Linear MIM-Width of Trees *

Svein Høgemo, Jan Arne Telle, and Erlend Raa Vågset

Department of Informatics, University of Bergen, Norway.
{svein.hogemo, jan.arne.telle, erlend.vagset}@uib.no

Abstract. We provide an $O(n \log n)$ algorithm computing the linear maximum induced matching width of a tree and an optimal layout.

1 Introduction

The study of structural graph width parameters like tree-width, clique-width and rank-width has been ongoing for a long time, and their algorithmic use has been steadily increasing [12, 18]. The maximum induced matching width, denoted MIM-width, and the linear variant LMIM-width, are graph parameters having very strong modelling power introduced by Vatshelle in 2012 [20]. The LMIM-width parameter asks for a linear layout of vertices such that the bipartite graph induced by edges crossing any vertex cut has a maximum induced matching of bounded size. Belmonte and Vatshelle [2] showed that interval graphs, bi-interval graphs, convex graphs and permutation graphs, where clique-width can be proportional to the square root of the number of vertices [11], all have LMIM-width 1 and an optimal layout can be found in polynomial time.

Since many well-known classes of graphs have bounded MIM-width or LMIM-width, algorithms that run in XP time in these parameters will yield polynomial-time algorithms on several interesting graph classes at once. Such algorithms have been developed for many problems: by Bui-Xuan et al [5] for the class of LCVS-VP - Locally Checkable Vertex Subset and Vertex Partitioning - problems, by Jaffke et al for non-local problems like Feedback Vertex Set [15, 14] and also for Generalized Distance Domination [13], by Golovach et al [10] for output-polynomial Enumeration of Minimal Dominating sets, by Bergougnoux and Kanté [3] for several Connectivity problems and by Galby et al for Semitotal Domination [9]. These results give a common explanation for many classical results in the field of algorithms on special graph classes and extends them to the field of parameterized complexity.

Note that very low MIM-width or LMIM-width still allows quite complex cuts compared to similarly defined graph parameters. For example, carving-width 1 allows just a single edge, maximum matching-width 1 a star graph, and rank-width 1 a complete bipartite graph. In contrast, LMIM-width 1 allows any cut

* This is the appendix of our WG submission, the long version with extra figures and full proofs
1 In [2], results are stated in terms of $d$-neighborhood equivalence, but in the proof, they actually gave a bound on LMIM-width.
where the neighborhoods of the vertices in a color class can be ordered linearly w.r.t. inclusion. In fact, it is an open problem whether the class of graphs having LMIM-width 1 can be recognized in polynomial-time or if this is NP-complete. Saether et al [19] showed that computing the exact MIM-width and LMIM-width of general graphs is W-hard and not in APX unless NP=ZPP, while Yamazaki [21] shows that under the small set expansion hypothesis it is not in APX unless P=NP. The only graph classes where we know an exact polynomial-time algorithm computing LMIM-width are the above-mentioned classes INTERVAL, BI-INTERVAL, CONVEX and PERMUTATION that all have structured neighborhoods implying LMIM-width 1 [2]. Belmonte and Vatshelle also gave polynomial-time algorithms showing that CIRCULAR ARC and CIRCULAR PERMUTATION graphs have LMIM-width at most 2, while DILWORTH k and k-TRAPEZOID have LMIM-width at most k [2]. Recently, Fomin et al [8] showed that LMIM-width for the very general class of H-GRAIDS is bounded by 2|E(H)|, and that a layout can be found in polynomial time if given an H-representation of the input graph. However, none of these results compute the exact LMIM-width. On the negative side, Mengel [16] has shown that STRONGLY CHORDAL SPLIT graphs, CO-COMPARABILITY graphs and CIRCLE graphs all can have MIM-width, and LMIM-width, linear in the number of vertices.

Just as LMIM-width can be seen as the linear variant of MIM-width, path-width can be seen as the linear variant of tree-width. Linear variants of other well-known parameters like clique-width and rank-width have also been studied. Arguably, the linear variant of MIM-width commands a more noteworthy position, since in contrast to these other linear parameters, for almost all well-known graph classes where the original parameter (MIM-width) is bounded then also the linear variant (LMIM-width) is bounded.

In this paper we give an $O(n \log n)$ algorithm computing the LMIM-width of an $n$-node tree. This is the first graph class of LMIM-width larger than 1 having a polynomial-time algorithm computing LMIM-width and thus constitutes an important step towards a better understanding of this parameter. The path-width of trees was first studied in the early 1990s by Möhring [17], with Ellis et al [7] giving an $O(n \log n)$ algorithm computing an optimal path-decomposition, and Bodlaender [4] an $O(n)$ algorithm. In 2013 Adler and Kanté [11] gave linear-time algorithms computing the linear rank-width of trees and also the linear clique-width of trees, by reduction to the path-width algorithm. Even though LMIM-width is very different from path-width, the basic framework of our algorithm is similar to the path-width algorithm in [7].

In Section 2 we give some standard definitions and prove the Path Layout Lemma, that if a tree $T$ has a path $P$ such that all components of $T \setminus N[P]$ have LMIM-width at most $k$ then $T$ itself has a linear layout with LMIM-width at most $k+1$. We use this to prove a classification theorem stating that a tree $T$ has LMIM-width at least $k+1$ if and only if there is a node $v$ such that after rooting $T$ in $v$, at least three children of $v$ themselves have at least one child whose rooted subtree has LMIM-width at least $k$. From this it follows that the LMIM-width of an $n$-node tree is no more than $\log n$. Our $O(n \log n)$ algorithm computing
LMIM-width of a tree $T$ picks an arbitrary root $r$ and proceeds bottom-up on the rooted tree $T_r$. In Section 3 we show how to assign labels to the rooted subtrees encountered in this process giving their LMIM-width. However, as with the algorithm computing pathwidth of a tree, the label is sometimes complex, consisting of LMIM-width of a sequence of subgraphs, of decreasing LMIM-width, that are not themselves full rooted subtrees. Proposition 1 is an 8-way case analysis giving a subroutine used to update the label at a node given the labels at all children. In Section 4 we give our bottom-up algorithm, which will make calls to the subroutine underlying Proposition 1 in order to compute the complex labels and the LMIM-width. Finally, we use all the computed labels to lay out the tree in an optimal manner.

2 Classifying LMIM-width of Trees

We use standard graph theoretic notation, see e.g. [6]. For a graph $G = (V, E)$ and subset of its nodes $S \subseteq V$ we denote by $N(S)$ the set of neighbors of nodes in $S$, by $N[S] = S \cup N(S)$ its closed neighborhood, and by $G[S]$ the graph induced by $S$. For a bipartite graph $G$ we denote by MIM($G$), or simply MIM if the graph is understood, the size of its Maximum Induced Matching, the largest number of edges whose endpoints induce a matching. Let $\sigma$ be the linear order corresponding to the enumeration $v_1, \ldots, v_n$ of the nodes of $G$, this will also be called a linear layout of $G$. For any index $1 \leq i < n$ we have a cut of $\sigma$ that defines the bipartite graph on edges “crossing the cut” i.e. edges with one endpoint in $\{v_1, \ldots, v_i\}$ and the other endpoint in $\{v_{i+1}, \ldots, v_n\}$. The maximum induced matching of $G$ under layout $\sigma$ is denoted $\text{mim} (\sigma, G)$, and is defined as the maximum, over all cuts of $\sigma$, of the value attained by the MIM of the cut, i.e. of the bipartite graph defined by the cut. The linear induced matching width – LMIM-width – of $G$ is denoted $\text{lmw} (G)$, and is the minimum value of $\text{mim} (\sigma, G)$ over all possible linear orderings $\sigma$ of the vertices of $G$.

