COMPUTING OPTIMAL LINEAR LAYOUTS OF TREES IN LINEAR TIME *

KONSTANTINOS SKODINIS †

Abstract. We present a linear time algorithm for computing linear layouts of trees which are optimal with respect to vertex separation. The best algorithm known so far is given by [7] and needs $O(n \log n)$ time. Our result solves several other related open problems on trees as for example the one of [17].

Key words. linear graph algorithms, optimal linear layouts, optimal search strategies, optimal interval augmentations

AMS subject classifications. 68R10, 05C05, 05C85

1. Introduction. The *vertex separation* of graphs has been introduced in [15] in the form of a *vertex separator game*. It is an important concept both in graph theory and computational complexity. Since vertex separation is closely related to the *black-white pebbling* of [6] and to the *graph separators*, it has wide applications in register allocation and in VLSI design.
The *vertex separation* of a graph can be defined using *linear layouts*, see [7]. A linear layout, or simply a layout, of an n-vertex graph G is a bijection L between the vertices of G and the places $1, \ldots, n$. The i-th vertex *cut* is the number of vertices with a place number less than or equal to i having a neighbour in G with a place number larger than i. The vertex separation of G with respect to L is the maximum over all vertex cuts. The vertex separation of G, $vs(G)$, is the minimum vertex separation over all layouts of G. L is *optimal* if the vertex separation of G with respect to L is $vs(G)$.
If the cuts concern edges instead of vertices we obtain the well known *edge separation (cut-width)*. The vertex and edge separations of a graph may be far different if there is no degree restriction. An example for this is the star graph G_n with n rays. The vertex separation of G_n is one whereas the edge separation is $\lceil \frac{n}{2} \rceil$.

Recently vertex separation has received more attention due to its closer relation to several other well known concepts in Computer Science as for example *edge searching*, *node searching*, and *interval thickness*.

The *edge searching* of a graph has been introduced in [19]. Here the edges of a graph G are considered as pipes contaminated by a gas. A team of searchers (clearers) has to clear all edges of G. To do so the team has to find an *edge search strategy* which is a sequence of the following activities.

(i) Place a searcher on a vertex as a guard
(ii) Remove a searcher from a vertex
(iii) Slide a searcher along an edge

After each activity the edges can be partitioned into two classes: *cleared* and *contaminated*. A contaminated edge becomes cleared if one of its endpoints is guarded and a searcher slides along the edge from the guarded endpoint to the other endpoint of the edge. Notice that the searcher which is sliding along the edge may be the same one which guards the starting point of the edge. A cleared edge becomes *recontaminated*

* The work of the author was supported by the German Research Association (DFG) grant Br-835/7-1
†University of Passau, 94030 Passau, Germany, e-mail: skodinis@fmi.uni-passau.de

if by some activity of type (*ii*) or (*iii*) there is an unguarded path leading from the cleared edge to a contaminated one. Of course a recontaminated edge must be cleared again.
The maximum number of the searchers needed by a strategy S to clear G, $es_S(G)$, is the edge search number of G with respect to S. The *edge search number* of G, $es(G)$, is the minimum over all possible edge search strategies. An edge search strategy S is *optimal* if $es_S = es(G)$.
In [14] it has been shown that recontamination can be disallowed without increasing the search number of any graph. Thus we can assume that every optimal edge search strategy of a graph is *progressive*, i.e. every edge is searched only once.

Another version of graph searching has been introduced in [13]. It is called *node searching* in order to be distinguished from *edge searching*. In this slightly different version the searchers do not slide along the edges, i.e. the third activity of the edge searching disappears. An edge is cleared once both endpoints are simultaneously guarded by searchers. The remaining notions are defined in the same way as in edge searching. The *node search number* of G is denoted by $ns(G)$.

The *interval thickness* of a graph has been introduced in [13]. Given a set of intervals I of the real line the *interval graph* of I is obtained by representing every interval by a vertex and every intersection of two intervals by an edge between their corresponding vertices. For applications of interval graphs see [10].
The interval thickness of a graph G, $\theta(G)$, is the smallest maximum clique over all interval supergraphs of G. An interval set I is an *optimal interval augmentation* of G if the interval graph H of I is a supergraph of G and the maximum clique of H is equal to $\theta(G)$. Interval augmentation has many applications in certain VLSI layouts like gate matrix layout, see [8, 9].

