in R_1 \cap R_2$. For $j = 1, 2$ let $r_j = \mathrm{root}(Q_j)$ and note that $\mathrm{root}(Q_j^+) = \mathrm{root}(Q_j)$. Since t is in the extended section of Q_j^+ there must be a leaf t_j' of Q_j and an edge (t_j', t_j'') with $t \in T(t_j'', s_{i+1})$, moreover there is a break at some vertex t_j on $[r_j, t_j']$. Since Q_1 and Q_2 are disjoint $[r_1, t_1']$ and $[r_2, t_2']$ are disjoint intervals on the path from the root of T to t. Assuming w.l.o.g. that t_1' is an ancestor of r_2 in T we see that t_2 is an s_{i+1} break in $[t_1', t]$ which is in contradiction to $t \in T(t_i'', s_{i+1})$.

This proves that the extended sections of distinct trees in $\mathcal{F}_i^{\alpha,+}$ are disjoint. Hence, the family $\mathcal{F}_i^{\alpha+}$ induces a chordal graph with clique number two. This chordal graph is 2-colorable. A two coloring induces a partition of $\mathcal{F}_i^{\alpha+}$ into two families such that each consists of pairwise disjoint subtrees of T. □

We are now back and continue the definition of N_S. Recall that we are at the point when we do know that s_0 is the first color of the signature of $\mathrm{path}(x, \mathrm{root}(y))$. We define for every $0 \leqslant i \leqslant \lambda$ and $\alpha \in \{0,1,2\}^i$ a subprogram $W(i,\alpha)$. The branching program is going to visit a sequence of these subprograms, starting with $W(0,\emptyset)$. We always make sure that when $W(i,\alpha)$ is called then the following invariant has been established:

> The sequence $(s_0, \ldots, s_i)$ is a prefix of the signature of $\mathrm{path}(x, \mathrm{root}(y))$. Moreover,
>
> (i) if $(s_0, \ldots, s_i)$ is a *proper* prefix of the signature of $\mathrm{path}(x, \mathrm{root}(y))$ then there is a tree Q in $\mathcal{F}_i^{\alpha}$ such that $\mathrm{root}(x) \in Q$ and t is a leaf of Q where

t is the unique vertex in T such that $\text{path}(x,t)$ is the maximal prefix of $\text{path}(x,\text{root}(y))$ with signature $(s_0,\dots,s_i)$.

(ii) otherwise when the signature of $\text{path}(x,\text{root}(y))$ is equal $(s_0,\dots,s_i)$ there is a tree Q in $\mathcal{F}_i^\alpha$ such that $\text{root}(x),\ \text{root}(y) \in Q$

Note that the invariant holds for $W(0,\emptyset)$ as we already know that source-col$(z) = s_0$ and the tree $T(\text{root}(x), s_0)$, that lies in the family $\mathcal{F}_0^\emptyset = \mathcal{F}(s_0)$, witnesses (i) or (ii).

We proceed with the description of $W(i,\alpha)$ (with $0 \leqslant i < \lambda$ and $\alpha \in \{0,1,2\}^i$). This subprogram consists of two steps.

Step 1 The first task of $W(i,\alpha)$ is to establish whether $\text{path}(x,\text{root}(y))$ has a prefix with signature $(s_0,\dots,s_{i+1})$. This will be done using a single permutation with the Red-Green Detection Tool (Tool 4). For all $Q \in \mathcal{F}_i^\alpha$, all leaves t of Q and all edges (t,t'), t being a parent of t', color the edge Red if s_i merges into s_{i+1} at (t,t'), otherwise color it Green. Looking at the relative positions of x and y in the permutation π generated by the tool the program learns whether the first colored edge on the path from root(x) to root(y) is Red or not. If the answer is not (i.e. the first colored edge is Green or there is no colored edge) $\text{path}(x,\text{root}(y))$ contains no merge from s_i to s_{i+1} and the whole program N_S outputs a 0. On the other hand if the first colored edge is Red, then it has been established that $\text{path}(x,\text{root}(y))$ has a prefix with signature $(s_0,\dots,s_{i+1})$.