We start by showing that if we have a path $P$ in a tree $T$ then the LMIM-width of $T$ is no larger than the largest LMIM-width of any component of $T \setminus N[P]$, plus 1. To define these components the following notion is useful.

**Definition 1 (Dangling tree).** Let $T$ be a tree containing the adjacent nodes $v$ and $u$. The dangling tree from $v$ in $u$, $T \langle v, u \rangle$, is the component of $T \setminus (u, v)$ containing $u$.

Given a node $x \in T$ with neighbours $\{v_1, \ldots, v_d\}$, the forest obtained by removing $N[x]$ from $T$ is a collection of dangling trees $\{T \langle v_i, u_{i,j} \rangle\}$, where $u_{i,j} \neq x$ is some neighbour of $v_i$. We can generalise this to a path $P = (x_1, \ldots, x_p)$ in place of $x$, such that $T \setminus N[P] = \{T \langle v_{i,j}, u_{i,j,m} \rangle\}$, where $v_{i,j} \in N(P)$ is a neighbour of $x_i$ and $u_{i,j,m} \notin N[P]$. See top part of Figure 1. This naming convention will be used in the following.

**Lemma 1 (Path Layout Lemma).** Let $T$ be a tree. If there exists a path $P = (x_1, \ldots, x_p)$ in $T$ such that every connected component of $T \setminus N[P]$ has
LMIM-width \( \leq k \) then \( \text{lmw}(T) \leq k + 1 \). Moreover, given the layouts for the components we can in linear time compute the layout for \( T \).

Proof. Using the optimal linear orderings of the connected components of \( T \setminus N[P] \), we give the below algorithm LINORD constructing a linear order \( \sigma_T \) on the nodes of \( T \) showing that \( \text{lmw}(T) \leq k + 1 \). The ordering \( \sigma_T \) starts out empty and the algorithm has an outer loop going through vertices in the path \( P = (x_1, \ldots, x_p) \).

When arriving at \( x_i \) it uses the concatenation operator \( \oplus \) to add the path node \( x_i \). Before looping over all neighbors \( v_{i,j} \) of \( x_i \) it adds the linear orders of each dangling tree from \( v_{i,j} \) and then \( v_{i,j} \) itself. See Figure 1 for an illustration.

function LINORD(T; tree, \( P = (x_1, \ldots, x_p) \); path, \( \{\sigma_{T(v_{i,j}, u_{i,j,m})} \} \); lin-ords)

\[ \sigma_T \leftarrow \emptyset \quad \text{\( \triangleright \) The list starts out empty} \]

for \( i \leftarrow 1, p \) do

\[ \sigma_T \leftarrow \sigma_T \oplus x_i \quad \text{\( \triangleright \) For all nodes on path \((x_1, \ldots, x_p)\)} \]

for \( j \leftarrow 1, |N(x_i)\setminus P| \) do

for \( m \leftarrow 1, |N(v_{i,j})\setminus x_i| \) do

\[ \sigma_T \leftarrow \sigma_T \oplus \sigma_{T(v_{i,j}, u_{i,j,m})} \quad \text{\( \triangleright \) Append given order of } \sigma_{T(v_{i,j}, u_{i,j,m})} \]

\[ \sigma_T \leftarrow \sigma_T \oplus v_{i,j} \quad \text{\( \triangleright \) Append } v_{i,j} \]

Fig. 1. A tree with a path \( P = (x_1, x_2, x_3, x_4) \), with nodes in \( N[N[P]] \) and dangling trees featured, and below it the order given by the Path Layout Lemma.

Firstly, from the algorithm it should be clear that each node of \( T \) is added exactly once to \( \sigma_T \), that it runs in linear time, and that there is no cut containing two crossing edges from two separate dangling trees. Now we must show that \( \sigma_T \) does not contain cuts with MIM larger than \( k + 1 \). By assumption the layout of each dangling tree has no cut with MIM larger than \( k \), and since these layouts can be found as subsequences of \( \sigma_T \) it follows that then also \( \sigma_T \) has no cut with more than \( k \) edges from a single dangling tree \( T(v_{i,j}, u_{i,j,m}) \). Also, we know that
edges from two separate dangling trees cannot both cross the same cut. The only edges of $T$ left to account for, i.e. not belonging to one of the dangling trees, are those with both endpoints in $N[N[P]]$, the nodes at distance at most 2 from a node in $P$. For every cut of $\sigma_T$ that contains more than a single crossing edge $(x_i, x_{i+1})$ there is a unique $x_i \in P$ and a unique $v_{i,j} \in N(x_i)$ such that every edge with both endpoints in $N[N[P]]$ that crosses the cut is incident on either $x_i$ or $v_{i,j}$, and since the edge connecting $x_i$ and $v_{i,j}$ also crosses the cut at most one of these edges can be taken into an induced matching. With these observations in mind, it is clear that $lmw(T) \leq mim(\sigma_T, T) \leq k + 1$.

**Definition 2 (k-neighbour and k-component index).** Let $x$ be a node in the tree $T$ and $v$ a neighbour of $x$. If $v$ has a neighbour $u \neq x$ such that $lmw(T(v,u)) \geq k$, then we call $v$ a $k$-neighbour of $x$. The $k$-component index of $x$ is equal to the number of $k$-neighbours of $x$ and is denoted $D_T(x,k)$, or shortened to $D(x,k)$.

**Theorem 1 (Classification of LMIM-width of Trees).** For a tree $T$ and $k \geq 1$ we have $lmw(T) \geq k + 1$ if and only if $D(x,k) \geq 3$ for some node $x$.

**Proof.** We first prove the backward direction by contradiction. Thus we assume $D(x,k) \geq 3$ for a node $x$ and there is a linear order $\sigma$ such that $mim(\sigma, T) \leq k$.

Let $v_1, v_2, v_3$ be the three $k$-neighbours of $x$ and $T_1, T_2, T_3$ the three trees of $T \setminus N[x]$ each of LMIM-width $k$, with $v_1$ connected to a node of $T_i$ for $i = 1, 2, 3$, that we know must exist by the definition of $D(x,k)$. We know that for each $i = 1, 2, 3$ we have a cut $C_i$ in $\sigma$ with MIM-$k$ and all $k$ edges of this induced matching coming from the tree $T_i$. Wlog we assume these three cuts come in the order $C_1, C_2, C_3$, i.e. with the cut having an induced matching of $k$ edges of $T_2$ in the middle. Note that in $\sigma$ all nodes of $T_1$ must appear before $C_2$ and all nodes of $T_3$ after $C_2$, as otherwise, since $T$ is connected and the distance between $T_2$ and the two trees $T_1$ and $T_3$ is at least two, there would be an extra edge crossing $C_2$ that would increase MIM of this cut to $k + 1$. It is also clear that $v_1$ has to be placed before $C_2$ and $v_3$ has to be placed after $C_2$, for the same reason, e.g. the edge between $v_1$ and a node of $T_1$ cannot cross $C_2$ without increasing MIM. But then we are left with the vertex $x$ that cannot be placed neither before $C_2$ nor after $C_2$ without increasing MIM of this cut by adding at least one of $(v_1,x)$ or $(v_3,x)$ to the induced matching. We conclude that $D(x,k) \geq 3$ for a node $x$ implies LMIM-width at least $k + 1$.