Rather surprisingly all notions mentioned above are closely related to each other in the following sense.
In [7] it has been shown that $vs(G) \leq es(G) \leq vs(G) + 2$ and that the edge search number of a graph G is identical to the vertex separation of the 2-expansion G' of G, i.e. $es(G) = vs(G')$. G' is obtained from G by replacing every edge $\{u, v\}$ by two new vertices x, y and edges $\{u, x\}, \{x, y\}$, and $\{y, v\}$. Moreover given an optimal layout L' of G' an algorithm is presented which transforms L' into an optimal edge search strategy of G in linear time in the size of G' and therefore in the size of G.
In [13] it has been shown that the node search number is equal to the vertex separation plus 1, i.e. $ns(G) = vs(G) + 1$. Furthermore given an optimal layout L of G an algorithm is presented which transforms L into an optimal node search strategy of G in linear time in the size of G.
Actually the straightforward implementations of the algorithms given by [7, 13] do not run in linear time in the size of G but it is easy to find a suitable data structure based on lists which guarantees linear time.
In [12] it has been shown that the interval thickness and the node search number of G coincide, i.e. $\theta(G) = ns(G)$. Furthermore the authors have presented an algorithm which transforms every optimal node search strategy S of G into an optimal interval augmentation of G in linear time in the size of S (and therefore in the size of G). The idea is to associate every vertex v of G with the interval $[i, j]$ where i is the step of the strategy at which v is visited by a searcher and j is the step at which the searcher is removed from v.
Summarizing we get: The above parameters are almost the same for every graph, i.e. $es(G) = vs(G') \leq vs(G) + 2$ and $ns(G) = vs(G) + 1 = \theta(G) + 1$. Moreover given a

graph G the construction of an optimal edge search strategy, an optimal node search strategy, and an optimal interval augmentation of G is possible in $O(t(n))$ time where $t(n)$ is the time required by the construction of an optimal layout of G.

Out of the complexity point of view given an input graph G and an integer l the problem whether or not $vs(G) \leq l$ and the problem whether or not $es(G) \leq l$ are both NP-complete, see [15] and [17] respectively. As a result of the discussion above we obtain that the problems whether or not $ns(G) \leq l$, and $\theta(G) \leq l$ are also NP-complete. They remain NP-complete even for chordal graphs [11], starlike graphs [11], and for planar graphs with maximum degree 3. The last claim is obtained from the fact that (i) the edge search number and the cut-width coincide for graphs with maximum degree 3 [16] and (ii) the Min-Cut problem is NP-complete for planar graphs with maximum degree 3 [18].
For some special graphs the above problems can be solved in polynomial time. Examples are cographs [4], permutation graphs [3], and k-trees for a fixed k [2]. Especially, if l is fixed then the problems are solvable in linear time (but exponential in l) [1].

For trees the vertex separation and the edge search number can be computed in linear time, see [7] and [17] respectively. Consequently the node search number and the interval thickness can be also computed in linear time.
However the computation of an optimal layout (an optimal edge search strategy, an optimal node search strategy, and an optimal interval augmentation) is more complicated. The best known algorithms need $O(n \log n)$ time [7, 17]. It is an open question whether this time can be reduced to $O(n)$.

In this paper we give an algorithm constructing optimal layouts of trees in linear time. As a consequence of this result and the discussion above the construction of an optimal edge search strategy, an optimal node search strategy, and an optimal interval augmentation is possible in linear time for any input tree.

2. Preliminaries. We consider undirected graphs $G = (V, E)$. For convenience loops and multiple edges are disallowed.
A *(linear) layout* of G is a one to one mapping $L : V \to \{1, 2, \ldots, |V|\}$. For a layout L of G and an integer i with $1 \leq i \leq |V|$, let $V_L(i)$ be the vertices of G which are mapped to integers less than or equal to i and adjacent to a vertex mapped to an integer larger than i. Formally $V_L(i) = \{\ u \mid \exists (u, v) \in E \text{ with } L(u) \leq i \text{ and } L(v) > i\ \}$. The *vertex separation* of G with respect to L, denoted by $vs_L(G)$, is the maximum number of vertices in any $V_L(i)$, i.e. $vs_L(G) = \max\limits_{1 \leq i \leq |V|} \{\ |V_L(i)|\ \}$. The *vertex separation* of G, denoted by $vs(G)$, is the minimum vertex separation over all layouts of G, i.e. $vs(G) = \min\limits_{L} \{vs_L(G)\}$. For example the vertex separation of a single-vertex graph is 0, of a chain 1, of a cycle 2, of a star graph 2, and of a k-clique $k-1$.
A layout L of G is *optimal* if $vs(G) = vs_L(G)$.