Note that if this is the case we know that it the item (i) of the invariant that holds. Indeed, $(s_0,\dots,s_i)$ is a *proper* prefix of $(s_0,\dots,s_{i+1})$ which as we learnt is the prefix of the signature of $\text{path}(x,\text{root}(y))$. Let Q_x be a tree in $\mathcal{F}_i^\alpha$ witnessing the item (i) and let t_x be the leaf of Q on the $\text{path}(x,\text{root}(y))$.

Step 2 The second task of $W(i,\alpha)$ is to decide which subprogram to continue with, i.e. which is the relevant family for the (x,y)-query out of $\mathcal{F}_{i+1}^{\alpha,0}$, $\mathcal{F}_{i+1}^{\alpha,1}$ and $\mathcal{F}_{i+1}^{\alpha,2}$.

The family $\mathcal{F}_{i+1}^{\alpha,0}$ is relevant if there is no s_{i+1}-break on the path from root(x) to the leaf t_x This again can be done using a single permutation with the Red-Green Detection Tool (Tool 4). Color the edges of T as follows. For all $Q \in \mathcal{F}_i^\alpha$, all leaves t of Q and all edges (t,t'), where t is a parent of t', color the edge (t,t') Green. For each vertex $t \in T$ which is a s_{i+1}-break color all edges from t to its children in T with Red (possibly overriding Green). From the permutation π generated by the Red-Green Detection Tool the program learns whether the first colored edge on the path from root(x) to root(y) is Red or not. Note that as root$(y) \notin Q_x$ we necessarily have a colored edge on this path.

If the first colored edge on the path from root(x) to root(y) is Green then there is no s_{i+1}-break on the path from root(x) to t_x in T and the next subprogram called is $W(i+1,\alpha 0)$. Note that the required invariant holds.

If the first colored edge on the path from root(x) to root(y) is Red then there is a s_{i+1}-break on the path from root(x) to t_x in T. In this case we know that $\mathcal{F}_i^{\alpha+}$ contains a tree Q which contains the maximal prefix of $\text{path}(x,\text{root}(y))$ with signature $(s_0,\dots,s_{i+1})$. Now it is up to deciding whether this tree Q belongs to $\mathcal{F}_{i+1}^{\alpha,1}$ or to $\mathcal{F}_{i+1}^{\alpha,2}$. The set of trees in $\mathcal{F}_i^{\alpha+} = \mathcal{F}_{i+1}^{\alpha,1} \cup \mathcal{F}_{i+1}^{\alpha,2}$ has been produced as offsprings of trees in $\mathcal{F}_i^\alpha$, and Q is actually the offspring Q_x^+ of Q_x in this union. Now consider the partition of $\mathcal{F}_i^\alpha$ into three subfamilies, those with an offspring in $\mathcal{F}_{i+1}^{\alpha,1}$, those with an offspring in $\mathcal{F}_{i+1}^{\alpha,2}$, and the rest. Since the trees in $\mathcal{F}_i^\alpha$ are disjoint root(x) is contained only in Q_x which is in one of these three families. With the Block Identification Tool (Tool 2) we can identify, using $3 \cdot \lceil \log 3 \rceil = 6$ permutations, the subfamily containing

Q_x. On the basis of this it can be decided whether $W(i+1, \alpha 1)$ or $W(i+1, \alpha 2)$ has to be called next. Whichever is chosen we have made sure that the invariant holds.

This completes the description of the subprograms $W(i, \alpha)$ for $0 \leqslant i < \lambda$ and $\alpha \in \{0,1,2\}^i$.