To prove the forward direction we first show the following partial claim: if $lmw(T) \geq k + 1$ then there exists a node $x \in T$ such that $D(x,k) \geq 3$; or there exists a strict subtree $S$ of $T$ with $lmw(S) \geq k + 1$. We will prove the contrapositive statement, so let us assume that every node in $T$ has $D(x,k) < 3$ and no strict subtree of $T$ has LMIM-width $\geq k + 1$ and show that then $lmw(T) \leq k$. For every node $x \in T$, it must then be true that $D(x,k) \leq 2$ and that $D(x,k+1) = 0$. The strategy of this proof is to show that there is always a path $P$ in $T$ such that all the connected components in $T \setminus N[P]$ have LMIM-width $\leq k - 1$. When we have shown this, we proceed to use the Path
Layout Lemma, to get that $lmw(T) \leq k$. To prove this, we define the following two sets of vertices:

$$X = \{ x | x \in V(T) \text{ and } D(x, k) = 2 \}, \quad Y = \{ y | y \in V(T) \text{ and } D(y, k) = 1 \}$$

Case 1: $X \neq \emptyset$

If $x_i$ and $x_j$ are in $X$, then every vertex on the path $P(x_i, \ldots, x_j)$ connecting $x_i$ and $x_j$ must be elements of $X$, as every node on this path clearly has a dangling tree with LMIM-width $k$ in the direction of $x_i$ and in the direction of $x_j$. The fact that every pair of vertices in $X$ are connected by a path in $X$ means that $X$ must be a connected subtree of $T$. Furthermore, this subtree must be a path, otherwise there are three disjoint dangling trees $T(v_1, u_1), T(v_2, u_2), T(v_3, u_3)$, each with LMIM-width $k$, and each hanging from a separate node. But then there is some vertex $w$ such that $T(v_1, u_1), T(v_2, u_2)$ and $T(v_3, u_3)$ are subtrees of dangling trees from different neighbours of $w$. But this implies that $D(w, k) \geq 3$, which we assumed were not the case, so this leads to a contradiction. We therefore conclude that all nodes in $X$ must lie on some path $P = (x_1, \ldots, x_p)$. The final part of the argument lies in showing that we can apply the Path Layout Lemma. For some $x_i \in P, i \in \{2, \ldots, p-1\}$, its $k$-neighbours are $x_{i-1}$ and $x_{i+1}$. For $x_1$, these neighbours are $x_2$ and some $x_0 \notin X$. For $x_p$, these neighbours are $x_{p-1}$ and some $x_{p+1} \notin X$. $x_0$ and $x_{p+1}$ may only have one $k$-neighbour – $x_1$ and $x_p$ respectively – or else they would be in $X$. If we make $P' = (x_0, \ldots, x_{p+1})$, we then see that every connected component in $T \setminus N[P']$ must have LMIM-width $\leq k - 1$. By the Path Layout Lemma, $lmw(T) \leq k$.

Case 2: $X = \emptyset, Y \neq \emptyset$

We construct the path $P$ in a simple greedy manner as follows. We start with $P = (y_1, y_2)$, where $y_1$ is some arbitrary node in $Y$, and $y_2$ its only $k$-neighbour. Then, if the highest-numbered node in $P$, call it $y_q$, has a $k$-neighbour $y' \notin P$, then we assign $y_{q+1}$ to $y'$, and repeat this process exhaustively. Since we look at finite graphs, we will eventually reach some node $y_p$ such that either $y_p \notin Y$ or $y_p$’s $k$-neighbour is $y_{p-1}$. We are then done and have $P = (y_1, \ldots, y_p)$, which must be a path in $T$, since every node $y_{i+1} \in P$ is a neighbour of $y_i$ and for $y_i$ we only assign maximally one such $y_{i+1}$. Also, every connected component of $T \setminus N[P]$ must have LMIM-width $\leq k - 1$. If not, some node $y_i \in P$ would have a $k$-neighbour $y' \notin P$, but by the assumption $X = \emptyset$ this is impossible, since then either $i < p$ and $y_i$ has two $k$-neighbours $y'$ and $y_{i+1}$, or else $i = p$ and $y_p \in Y$ and $y_i$ has the two $k$-neighbors $y'$ and $y_{i-1}$ (in case $i = p$ and $y_p \notin Y$ then by definition of $Y$ the node $y_i$ could not have a $k$-neighbor $y'$). By the Path Layout Lemma, $lmw(T) \leq k$.

Case 3: $X = \emptyset, Y = \emptyset$

If you make $P = (x)$ for some arbitrary $x \in T$, it is obvious that every connected component of $T \setminus N[P]$ has LMIM-width $\leq k - 1$. By the Path Layout Lemma, $lmw(T) \leq k$. 
We have proven the partial claim that if $\text{lmw}(T) \geq k + 1$ then there exists a node $x \in T$ such that $D(x, k) \geq 3$; or there exists a strict subtree $S$ of $T$ with $\text{lmw}(S) \geq k + 1$. To finish the backward direction of the theorem we need to show that if $\text{lmw}(T) \geq k + 1$ then there exists a node $x \in T$ with $D(x, k) \geq 3$; or there exists a strict subtree $S$ of $T$ with $\text{lmw}(S) \geq k + 1$. By the partial claim, there must then exist a strict subtree $S_0$ with $\text{lmw}(S_0) = k + 1$ with no strict subtree with LMIM-width $> k$. By the partial claim, $S_0$ must contain a node $x_0$ with $D_{S_0}(x_0, k) \geq 3$. But every dangling tree $S_0(v, u)$ is a subtree of $T(v, u)$, and so if $D_{S_0}(x_0, k) \geq 3$, then $D_T(x_0, k) \geq 3$ contradicting our assumption.

![Diagram of a tree with nodes and subtrees](image)

**Fig. 2.** The smallest tree with LMIM-width 2, having a node $v$ with three 1-neighbors $u_1$, $u_2$, $u_3$ having dangling trees $S_1$, $S_2$, $S_3$, respectively, so that $D(v, 1) = 3$

By Theorem 1, every tree with LMIM-width $k \geq 2$ must be at least 3 times bigger than the smallest tree with LMIM-width $k - 1$, which implies the following.

**Remark 1.** The LMIM-width of an $n$-node tree is $O(\log n)$.

### 3 Rooted trees, $k$-critical nodes and labels

Our algorithm computing LMIM-width will work on a rooted tree, processing it bottom-up. We will choose an arbitrary node $r$ of the tree $T$ and denote by $T_r$ the tree rooted in $r$. For any node $x$ we denote by $T_r[x]$ the standard complete subtree of $T_r$ rooted in $x$. During the bottom-up processing of $T_r$ we will compute a label for various subtrees. The notion of a $k$-critical node is crucial for the definition of labels.

**Definition 3 ($k$-critical node).** Let $T_r$ be a rooted tree with $\text{lmw}(T_r) = k$. We call a node $x$ in $T_r$ $k$-critical if it has exactly two children $v_1$ and $v_2$ that each has at least one child, $u_1$ and $u_2$ respectively, such that $\text{lmw}(T_r[u_1]) = \text{lmw}(T_r[u_2]) = k$. Thus $x$ is $k$-critical if and only if $\text{lmw}(T) = k$ and $D_{T_r}(x, k) = 2$. 
Remark 2. If $T_r$ has LMIM-width $k$ it has at most one $k$-critical node.

Proof. For a contradiction, let $x$ and $x'$ be two $k$-critical nodes in $T_r$. There are then four nodes, $v_1, v_r, v_r', u_r$, the two $k$-neighbours of $x$ and $x'$ respectively, such that there exist dangling trees $T(v_1, u_1), T(v_r, u_r), T(v_r', u_r'), T(v_r', u_r')$ that all have LMIM-width $k$. If $x$ and $x'$ have a descendant/ancestor relationship in $T_r$, then assume wlog that $x'$ is a descendant of $v_1$, and note that $T(v_r, u_r), T(v_r', u_r)$ and $T(v_r', u_r')$ are disjoint trees in different neighbours of $x'$, thus $D_T(x', k) = 3$ and by Theorem 1 $T_r$ should have LMIM-width $k + 1$. Otherwise, all the dangling trees are disjoint, thus $D_T(x, k) = D_T(x', k) = 3$ and we arrive at the same conclusion.