In this paper we deal with trees. A *tree* T is a connected acyclic graph. For a vertex u of T the connected components obtained by the removal of u from T are called the *branches* of T at u.
In [7] the following lemma has been shown.

LEMMA 2.1. *Let T be a tree and $k \geq 1$ an integer. Then $vs(T) \leq k$ if and only if for all vertices u of T at most two branches of T at u have vertex separation k and all other branches have vertex separation at most $k-1$.*

We directly obtain the following corollary.

COROLLARY 2.2. *Let T be a tree with n vertices. Then $vs(T) \leq \log_3 n + 1 < \log n$.*

Since the vertex separation of a rooted tree is identical to the vertex separation of the underlying unrooted tree from now on we will consider rooted trees only. Then the father and son relation is well defined with respect to the root.

Let T be a (rooted) tree and u a vertex of T. The subtree $T[u]$ of T is rooted at u and consists of u and all descendants of u in T. The subtrees rooted at the sons of u are called the subtrees of u. Let $T[u]$ be a subtree of T and $v_1, \ldots, v_k$ vertices of $T[u]$. We denote by $T[u; v_1, \ldots, v_k]$ the tree obtained from $T[u]$ by removing the subtrees $T[v_1], \ldots, T[v_k]$. Notice that $T[u; v_1, \ldots, v_k]$ is rooted at u.

Next we define critical vertices and vertex labelling.

DEFINITION 2.3. *Let T be a tree with $vs(T) = k$. A vertex v is k-critical in T if and only if v has exactly two sons v_1 and v_2 with $vs(T[v_1]) = vs(T[v_2]) = k$.*

Observe that by Lemma 2.1 v can not have more than two sons with vertex separation k.

LEMMA 2.4. *Let T be a tree with $vs(T) = k$. At most one vertex in T is k-critical.*

Proof. Suppose that two vertices v and v' are k-critical in T. Then v or v' has three branches with vertex separation k which contradicts to Lemma 2.1. □

The following labelling technique has been introduced in [7]. Similar methods were used in [20, 5, 17].

DEFINITION 2.5. *Let T be a tree rooted at u with $vs(T) = k_1$. A list of integers $l = (k_1, k_2, \ldots, k_p)$ with $k_1 > k_2 > \ldots > k_p \geq 0$ is a label of u if there is a set of vertices $\{v_1, \ldots, v_p\}$ in T such that*

(i) *For $1 \leq i < p$, $vs(T[u; v_1, \ldots, v_i]) = k_{i+1}$,*

(ii) *For $1 \leq i < p$, v_i is a k_i-critical vertex in $T[u; v_1, \ldots, v_{i-1}]$,*

(iii) *v_p is u. Moreover if k_p is marked with a prime $(')$ then there is no k_p-critical vertex in $T[u; v_1, \ldots, v_{p-1}]$. If k_p is not marked with a prime then v_p is a k_p-critical vertex.*

Notice that the tree $T[u; v_1, \ldots, v_p]$ is the empty tree and that the vertex separation of T is the largest integer occurring in l.

If we do not care whether or not the last element of a label is critical then we denote the label by $(k_1, k_2, \ldots, k_p^*)$.

The label $l = (k_1, k_2, \ldots, k_p^*)$ of the root of the tree and the corresponding vertices $v_1, \ldots, v_p$ are well defined.

LEMMA 2.6. *Let T be a tree rooted at u with $vs(T) = k_1$. Then both the label $l = (k_1, k_2, \ldots, k_p^*)$ of u and the corresponding vertices $v_1, \ldots, v_p$ of T are unique.*

Proof. We prove this by induction on k_1.

If $k_1 = 0$ then T consists of a single vertex u. Hence there is only one possibility to define l and v_1 namely $l = (0')$ and $v_1 = u$.

For the induction step consider a tree T with $vs(T) = k_1 > 0$.

If T has no k_1-critical vertex then by Definition 2.5 l is (k_1') and v_1 is u. Thus both the label of u and the vertex v_p are unique.

If T has a k_1-critical vertex v_1 then the uniqueness of v_1 follows from Lemma 2.4.