It remains to describe the finishing subprograms $W(\lambda, \alpha)$ with $\alpha \in \{0,1,2\}^\lambda$. By the invariant we do know that $S = (s_0, \ldots, s_\lambda)$ is the prefix of the signature of $\mathrm{path}(x, \mathrm{root}(y))$. The subprogram $W(\lambda, \alpha)$ first checks whether the signature of $\mathrm{path}(x, \mathrm{root}(y))$ is actually equal to S. It happens if and only if $\mathrm{root}(x)$ and $\mathrm{root}(y)$ belong to the same tree in $\mathcal{F}^\alpha_\lambda$. We check this with the Relative Membership Tool (Tool 1) using two permutations, where the blocks are sets of elements z of P that have $\mathrm{root}(z)$ in the same Q from $\mathcal{F}^\alpha_\lambda$. If it turns out that $\mathrm{root}(x)$ and $\mathrm{root}(y)$ are in different subtrees from $\mathcal{F}^\alpha_\lambda$ then the whole program N_S outputs 0. If $\mathrm{root}(x)$ and $\mathrm{root}(y)$ lie in the same $Q \in \mathcal{F}^\alpha_\lambda$, $W(\lambda, \alpha)$ is designed to check one more, final, thing: whether $s_\lambda \in \text{dest-col}(< y)$. This is a simple application of the Set Membership Tool (Tool 3). For this we need again 3 permutations. If $s_\lambda \notin \text{dest-col}(< y)$ then N_S one more time outputs 0. Otherwise N_S outputs 1.

3.5. **The case when x is left of y in T.** When designing the corresponding branching program we will closely follow the previous case. Accordingly we start by the appropriate variant of Corollary 9, namely with a direct corollary of Lemma 7. To simplify notation, for the meet of $\mathrm{root}(x)$ and $\mathrm{root}(y)$ we will again write $m_{x,y}$.

Corollary 10. *Let x, y be different elements in P such that x is left of y in T. Then $x < y$ in P if and only if there exist a vector $v \in \mathrm{Active}(m_{x,y})$ and two color sequences $(\gamma_0 > \gamma_1 > \cdots > \gamma_\ell)$, and $(\delta_0 > \delta_1 > \cdots > \delta_m)$ such that $\gamma_0 = \delta_0$, $\gamma_\ell \in \text{dest-col}(> x)$, $\delta_m \in \text{dest-col}(< y)$ and the paths $\mathrm{path}((m_{x,y}, v), \mathrm{root}(x))$, $\mathrm{path}((m_{x,y}, v), \mathrm{root}(y))$ have signatures $(\gamma_0, \gamma_1, \ldots, \gamma_\ell)$ and $(\delta_0, \delta_1, \ldots, \delta_m)$ respectively.*

Now we are ready to define the corresponding branching program. Just like before the root node $N_{x \text{ left of } y}$ is a major parallel node. It has a child $N(S, R)$ for every possible pair of color sequences $S = (s_0 > s_1 > \cdots > s_\lambda)$, $R = (r_0 > r_1 > \cdots > r_\mu)$ such that $s_0 = r_0$. It ouptuts a 1 if and only if there is a vector $v \in \mathrm{Active}(m_{x,y})$ such that $\mathrm{path}((m_{x,y}, v), \mathrm{root}(x))$ has signature $(s_0, s_1, \ldots, s_\lambda)$, $\mathrm{path}(m_{x,y}, v), \mathrm{root}(y))$ has signature $(r_0, r_1, \ldots, r_\mu)$, $s_\lambda \in \text{dest-col}(> x)$ and $r_\mu \in \text{dest-col}(< y)$. The number of such possible pairs is clearly at most $\left(2^{5^{k+1}}\right)^2$.

For given S and R the node $N_{S,R}$ will have two children $N_{x,S}$ and $N_{y,R}$. The task of $N_{x,S}$ will be to check whether there is a vector $v \in \mathrm{Active}(m_{x,y})$ such that $\mathrm{path}((m_{x,y}, v), \mathrm{root}(x))$ has signature $(s_0, s_1, \ldots, s_\lambda)$ and $s_\lambda \in \text{dest-col}(> x)$. The task of $N_{y,R}$ is defined in a symmetric manner. If both of these subprograms succeed then $N_{S,R}$ evaluates to 1, otherwise it evaluates to 0. As the two subprograms at these nodes are by symmetry identical, in what follows we will only focus on the description of the subprogram rooted at $N_{x,S}$.