Definition 4 (label). Let rooted tree $T_r$ have $lmw(T_r) = k$. Then $label(T_r)$ consists of a list of decreasing numbers, $(a_1, \ldots, a_p)$, where $a_1 = k$, appended with a string called last type, which tells us where in the tree an $a_p$-critical node lies, if it exists at all. If $p = 1$ then the label is simple, otherwise it is complex. The $label(T_r)$ is defined recursively, with type 0 being a base case for singletons and for stars, and with type 4 being the only one defining a complex label.

- Type 0: $r$ is a leaf, i.e. $T_r$ is a singleton, then $label(T_r) = (0, t.0)$; or all children of $r$ are leaves, then $label(T_r) = (1, t.0)$
- Type 1: No $k$-critical node in $T_r$, then $label(T_r) = (k, t.1)$
- Type 2: $r$ is the $k$-critical node in $T_r$, then $label(T_r) = (k, t.2)$
- Type 3: A child of $r$ is $k$-critical in $T_r$, then $label(T_r) = (k, t.3)$
- Type 4: There is a $k$-critical node $u_k$ in $T_r$ that is neither $r$ nor a child of $r$.

Let $w$ be the parent of $u_k$. Then $label(T_r) = k \oplus label(T_r \setminus T_r[w])$

In type 4 we note that $lmw(T_r \setminus T_r[w]) < k$ since otherwise $u_k$ would have three $k$-neighbors (two children in the tree and also its parent) and by Theorem 4 we would then have $lmw(T_r) = k + 1$. Therefore, all numbers in $label(T_r \setminus T_r[w])$ are smaller than $k$ and a complex label is a list of decreasing numbers followed by last type $\in \{t.0, t.1, t.2, t.3\}$. We now give a Proposition that for any node $x$ in $T_r$ will be used to compute $label(T_r[x])$ based on the labels of the subtrees rooted at the children and grand-children of $x$. The subroutine underlying this Proposition, see the decision tree in Figure 3, will be used when reaching node $x$ in the bottom-up processing of $T_r$.

Proposition 1. Let $x$ be a node of $T_r$ with children $Child(x)$, and given $label(T_r[x])$ for all $v \in Child(x)$. We define (and compute) $k = \max_{v \in Child(x)} \{lmw(T_r[v])\}$ and $N_k = \{v \in Child(x) \mid lmw(T_r[v]) = k\}$ and denote by $N_k = \{v_1, \ldots, v_q\}$ and by $l_i = label(T_r[v_i])$. Define (compute) $t_k = D_{T_r}[x](k, t.0)$ by noting that $t_k = \{|v_i \in N_k \mid v_i \text{ has child } u_j \text{ with } lmw(T_r[u_j]) = k\}$. Given this information, we can find $label(T_r[x])$ as follows:

- **Case 0:** if $|Child(x)| = 0$ then $label(T_r[x]) = (0, t.0)$; else if $k = 0$ then $label(T_r[x]) = (1, t.0)$
- **Case 1:** Every label in $N_k$ is simple and has last type equal to $t.1$ or $t.0$, and $t_k \leq 1$. Then, $label(T_r[x]) = (k, t.1)$
- **Case 2**: Every label in \( N_k \) is simple and has last-type equal to t.1 or t.0, but \( t_k = 2 \). Then, \( \text{lmw}(T_r[x]) = (k, t.2) \)
- **Case 3**: Every label in \( N_k \) is simple and has last-type equal to t.1 or t.0, but \( t_k \geq 3 \). Then, \( \text{lmw}(T_r[x]) = (k + 1, t.1) \)
- **Case 4**: \( |N_k| \geq 2 \) and for some \( v_i \in N_k \), either \( l_i \) is a complex label, or \( l_i \) has last-type equal to either t.2 or t.3. Then, \( \text{lmw}(T_r[x]) = (k + 1, t.1) \)
- **Case 5**: \( |N_k| = 1 \), \( l_i \) is a simple label and \( l_i \) has last-type equal to t.2. Then, \( \text{lmw}(T_r[x]) = (k, t.3) \)
- **Case 6**: \( |N_k| = 1 \), \( l_i \) is either complex or has last-type equal to t.3, and \( k \notin \text{label}(T_r[x] \setminus T_r[w]) \), where \( w \) is the parent of the \( k \)-critical node in \( T_r[v_i] \). Then, \( \text{lmw}(T_r[x]) = k \oplus \text{label}(T_r[x] \setminus T_r[w]) \)
- **Case 7**: \( |N_k| = 1 \), \( l_i \) is either complex or has last-type equal to t.3, and \( k \in \text{label}(T_r[x] \setminus T_r[w]) \), where \( w \) is the parent of the \( k \)-critical node in \( T_r[v_i] \). Then, \( \text{lmw}(T_r[x]) = (k + 1, t.1) \)

\[
\text{lmw}(T_r[x]) = k + 1 \text{ and } T_r[x] \text{ is a type 1 tree} \\
\text{lmw}(T_r[x]) = k \text{ and } T_r[x] \text{ is a type } (k+1). \text{tree} \\
\text{lmw}(T_r[x]) = k \oplus \text{label}(T_r[x] \setminus T_r[w]) \\
\text{lmw}(T_r[x]) = (k + 1, t.1)
\]

**Fig. 3.** A decision tree corresponding to the case analysis of Proposition 1

**Proof.** We show that exactly one case applies to every rooted tree and in each case we assign the label according to Definition 4. First the base case: either \( x \) is a leaf or all its children are leaves and we are in Case 0 and the label is assigned according to Def. 4. Otherwise, observe the decision tree in Figure 3. It follows from Def. 4, \( k, N_k \) and \( t_k \) that cases 1 up to 7 of Prop. 1 corresponds to cases 1 up to 7 in the decision tree - we mention this correspondence in the below - and this proves that exactly one case applies to every rooted tree. The following facts simplify the case analysis: \( \text{lmw}(T_r[x]) \) is equal to either \( k \) or \( k + 1 \), and since no subtree rooted in a child of \( x \) has LMIM-width \( k + 1 \) there cannot be any \( (k + 1) \)-critical node in \( T_r[x] \), therefore if \( \text{lmw}(T_r[x]) = k + 1 \), \( T_r[x] \) is always a type 1
tree and by Theorem \[\text{it must contain a node } v \text{ such that } D_{T_r[x]}(v, k) \geq 3.\]
This node must either be a \(k\)-critical node in a rooted subtree of \(T_r[x]\), or \(x\) itself. We go through the cases 1 to 7 in order.

Note that in Cases 1, 2, and 3 the condition 'Every label in \(N_k\) is simple and has last_type equal to t.1 or t.0' means there are no \(k\)-critical nodes in any subtree of \(T_r[x]\), because every \(T_r[v]\) for \(v \in \text{Child}(x)\) is either of type 1 or has LMIM-width < \(k\):

**Case 1:** By definition of \(t_k\), \(D_{T_r[x]}(x, k) \leq 1\). Therefore, \(\text{lmw}(T_r[x]) = k\), and \(T_r[x]\) is a type 1 tree.

**Case 2:** By definition of \(t_k\), \(D_{T_r[x]}(x, k) = 2\), and no other nodes are \(k\)-critical, therefore \(\text{lmw}(T_r[x]) = k\). But now \(x\) is \(k\)-critical in \(T_r[x]\) so \(T_r[x]\) is a type 2 tree.