If $v_1 = u$ then $l = (k_1)$ and the label is unique. If $v_1 \neq u$ then by Definition 2.5 we have $vs(T[u; v_1]) = k_2 < k_1$. By the inductive hypothesis both the label $(k_2, \ldots, k_p^*)$ of u in $T[u; v_1]$ and the vertices $v_2, \ldots, v_p$ are unique. Hence both the label $l = (k_1, k_2, \ldots, k_p^*)$ of u and the corresponding vertices $v_1, \ldots, v_p$ of T are unique. □

Let T be a tree rooted at u and $u_1, \ldots, u_d$ the sons of u. Furthermore let l_i be the label of u_i in $T[u_i]$ for $1 \leq i \leq d$.

In [7] a linear algorithm *compute_label* is given which computes the label l of u

in T. Since the vertex separation is the largest integer occurring in l this algorithm also computes the vertex separation of T in linear time. The idea of the algorithm is to compute l bottom up. The label l is computed by recursively computing the labels l_i and then by combining them.
The combination is realized by a function $combine\_labels(l_1, \ldots, l_d)$. To achieve linear time the labels are represented as lists of blocks. A block contains consecutive integers and is represented by an interval consisting of the endpoints of the block in decreasing order. For example the vertex label $l = (9,8,7,6,4,3,2,0')$ is represented by $\lambda = ((9,6),(4,2),(0',0'))$. We call this representation of a label *block representation*.

Let us recall the function *compute_label* of [7].

function $compute\_label(T : tree; u : vertex) : label$;
  if (u is the only vertex of T) **then** $\lambda := ((0',0'))$;
    return λ
  else { Let $u_1, \ldots, u_d$ be the sons of u in T };
    for all u_i **do** $\lambda := compute\_label(T[u_i], u_i)$;
    return $combine\_lists(u, \lambda_1, \ldots, \lambda_d)$;

The proof of the linear time complexity of *compute_labels* is based on the following lemma, see [7] for more details.

LEMMA 2.7. *Let T be a tree rooted at u and $u_1, \ldots, u_d$ the sons of u. Let l_i be the label of u_i in $T[u_i]$ and m_i be the largest integer occurring in l_i for $1 \le i \le d$. Finally let $m_t = \max_{1 \le i \le d} \{m_i\}$ and $m_s = \max_{1 \le i \ne t \le d} \{m_i\}$ if $d > 1$, otherwise $m_s = 0$. The time required by combine_labels to compute the label of u is*

$$O(d + 2m_s + \sum_{1 \le i \ne s,t}^{d} m_i).$$

However the computation of an optimal layout of T is more complicated. The best known algorithm needs $O(n \log n)$ time [7] where it was stated as an open question whether or not there exists a linear time layout algorithm. In the following we give an affirmative answer.

3. Optimal layouts in linear time. In this section we establish a linear time algorithm which constructs optimal layouts of trees. It is a bottom up algorithm and can be seen as an extended version of the vertex separation algorithm. Again the idea is to construct the layout of a tree by first constructing the layouts of the subtrees of its root and then by combining them.

First we show how to construct an optimal layout of certain trees (we call them simple trees) using the optimal layouts of its subtrees. Then we extend the construction to arbitrary trees. Finally we present our algorithm and its data structure.

For the sake of convenience we define a layout of a tree by a list of vertices. For example the list $(v_1, \ldots, v_n)$ corresponds to the layout $L(v_i) = i$ for $1 \le i \le n$ and vice versa. So we may use the concatenation operator & of lists which makes the proofs of our claims more readable.

3.1. Simple trees and their optimal layouts. Here we introduce and investigate *simple trees.*

DEFINITION 3.1. *Let T be a tree with $vs(T) = k$. T is k'-simple if the label of its root is (k'). T is k-simple if the label of its root is (k).*

Notice that by Definition 2.5 every $0'$-simple tree consists of a single vertex and that there are no 0-simple trees.

LEMMA 3.2. *Let T be a k'-simple tree rooted at u. Then at most one of the subtrees of u in T is k'-simple and all other subtrees have vertex separation at most $k-1$.*

Proof. By Definition 2.5 there is no k-critical vertex in T. Thus u is no k-critical vertex in T and therefore u has at most one subtree, say $T[u_i]$, with vertex separation k and all other subtrees have vertex separation at most $k-1$. Suppose that $T[u_i]$ is not k'-simple. Since $vs(T[u_i]) = k$, $T[u_i]$ contains a k-critical vertex v. But then T has a k-critical vertex which is a contradiction because T has no k-critical vertex.