The subprogram for $N_{x,S}$ makes use of the families $\mathcal{F}^\alpha_i$ of trees introduced in Subsection 3.4. It starts with checking whether there is a vector $v \in \mathrm{Active}(m_{x,y})$ of Γ-color s_0, which happens if and only if $m_{x,y}$ is a vertex of some tree $Q \in \mathcal{F}(s_0)$. This can be verified by the extended Red-Green Detection Tool (Tool 5) with a single permutation. Thus, color some of the edges in T as follows: For every $Q \in \mathcal{F}(s_0)$ and every vertex t of Q color all edges going from t to some of its children t' in T with color Red and all other edges Green. Then $m_{x,y}$ is a vertex of some tree $Q \in \mathcal{F}(s_0)$ if and only if the first colored edge on the path from $m_{x,y}$ to $\mathrm{root}(x)$ is Red. If the answer to this question is no, we return 0. Otherwise $N_{x,S}$ proceeds step by

step verifying whether the consecutive colors of the signature of path$((m_{x,y}, v), \text{root}(x))$ agree with S.

From now on, to simplify notation, for the dag vertex $(m_{x,y}, v)$ we will use that shorthand notation $m^*_{x,y}$.

Following the argumentation from Subsection 3.4 we define now for every $0 \leqslant i \leqslant \lambda$ and $\alpha \in \{0,1,2\}^i$ a subprogram $W(i,\alpha)$. The branching program is going to visit a sequence of these subprograms starting with $W(0,\emptyset)$. We always make sure that when $W(i,\alpha)$ is called then the following invariant has been established:

> The sequence $(s_0,\dots,s_i)$ is a prefix of the signature of path$(m^*_{x,y}, \text{root}(x))$. Moreover,
>
> (i) if $(s_0,\dots,s_i)$ is a *proper* prefix of the signature of path$(m^*_{x,y}, \text{root}(x))$ then there is a tree Q^* in $\mathcal{F}^\alpha_i$ such that $m_{x,y}$ is in Q^* and t is a leaf of Q^* where t is the unique vertex in T such that path$(m^*_{x,y}, t)$ is the maximal prefix of path$(m^*_{x,y}, \text{root}(x))$ with signature $(s_0,\dots,s_i)$.
>
> (ii) otherwise when the signature of path$(m^*_{x,y}, \text{root}(x))$ is equal $(s_0,\dots,s_i)$, there is a tree Q in $\mathcal{F}^\alpha_i$ such that $m_{x,y}$ and root(x) are both in Q.

Note that the invariant holds for $W(0,\emptyset)$ as we already know that the Γ-color of $m^*_{x,y}$ is s_0 and the tree $T(\text{root}(z), s_0) \in \mathcal{F}^\emptyset_0 = \mathcal{F}(s_0)$ contains tree$(m^*_{x,y}, s_0)$ as a subtree for some $z \in P$, and so witnesses (i) or (ii).

We proceed with the description of $W(i,\alpha)$ (with $0 \leqslant i < \lambda$ and $\alpha \in \{0,1,2\}^i$). This subprogram consists of two steps.

Step 1 The first task of $W(i,\alpha)$ is to establish whether path$(m^*_{x,y}, \text{root}(x))$ has a prefix with signature $(s_0,\dots,s_{i+1})$. This will be done using a single permutation with the extended Red-Green Detection Tool (Tool 5). Thus, color some of the edges in T as follows: For all $Q \in \mathcal{F}^\alpha_i$, all leaves t of Q and all edges (t,t'), t being a parent of t', color the edge Red if s_i merges into s_{i+1} at (t,t'), otherwise color it Green. Looking at the relative positions of x and y in the permutation π generated by the tool the program learns whether the first colored edge on the path from $m_{x,y}$ to root(x) is Red or not. If the answer is no (i.e. the first colored edge is Green or there is no colored edge), then path$(m^*_{x,y}, \text{root}(x))$ contains no merge from s_i to s_{i+1} and the whole program $N_{x,S}$ outputs a 0. On the other hand if the first colored edge on this path is Red, then it has been established that path$(m^*_{x,y}, \text{root}(x))$ has a prefix with signature $(s_0,\dots,s_{i+1})$.

Note that at this point if $N_{x,S}$ is still running then we know that item (i) of the invariant holds. Indeed, $(s_0,\dots,s_i)$ is a *proper* prefix of $(s_0,\dots,s_{i+1})$ which as we learnt is the prefix of the signature of path$(m^*_{x,y}, \text{root}(x))$. Let Q^* be a tree in $\mathcal{F}^\alpha_i$ witnessing the item (i) and let t^* be the leaf of Q^* on path$(m^*_{x,y}, \text{root}(x))$. In particular, we know that root$(y) \notin Q^*$.