**Case 3:** By definition of \(t_k\), \(D_{T_r[x]}(x, k) = 3\) and \(\text{lmw}(T_r[x]) = k + 1\).

For the remaining Cases 4, 5, 6 and 7, some \(T_r[v]\) for \(v \in \text{Child}(x)\) has LMIM-width \(k\) and is of type 2, 3 or 4, so at least one \(k\)-critical node exists in some subtree of \(T_r[x]\):

**Case 4:** There is a \(k\)-critical node \(u_k\) in some \(T_r[v_i]\) (not of type 1), and some other \(v_j\) has \(\text{lmw}(T_r[v_j]) = k\) (because \(|N_k| \geq 2\)). Now observe \(w\) the parent of \(u_k\). The dangling tree \(T_r[x \setminus T_r[w]]\) is a supertree of \(T_r[v_j]\) and thus has LMIM-width \(\geq k\). Therefore \(w\) is a \(k\)-neighbour of \(u_k\) and by Theorem \[\text{lmw}(T_r[x]) = k + 1\].

**Case 5:** \(x\) has only one child \(v\) with \(\text{lmw}(T_r[v]) = k\), and \(v\) is itself \(k\)-critical (\(T_r[v]\) is type 2). \(x\) cannot be a \(k\)-neighbour of \(v\) in the unrooted \(T_r[x]\), because every dangling tree from \(x\) is some \(T_r[v_i], v_i \neq v\) of \(x\), which we know has LMIM-width < \(k\). Since no other node in \(T\) is \(k\)-critical, \(\text{lmw}(T_r[x]) = k\), and since \(v\), a child of \(x\), is \(k\)-critical in \(T_r[x]\), \(T_r[x]\) is a type 3 tree.

**Case 6:** \(x\) has only one child \(v\) with \(\text{lmw}(T_r[v]) = k\), and there is a \(k\)-critical node \(u_k\) with parent \(w\) – neither of which are equal to \(x\) – in \(T_r[v]\) (\(T_r[v]\) is a type 3 or type 4 tree). Moreover, no tree rooted in another child of \(w\), apart from \(u_k\), can have LMIM-width \(\geq k\), since this would imply \(D_{T_r[v]}(u_k, k) = 3\) and thus \(\text{lmw}(T_r[v]) > k\); nor can \(T_r[x \setminus T_r[w]]\) have LMIM-width = \(k\), since then we would have \(k\) in \(\text{label}(T_r[x \setminus T_r[w]])\) disagreeing with the condition of Case 6. Therefore \(D_{T_r[x]}(u, k) = 2\), and \(\text{lmw}(T_r[x]) = k\). \(T_r[x]\) is thus a type 4 tree and the label is assigned according to the definition.

**Case 7:** \(T_r[v], u_k\) and \(w\) are as described in Case 6. But here, \(\text{lmw}(T_r[x \setminus T_r[w]]) = k\) (since the condition says that \(k\) is in its label), and thus \(w\) is a \(k\)-neighbour of its child \(u_k\) and by Theorem \[\text{lmw}(T_r[x]) = k + 1\].

We conclude that \(\text{label}(T_r[x])\) has been assigned the correct value in all possible cases.

## 4 Computing LMIM-width of Trees and Finding a Layout

The subroutine underlying Prop. \[\text{it will be used in a bottom-up algorithm that starts out at the leaves and works its way up to the root, computing labels}\]
of subtrees $T_r[x]$. However, in two cases (Case 6 and 7) we need the label of $T_r[x] \setminus T_r[w]$, which is not a complete subtree rooted in any node of $T_r$. Note that the label of $T_r[x] \setminus T_r[w]$ is again given by a (recursive) call to Prop. 1 and is then stored as a suffix of the complex label of $T_r[x]$. We will compute these labels by iteratively calling Prop. 1 (substituting the recursion by iteration). We first need to carefully define the subtrees involved when dealing with complex labels.

From the definition of labels it is clear that only type 4 trees lead to a complex label. In that case we have a tree $T_r[x]$ of LMIM-width $k$ and a $k$-critical node $u_k$ that is neither $x$ nor a child of $x$, and the recursive definition gives $\text{label}(T_r[x]) = k \oplus \text{label}(T_r[x] \setminus T_r[w])$ for $w$ the parent of $u_k$. Unravelling this recursive definition, this means that if $\text{label}(T_r[x]) = (a_1, \ldots, a_p, \text{last type})$, we can define a list of nodes $(w_1, \ldots, w_{p-1})$ where $w_i$ is the parent of an $a_i$-critical node in $T_r[x] \setminus (T_r[w_1] \cup \ldots \cup T_r[w_{i-1}])$. We expand this list with $w_p = x$, such
that there is one node in $T_r[x]$ corresponding to each number in $\text{label}(T_r[x])$, and $T_r[x]\setminus(T_r[w_1] \cup \ldots \cup T_r[w_p]) = \emptyset$.

Now, in the first level of a recursive call to Prop. 4 the role of $T_r[x]$ is taken by $T_r[x]\setminus T_r[w_1]$, and in the next level it is taken by $(T_r[x]\setminus T_r[w_1])\setminus T_r[w_2]$ etc. The following definition gives a shorthand for denoting these trees.

**Definition 5.** Let $x$ be a node in $T_r$, $\text{label}(T_r[x]) = (a_1, a_2, \ldots, a_p, \text{last_type})$ and the corresponding list of vertices $(w_1, \ldots, w_p)$ is as we describe in the above text. For any non-negative integer $s$, the tree $T_r[x,s]$ is the subtree of $T_r[x]$ obtained by removing all trees $T_r[w_i]$ from $T_r[x]$, where $a_i \geq s$. In other words, if $q$ is such that $a_q \geq s > a_{q+1}$, then $T_r[x,s] = T_r[x]\setminus(T_r[w_1] \cup T_r[w_2] \cup \ldots \cup T_r[w_q])$.

**Remark 3.** Some important properties of $T_r[x,s]$ are the following. Let $T_r[x,s]$, $\text{label}(T_r[x,s])$, $(w_1, \ldots, w_p)$ and $q$ as in the definition. Then

1. If $s > a_1$, then $T_r[x,s] = T_r[x]$
2. $\text{label}(T_r[x,s]) = (a_{q+1}, \ldots, a_p, \text{last_type})$
3. $\text{lmw}(T_r[x,s]) = a_{q+1} < s$
4. $\text{lmw}(T_r[x,s+1]) = s$ if and only if $s \in \text{label}(T_r[x])$
5. $T_r[x,s+1] \neq T_r[x,s]$ if and only if $s \in \text{label}(T_r[x])$

**Proof.** These follow from the definitions, maybe the last one requires a proof: 

**Backward direction:** Let $s = a_q$ for some $1 \leq q \leq p$. Then $T_r[x,s+1] = T_r[x]\setminus(T_r[w_1] \cup \ldots \cup T_r[w_{q-1}])$ and $T_r[x,s] = T_r[x]\setminus(T_r[w_1] \cup \ldots \cup T_r[w_q])$. These two trees are clearly different.

**Forward direction:** Let $T_r[x,s] = T_r[x]\setminus(T_r[w_1] \cup \ldots \cup T_r[w_q])$ and $T_r[x,s+1] = T_r[x]\setminus(T_r[w_1] \cup \ldots \cup T_r[w_{q'}])$ with $q' < q$ and $a_{q'} > a_q$ (because numbers in a label are strictly descending). $a_q < s + 1$ and $a_q \geq s$, ergo $a_q = s$.