Step 2 The second task of $W(i,\alpha)$ is to decide which subprogram to continue with, i.e. which is the relevant family for the (x,y)-query out of $\mathcal{F}^{\alpha,0}_{i+1}$, $\mathcal{F}^{\alpha,1}_{i+1}$ and $\mathcal{F}^{\alpha,2}_{i+1}$.

The family $\mathcal{F}^{\alpha,0}_{i+1}$ is relevant if there is no s_{i+1}-break on the path from $m_{x,y}$ to the leaf t^* in Q^*. This again can be done using a single permutation with the extended Red-Green Detection Tool (Tool 5). Color the edges of T as follows: For all $Q \in \mathcal{F}^\alpha_i$, all leaves t of Q and all edges (t,t'), where t is a parent of t', color the edge (t,t') Green. For each vertex $t \in T$ which is an s_{i+1}-break color all edges from t going to its children in T with

RED (possibly overriding GREEN). From the permutation π generated by the extended RED-GREEN Detection Tool the program learns whether the first colored edge from $m_{x,y}$ to root(x) is RED or not. Note that as root$(x) \notin Q^*$ we necessarily have a colored edge on this path.

If the first colored edge on the path from $m_{x,y}$ to root(x) is GREEN then there is no s_{i+1}-break on the path from $m_{x,y}$ to t^*-path in T and the next subprogram called is $W(i+1, \alpha 0)$. Note that the required invariant holds.

If the first colored edge on the path from $m_{x,y}$ to root(x) is RED then there is an s_{i+1}-break on the path from $m_{x,y}$ to t^* in T. In this case we know that $\mathcal{F}_i^{\alpha+}$ contains a tree Q^+ which contains the maximal prefix of path$(m^*_{x,y}, \text{root}(x))$ with signature $(s_0, \dots, s_{i+1})$. Now it is up to deciding whether this tree Q^+ belongs to $\mathcal{F}_{i+1}^{\alpha,1}$ or to $\mathcal{F}_{i+1}^{\alpha,2}$. The set of trees in $\mathcal{F}_i^{\alpha+} = \mathcal{F}_{i+1}^{\alpha,1} \cup \mathcal{F}_{i+1}^{\alpha,2}$ has been produced as an offspring of trees in $\mathcal{F}_i^{\alpha}$, and Q^+ is actually the offspring $Q^{*,+}$ of Q^* in this union. Now consider the partition of $\mathcal{F}_i^{\alpha}$ into three subfamilies, those with an offspring in $\mathcal{F}_{i+1}^{\alpha,1}$, those with an offspring in $\mathcal{F}_{i+1}^{\alpha,2}$, and the rest. Since the trees in $\mathcal{F}_i^{\alpha}$ are disjoint $m_{x,y}$ is contained only in Q^* which is in one of these three families. With a repeated application of the extended RED-GREEN Detection Tool (Tool 5) we can identify, using only two permutations, the subfamily containing Q^*. First take the subfamily containing those trees from $\mathcal{F}_i^{\alpha}$ whose offspring is in $\mathcal{F}_{i+1}^{\alpha,1}$. For every tree Q in this family and every vertex t of Q color all edges going from t to its children in T with RED and color all the other edges GREEN. From the permutation provided by the tool the program learns whether the first edge on the path from $m_{x,y}$ to root(x) is RED or not. The answer is yes exactly if $m_{x,y}$ is in on of the trees from the first subfamily. After possibly repeating this for the second subfamily, we will know in which of them is Q^*.

On the basis of this $W(i, \alpha)$ decides whether $W(i+1, \alpha 1)$ or $W(i+1, \alpha 2)$ has to be called next. Whichever is chosen we have made sure that the invariant holds.

This completes the description of the subprograms $W(i, \alpha)$ for $0 \leqslant i < \lambda$ and $\alpha \in \{0,1,2\}^i$.