Note that for any $s$ the tree $T_r[x,s]$ is defined only after we know $\text{label}(T_r[x])$. In the algorithm, we compute $\text{label}(T_r[x])$ by iterating over increasing values of $s$ (until $s > \text{lmw}(T_r[x])$) since by Remark 3 we then have $T_r[x,s] = T_r[x]$ and we could hope for a loop invariant saying that we have correctly computed $\text{label}(T_r[x,s])$. However, $T_r[x,s]$ is only known once we are done. Instead, each iteration of the loop will correctly compute the label of the following subtree called $T_{\text{union}}[x,s]$, which is not always equal to $T_r[x]$, but importantly for $s > \text{lmw}(T_r[x])$, we will have $T_{\text{union}}[x,s] = T_r[x,s] = T_r[x]$.

**Definition 6.** Let $x$ be a node in $T_r$ with children $v_1, \ldots, v_d$. $T_{\text{union}}[x,s]$ is then equal to the tree induced by $x$ and the union of all $T_r[v_i, s]$ for $1 \leq i \leq d$. More technically, $T_{\text{union}}[x,s] = T_r[V']$ where $V' = x \cup V(T_r[v_1, s]) \cup \ldots \cup V(T_r[v_d, s])$.

Given a tree $T$, we find its LMIM-width by rooting it in an arbitrary node $r$, and computing labels by processing $T_r$ bottom-up. The answer is given by the first element of $\text{label}(T_r[r])$, which by definition is equal to $\text{lmw}(T)$. At a leaf $x$ of $T_r$, we initialize by $\text{label}(T_r[x]) \leftarrow (0,t,0)$, and at a node $x$ for which all children are leaves we initialize by $\text{label}(T_r[x]) \leftarrow (1,t,0)$, according to Definition 4. When reaching a higher node $x$ we compute label of $T_r[x]$ by calling function $\text{MakeLabel}(T_r,x)$. 


function MAKELABEL($T_r$, $x$) 

▷ finds $\text{cur\_label} = \text{label}(T_r[x])$

$\text{cur\_label} \leftarrow (0, t.0)$  
▷ This is $\text{label}(T_{\text{union}}[x, 0])$

\{$v_1, \ldots, v_d$\} = children of $x$

\text{if } 0 \in \text{label}(T_r[v_i]) \text{ for some } i \text{ then}

$\text{cur\_label} \leftarrow (1, t.0)$  
▷ This is then $\text{label}(T_{\text{union}}[x, 1])$

\text{for } s \leftarrow 1, \text{max}_{i=1}^{d}\{\text{first element of label}(T_r[v_i])\} \text{ do}

\{$l'_1, \ldots, l'_d$\} = \{\text{label}(T_r[v_1, s + 1]) \mid 1 \leq i \leq d\}

$N_s = \{v_i \mid 1 \leq i \leq d, s \in l'_i\}$

t_s = \{|\{v_i \mid v_i \in N_s, v_i \text{ has child } u_j \text{ s.t. } s \in \text{label}(T_r[u_j, s + 1])\}|\}

\text{if } |N_s| > 0 \text{ then}

\text{case } \leftarrow \text{the case from Prop. 1 applying to } s, \{l'_1, \ldots, l'_d\}, N_s \text{ and } t_s

\text{cur\_label} \leftarrow \text{as given by case in Prop. 1}(s \oplus \text{cur\_label} \text{ if Case 6})

Fig. 5. The same decision tree as shown in Prop. 1 but adapted to MAKELABEL

Lemma 2. Given labels at descendants of node $x$ in $T_r$, MAKELABEL($T_r$, $x$) computes $\text{label}(T_r[x])$ as the value of $\text{cur\_label}$.

Proof. Assume that $x$ has the children $v_1, \ldots, v_d$, and denote their set of labels as $L = \{l_1, \ldots, l_d\}$. MAKELABEL keeps a variable $\text{cur\_label}$ that is updated maximally $k$ times in a for loop, where $k$ is the biggest number in any label of children of $x$. The following claim will suffice to prove the lemma, since for $s > \ln w(T_r[x])$, we have $T_{\text{union}}[x, s] = T_r[x]$.

Claim: At the end of the $s$'th iteration of the for loop the value of $\text{cur\_label}$ is equal to $\text{label}(T_{\text{union}}[x, s + 1])$. 
Base case: We have to show that before the first iteration of the loop we have \( \text{cur.label} = \text{label}(T_{\text{unio}n[x, 1]}) \). If some label \( l_i \in L \) has 0 as an element then \( T_{\text{unio}n[x, 1]} \) is isomorphic to a star with \( x \) as the center and \( v_i \) as a leaf. By Prop. 1 in this case \( \text{label}(T_{\text{unio}n[x, 1]}) = (1, t, 0) \) and this is what \( \text{cur.label} \) is initialized to. If no \( l_i \in L \) has 0 as an element, then by Remark 3.5, \( T_{\text{unio}n[x, 1]} = T_{\text{unio}n[x, 0]} \) which by definition is the singleton node \( x \) and by Prop. 1, the label of this tree is \((0, t, 0)\) and this is what \( \text{cur.label} \) is initialized to.

Induction step: We assume \( \text{cur.label} = \text{label}(T_{\text{unio}n[x, s]}) \) at the start of the \( s \)th iteration of the for loop and show that at the end of the iteration, \( \text{cur.label} = \text{label}(T_{\text{unio}n[x, s + 1]}) \).

The first thing done in the for loop is the computation of \( \{l'_i \mid 1 \leq i \leq d, \ l'_i = \text{label}(T_r[v_i, s + 1])\} \). By Remark 3.2, \( \text{label}(T_r[v_i, s + 1]) \subseteq \text{label}(T_r[v_i]) \) for all \( i \), therefore \( l'_1, \ldots, l'_d \) are trivial to compute. The second thing done is to set \( N_s \) as the set of all children of \( x \) whose labels contain \( s \), and \( t_s \) as the number of nodes in \( N_s \) that themselves have children whose labels contain \( s \). Let us first look at what happens when \( |N_s| = 0 \):

By Remark 3.5, for every child \( v_i \) of \( x, T_r[v_i, s + 1] = T_r[v_i, s] \) if \( s \notin \text{label}(T_r[v_i]) \). Therefore, if \( |N_s| = 0 \), then \( T_{\text{unio}n[x, s + 1]} = T_{\text{unio}n[x, s]} \), and from the induction assumption, \( \text{label}(T_{\text{unio}n[x, s + 1]}) = \text{cur.label} \), and indeed when \( |N_s| = 0 \) then iteration \( s \) of the loop does not alter \( \text{cur.label} \).

Otherwise, we have \( |N_s| > 0 \) and make a call to the subroutine given by Prop. 1 see the decision tree in Figure 5, to compute \( \text{label}(T_{\text{unio}n[x, s + 1]}) \) and argue first that the variables used in that call correspond to the variables used in Prop. 1 to compute \( \text{label}(T_r[x]) \). The correspondence is given in Table 4. Most of these are just observations: \( T_{\text{unio}n[x, s + 1]} \) corresponds to \( T_r[x] \)

| Proposition 1 | for loop iteration \( s \) | Explanation |
|---------------|--------------------------|-------------|
| \( T_r[x, k] \) | \( T_{\text{unio}n[x, s + 1], s} \) | Tree needing label, max \( \text{lmw} \) of children |
| \( T_r[v_1], \ldots, T_r[v_d] \) | \( T_{r[v_i, s], \ldots, T_r[v_d, s]} \) | Subtrees of children |
| \( l_1, \ldots, l_d, N_s, t_s \) | \( l'_1, \ldots, l'_d, N_s, t_s \) | Child labels, those with max, root comp. index |
| \( \text{label}(T_r[x] \setminus T_r[w]) \) | \( \text{cur.label} \) | This is also \( \text{label}(T_{\text{unio}n[x, s + 1] \setminus T_r[w, s + 1]} \) |

in Prop. 1 and \( T_r[v_1, s + 1], \ldots, T_r[v_d, s + 1] \) corresponds to \( T_r[v_1], \ldots, T_r[v_d] \).

\( \{l'_i \mid 1 \leq i \leq d, \ l'_i = \text{label}(T_r[v_i, s + 1])\} \) correspond to \( \{\text{label}(T_r[v]) \mid v \in \text{Child}\} \) in Prop. 1. \( N_s \) is defined in the algorithm so that it corresponds to \( N_k \) in Prop. 1. Since \( |N_s| > 0 \), some \( v_i \) has \( s \) in its label \( l'_i \). By Remark 3.3 and 3.4 we can infer that \( s \) is the maximum LMIM-width of all \( T_r[v_i, s + 1] \), therefore \( s \) corresponds to \( k \) in Proposition 1.

It takes a bit more effort to show that \( t_s \) computed in iteration \( s \) of the for loop corresponds to \( t_k = D_{T_r}[x, k] \) in Prop. 1, meaning we need to show that \( t_s = D_{T_{\text{unio}n}[x, s + 1]}[x, s] \). Consider \( v_i \), a child of \( x \). In accordance with MAKELABEL we say that \( v_i \) contributes to \( t_s \) if \( v_i \in N_s \) and \( v_i \) has a child \( u_j \) with \( s \) in its label. We thus need to show that \( v_i \) contributes to \( t_s \) if and only if \( v_i \) is an \( s \)-neighbour of \( x \) in \( T_{\text{unio}n}[x, s + 1] \). Observe that by Remark 3.4
Lastly, we show that if $T[v_i, s + 1]$ is an $S$-critical node – then the algorithm has $\text{lwm}(T_r[v_i, s + 1]) = s$ if and only if $s$ is in the labels of both $T_r[v_i]$ and $T_r[u_j]$. If $s \notin \text{label}(T_r[u_j, s + 1])$, then $\text{lwm}(T_r[u_j, s + 1]) < s$, and if this is true for all children of $v_i$, then $v_i$ is not an $s$-neighbour of $x$ in $T_{\text{union}}[x, s + 1]$. If $s \notin \text{label}(T_r[v_i, s + 1])$, then $\text{lwm}(T_r[v_i, s + 1]) < s$ and no subtree of $T_r[v_i, s + 1]$ can have LMIM-width $s$. However, if $s \in \text{label}(T_r[u_j, s + 1])$ and $s \in \text{label}(T_r[v_i, s + 1])$ (this is when $v_i$ contributes to $t_s$), then $T_r[v_i, s + 1] \cap T_r[u_j]$ must be equal to $T_r[u_j, s + 1]$ and $T_r[u_j, s + 1] \subseteq T_{\text{union}}[x, s + 1]$, and we conclude that $v_i$ is an $s$-neighbour of $x$ in $T_{\text{union}}[x, s + 1]$ if and only if $v_i$ contributes to $t_s$, so $t_s = D_{T_{\text{union}}[x, s + 1]}(x, s)$.

Lastly, we show that if $T_{\text{union}}[x, s + 1]$ is a Case 6 or Case 7 tree – that is, $|N_x| = 1$, and $T_r[v_1, s + 1]$ is a type 3 or type 4 tree, with $w$ being the parent of an $s$-critical node – then the algorithm has label$(T_{\text{union}}[x, s + 1] \setminus T_r[w, s + 1])$ available for computation, indeed that this is the value of cur_label. We know, by definition of label and Remark 3.5 that $T_r[v_i, s + 1] = T_r[w, s + 1]$. But since $|N_x| = 1$, for every $j \neq i$, $T_r[v_j, s + 1] \setminus T_r[v_j, s] = \emptyset$. Therefore $T_{\text{union}}[x, s + 1] \setminus T_{\text{union}}[x, s] = T_r[w, s + 1]$ and $T_{\text{union}}[x, s + 1] \setminus T_r[w, s + 1] = T_{\text{union}}[x, s]$. But by the induction assumption, cur_label = label$(T_{\text{union}}[x, s])$. Thus cur_label corresponds to label$(T_r[x] \setminus T_r[w])$ in Prop. 1.

We have now argued for all the correspondences in Table 4. By that, we conclude from Prop. 2 and Definition 6 and the inductive assumption that cur_label = label$(T_{\text{union}}[x, s + 1] \setminus T_r[w, s + 1])$ at the end of the $s$’th iteration of the for loop in MAKE-LABEL. It runs for $k$ iterations, where $k$ is equal to the biggest number in any label of the children of $x$, and cur_label is then equal to label$(T_{\text{union}}[x, k + 1])$.

Since $k \geq \text{lwm}(T_r[v_i])$ for all $i$, by definition $T_r[v_i, k + 1] = T_r[v_i]$ for all $i$, and thus $T_{\text{union}}[x, k + 1] = T_r[x]$. Therefore, when MAKE-LABEL finishes, cur_label = label$(T_r[x])$.

**Theorem 2.** Given any tree $T$, $\text{lwm}(T)$ can be computed in $O(n \log(n))$-time.

**Proof.** We find $\text{lwm}(T)$ by bottom-up processing of $T_r$ and returning the first element of label$(T_r)$. After correctly initializing at leaves and nodes whose children are all leaves, we make a call to MAKE-LABEL for each of the remaining nodes. Correctness follows by Lemma 2 and induction on the structure of the rooted tree. For the timing we show that each call runs in $O(\log n)$ time. For every integer $s$ from 1 to $m$, the biggest number in any label of children of $x$, which is $O(\log n)$ by Remark 3.5, the algorithm checks how many labels of children of $x$ contain $s$ (to compute $N_s$), and how many labels of grandchildren of $x$ contain $s$ (to compute $t_s$). The labels are sorted in descending order, therefore the whole loop goes only once through each of these labels, each of length $O(\log n)$. Other than this, MAKE-LABEL only does a constant amount of work. Therefore, MAKE-LABEL($T_r, x$), if $x$ has $a$ children and $b$ grandchildren, takes time proportional to $O(\log n)(a + b)$. As the sum of the number of children and grandchildren over all nodes of $T_r$ is $O(n)$ we conclude that the total runtime to compute $\text{lwm}(T)$ is $O(n \cdot \log n)$.

**Theorem 3.** A layout of LMIM-width $\text{lwm}(T)$ of a tree $T$ can be found in $O(n \cdot \log n)$-time.
Proof. Given $T$ we first run the algorithm computing $lmw(T)$ by finding labels of all nodes and various subtrees. Given $T$ we first run the algorithm computing $lmw(T)$ finding the label of every full rooted subtree in $T_r$. We give a recursive layout-algorithm that uses these labels in tandem with LinORD presented in the Path Layout Lemma. We call it on a rooted tree where labels of all subtrees are known. For simplicity we call this rooted tree $T_r$ even though in recursive calls this is not the original root $r$ and tree $T$. The layout-algorithm goes as follows:

1) Let $lmw(T_r) = k$ and find a path $P$ in $T_r$ such that all trees in $T_r \setminus N[P]$ have LMIM-width $< k$. The path depends on the type of $T_r$ as explained in detail below.

2) Call this layout-algorithm recursively on every rooted tree in $T_r \setminus N[P]$ to obtain linear layouts; to this end, we need the correct label for every node in these trees.

3) Call LinORD on $T_r$, $P$ and the layouts provided in step 2.

Every tree in the forest $T \setminus N[P]$ is equal to a dangling tree $T(v, u)$, where $v$ is a neighbour of some $x \in P$.