It remains to describe the finishing subprograms $W(\lambda, \alpha)$ with $\alpha \in \{0,1,2\}^\lambda$. By the invariant we do know that $S = (s_0, \dots, s_\lambda)$ is the prefix of the signature of path$(m^*_{x,y}, \text{root}(x))$. The subprogram $W(\lambda, \alpha)$ first checks whether the signature of path$(m^*_{x,y}, \text{root}(x))$ is actually equal to S. It happens if and only if $m_{x,y}$ and root(x) belong to the same tree in $\mathcal{F}_\lambda^\alpha$. We check this again with the extended RED-GREEN Detection Tool (Tool 5) using one permutation. Color all edges of T that do not belong to any tree $Q \in \mathcal{F}_\lambda^\alpha$ with color RED and keep all the other edges uncolored. Then from the permutation constructed by the tool the program learns whether the first edge on the path from $m_{x,y}$ to root(x) is RED or not. In our case this is the same as asking whether there is a RED edge on this path or not, and so the answer is no exactly if $m_{x,y}$ and root(x) belong to the same tree in $\mathcal{F}_\lambda^\alpha$.

If it turns out that $m_{x,y}$ and root(x) are in different subtrees from $\mathcal{F}_\lambda^\alpha$ then the whole program $N_{x,S}$ outputs 0. If they lie in the same $Q \in \mathcal{F}_\lambda^\alpha$, $W(\lambda, \alpha)$ is designed to check one more, final, thing, namely whether whether $s_\lambda \in$ dest-col$(> x)$. This is a simple application of the Set Membership Tool (Tool 3). For this we need 3 permutations. If $s_\lambda \notin$ dest-col$(> x)$ then $N_{x,S}$ outputs again 0, otherwise it outputs 1.

References

[1] Nicolas Bonichon, Cyril Gavoille, and Arnaud Labourel. Short labels by traversal and jumping. In *6th Czech-Slovak International Symposium on Combinatorics, Graph Theory, Algorithms and Applications*, volume 28 of *Electron. Notes Discrete Math.*, pages 153–160, 2007.

[2] Graham R. Brightwell and Paolo Giulio Franciosa. On the Boolean dimension of spherical orders. *Order*, 13(3):233–243, 1996.

[3] Ben Dushnik and Edwin W. Miller. Partially ordered sets. *Amer. J. Math.*, 63:600–610, 1941.

[4] Gwenaël Joret, Piotr Micek, Kevin G. Milans, William T. Trotter, Bartosz Walczak, and Ruidong Wang. Tree-width and dimension. *Combinatorica*, 36(4):431–450, 2016. arXiv:1301.5271.

[5] Gwenaël Joret, Piotr Micek, William T. Trotter, Ruidong Wang, and Veit Wiechert. On the dimension of posets with cover graphs of treewidth 2. *Order*, to appear. arXiv:1406.3397.

[6] Gwenaël Joret, Piotr Micek, and Veit Wiechert. Sparsity and dimension. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2016)*, 2016. arXiv:1507.01120.

[7] David Kelly. On the dimension of partially ordered sets. *Discrete Math.*, 35:135–156, 1981.

[8] J. Nešetřil and P. Pudlák. A note on Boolean dimension of posets. In *Irregularities of partitions (Fertőd, 1986)*, volume 8 of *Algorithms Combin. Study Res. Texts*, pages 137–140. Springer, Berlin, 1989.

[9] Noah Streib and William T. Trotter. Dimension and height for posets with planar cover graphs. *European J. Combin.*, 35:474–489, 2014.

[10] Mikkel Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *J. ACM*, 51(6):993–1024, 2004.

[11] William T. Trotter and Bartosz Walczak. Boolean dimension and local dimension, 2017. arXiv:1705.09167, accepted at EuroComb 2017.

[12] William T. Trotter, Jr. and John I. Moore, Jr. The dimension of planar posets. *J. Combinatorial Theory Ser. B*, 22(1):54–67, 1977.

[13] Bartosz Walczak. Minors and dimension. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2015)*, pages 1698–1707, 2015. arXiv:1407.4066.

[14] Bartosz Walczak. Minors and dimension. *J. Combin. Theory Ser. B*, 122:668–689, 2017. arXiv:1407.4066.