We observe that if $lmw(T) = k$, then by definition $lmw(T(v, u)) = k$ if and only if $v$ is a $k$-neighbour of $x$. It follows that every tree in $T \setminus N[P]$ has LMIM-width at most $k - 1$ if and only if no node in $P$ has a $k$-neighbour that is not in $P$. We use this fact to show that for every type of tree we can find a satisfying path in the following way:

Type 0 trees: Choose $P = (r)$. Since $T \setminus N[r] = \emptyset$ in these trees, this must be a satisfying path.

Type 1 trees: These trees contain no $k$-critical nodes, which by definition means that for any node $x$ in $T_r$, at most one of its children is a $k$-neighbour of $x$. Choose $P$ to start at the root $r$, and as long as the last node in $P$ has a $k$-neighbour $v$, $v$ is appended to $P$. This set of nodes is obviously a path in $T_r$. No node in $P$ can possibly have a $k$-neighbour outside of $P$, therefore all connected components of $T \setminus N[P]$ have LMIM-width $\leq k - 1$. Furthermore, all components of $T \setminus N[P]$ are full rooted sub-trees of $T_r$ and so the labels are already known.

Type 2 trees: In these trees the root $r$ is $k$-critical. We look at the trees rooted in the two $k$-neighbours of $r$, $T_r[v_1]$ and $T_r[v_2]$. By Remark 2 these must be Type 1 trees, and so we find paths $P_1, P_2$ in $T_r[v_1]$ and $T_r[v_2]$ respectively, as described above. Gluing these paths together at $r$ we get a satisfying path for $T_r$, and we still have correct labels for the components $T \setminus N[P]$.

Type 3 trees: In these trees, $r$ has exactly one child $v$ such that $T_r[v]$ is of type 2 and none of its other children have LMIM-width $k$. We choose $P$ as we did above for $T_r[v]$. $r$ is clearly not a $k$-neighbour of $v$, or else $D_T(v, k) = 3$. Every other node in $P$ has all their neighbours in $T_r[v]$. Again, every tree in $T \setminus N[P]$ is a full rooted subtree, and every label is known.

Type 4 trees: In these trees, $T_r$ contains precisely one node $w \neq r$ such that $w$ is the parent of a $k$-critical node, $x$. This $w$ is easy to find using the labels, and clearly the tree $T_r[w]$ is a type 3 tree with LMIM-width $k$. We find a path $P$
that is satisfying in $T_r[w]$ as described above. $w$ is still not a $k$-neighbour of $x$, therefore $P$ is a satisfying path. In this case, we have one connected component of $T \setminus N[P]$ that is not a full rooted subtree of $T_r$, that is $T_r \setminus T_r[w]$. Thus for every ancestor $y$ of $w$ (the blue path in Figure 6) $T_r[y] \setminus T_r[w]$ is not a full rooted subtree either, and we need to update the labels of these trees. However, $T_r[y] \setminus T_r[w]$ is by definition equal to $T_r[y, k]$, whose label is equal to $\text{label}(T_r[y])$ without its first number. Thus we quickly find the correct labels to do the recursive call.

![Fig. 6. The path $P$ in green for the proof of Theorem 3.](image)

5 Conclusion

We have given an $O(n \log n)$ algorithm computing the LMIM-width and an optimal layout of an $n$-node tree. This is the first graph class of LMIM-width larger than 1 having a polynomial-time algorithm computing LMIM-width and thus constitutes an important step towards a better understanding of LMIM-width. Indeed, for the development of FPT algorithms computing tree-width and pathwidth of general graphs, one could argue that the algorithm of [7] computing optimal path-decompositions of a tree in time $O(n \log n)$ was a stepping stone. The situation is different for MIM-width and LMIM-width, as it is W-hard to compute these parameters [19], but it is similar in the sense that our objective has been to achieve an understanding of how to take a graph and assemble a decomposition of it, in this case a linear one, such that it has cuts of low MIM. To achieve this objective a polynomial-time algorithm for trees has been our main goal.

Bibliography

[1] Isolde Adler and Mamadou Moustapha Kanté. Linear rank-width and linear clique-width of trees. *Theor. Comput. Sci.*, 589:87–98, 2015.
[2] Rémy Belmonte and Martin Vatshelle. Graph classes with structured neighborhoods and algorithmic applications. *Theor. Comput. Sci.*, 511:54 – 65, 2013.
[3] Benjamin Bergougnoux and Mamadou Moustapha Kanté. Rank based approach on graphs with structured neighborhood. *CoRR*, abs/1805.11275, 2018.

[4] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

[5] Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. Fast dynamic programming for locally checkable vertex subset and vertex partitioning problems. *Theor. Comput. Sci.*, 511:66 – 76, 2013.

[6] Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of Graduate texts in mathematics. Springer, 2012.

[7] John A. Ellis, Ivan Hal Sudborough, and Jonathan S. Turner. The vertex separation and search number of a graph. *Inf. Comput.*, 113(1):50–79, 1994.

[8] Fedor V. Fomin, Petr A. Golovach, and Jean-Florent Raymond. On the tractability of optimization problems on H-graphs. In *Proc. ESA 2018*, pages 30:1 – 30:14, 2018.

[9] Esther Galby, Andrea Munaro, and Bernard Ries. Semitotal domination: New hardness results and a polynomial-time algorithm for graphs of bounded mim-width. *CoRR*, abs/1810.06872, 2018.

[10] Petr A. Golovach, Pinar Heggernes, Mamadou Moustapha Kanté, Dieter Kratsch, Sigve Hortemo Sæther, and Yngve Villanger. Output-polynomial enumeration on graphs of bounded (local) linear mim-width. *Algorithmica*, 80(2):714–741, 2018.

[11] Martin Charles Golumbic and Udi Rotics. On the clique-width of some perfect graph classes. *Int. J. Found. Comput. Sci.*, 11(3):423–443, 2000.

[12] Petr Hlinený, Sang-il Oum, Detlef Seese, and Georg Gottlob. Width parameters beyond tree-width and their applications. *Comput. J.*, 51(3):326–362, 2008.

[13] Lars Jaffke, O-joung Kwon, Torstein J. F. Strømme, and Jan Arne Telle. Generalized distance domination problems and their complexity on graphs of bounded mim-width. In *13th International Symposium on Parameterized and Exact Computation, IPEC 2018, August 20-24, 2018, Helsinki, Finland*, pages 6:1–6:14, 2018.

[14] Lars Jaffke, O-joung Kwon, and Jan Arne Telle. Polynomial-time algorithms for the longest induced path and induced disjoint paths problems on graphs of bounded mim-width. In *12th International Symposium on Parameterized and Exact Computation, IPEC 2017, September 6-8, 2017, Vienna, Austria*, pages 21:1–21:13, 2017.

[15] Lars Jaffke, O-joung Kwon, and Jan Arne Telle. A unified polynomial-time algorithm for feedback vertex set on graphs of bounded mim-width. In *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, pages 42:1–42:14, 2018.

[16] Stefan Mengel. Lower bounds on the mim-width of some graph classes. *Discrete Applied Mathematics*, 248:28–32, 2018.

[17] Rolf H Möhring. Graph problems related to gate matrix layout and pla folding. In *Computational graph theory*, pages 17–51. Springer, 1990.
[18] Sang-il Oum. Rank-width: Algorithmic and structural results. *Discrete Applied Mathematics*, 231:15–24, 2017.

[19] Sigve Hortemo Sæther and Martin Vatshelle. Hardness of computing width parameters based on branch decompositions over the vertex set. *Theor. Comput. Sci.*, 615:120–125, 2016.

[20] Martin Vatshelle. *New width parameters of graphs*. PhD thesis, University of Bergen, Norway, 2012.

[21] Koichi Yamazaki. Inapproximability of rank, clique, boolean, and maximum induced matching-widths under small set expansion hypothesis. *Algorithms*, 11(11):173, 2018.