Hence at most one of the subtrees of u is k'-simple and all other subtrees have vertex separation at most $k-1$. □

LEMMA 3.3. *Let T be a k-simple tree rooted at u. Then exactly two of the subtrees of u in T are k'-simple and all other subtrees have vertex separation at most $k-1$.*

Proof. By Definition 2.5 $vs(T) = k$ and the vertex u is a k-critical vertex in T. Thus u has exactly two subtrees, say $T[u_i]$ and $T[u_j]$, with vertex separation k and all other subtrees have vertex separation at most $k-1$. Suppose some of these two subtrees, say $T[u_i]$, is not k'-simple. Since $vs(T[u_i]) = k$, $T[u_i]$ contains a k-critical vertex v. But then there are two different k-critical vertices in T which is a contradiction to Lemma 2.4.

Hence u has exactly two k'-simple subtrees and all other subtrees have vertex separation at most $k-1$. □

We will recursively construct an optimal layout of a tree by combining the optimal layouts of the subtrees of the root. Moreover we will invest no more than linear time for the whole process. This is only possible if the optimal layouts of the subtrees have a suitable form. For the simple trees considered here we introduce optimal left and right extendable layouts of their k'-simple subtrees. As we will see this is sufficient for the later considered arbitrary trees too.

For a tree T let T^w be the tree obtained from T by adding a new vertex w and an edge between w and the root of T.

DEFINITION 3.4. *Let T be a k'-simple tree and L an optimal layout of T. If $k = 0$ then L is both left and right extendable. If $k > 0$ then L is left extendable if $vs_{(w)\&L}(T^w) = k$ and right extendable if $vs_{L\&(w)}(T^w) = k$.*

Notice that by definition left and right extendable layouts are optimal. For the case $k = 0$ remember that every $0'$-simple tree has only one vertex and therefore only one layout which is optimal.

The following lemma shows that every k'-simple tree has a left and a right extendable layout.

LEMMA 3.5. *Every k'-simple tree T has a left extendable layout $le(L)$ and a right extendable layout $ri(L)$.*

Proof. Let T be a tree rooted at u and $u_1, \ldots, u_d$ the sons of u. The proof is by induction on the height h of T.

For $h = 0$ the claim is clear since Definition 3.4. For $h = 1$ the tree T is a star graph with vertex separation $k = 1$.
Define

$$le(L) = ri(L) = (u)\&(u_{\pi(1)})\&(u_{\pi(2)})\& \cdots \&(u_{\pi(d)})$$

where π is a permutation over $(1, 2, \ldots, d)$, see Figure 3.1 (a). Obviously $le(L)\&(w)$ and $(w)\&ri(L)$ are layouts of T^w with vertex separation 1. Thus the layout $le(L) = ri(L)$ is a left as well as a right extendable layout of T.

For the induction step consider T of height $h+1$. By Lemma 3.2 at most one subtree of u is k'-simple and all other subtrees have vertex separation at most $k-1$. If there exists a k'-simple subtree of u, say $T[u_d]$, then by the inductive hypothesis $T[u_d]$ has a left extendable layout $le(L_d)$ and a right extendable layout $ri(L_d)$. Define

$$le(L) = (u)\&L_{\pi(1)}\&L_{\pi(2)}\&\cdots\&L_{\pi(d-1)}\&le(L_d)$$

and

$$ri(L) = ri(L_d)\&(u)\&L_{\pi(1)}\&L_{\pi(2)}\&\cdots\&L_{\pi(d-1)}$$

where π is a permutation over $(1,\ldots,d-1)$ and $L_1,\ldots,L_{d-1}$ are optimal layouts of $T[u_1]$, $\ldots,T[u_{d-1}]$ of u, see Figure 3.1 (b) and (c). Consider layout $le(L)$. For every $1 \leq i \leq d-1$ observe that the vertex separation of $T[u_{\pi(i)}]$ in $le(L)$ is at most $k-1$ and that u is the only vertex to the left of $T[u_{\pi(i)}]$ which is connected to $T[u_{\pi(i)}]$ or to a vertex to the right of $T[u_{\pi(i)}]$. Next observe that the connection between u and the root u_d of $T[u_d]$ does not increase the vertex separation of T in $le(L)$ since $le(L_d)$ is left extendable. Thus the vertex separation of T with respect to $le(L)$ is k and thus optimal. Moreover the vertex separation of T^w with respect to the layout $(w)\&le(L)$ is k since w is to the left of u and is connected to u only. Hence $le(L)$ is left extendable. A similar argument as before shows that $ri(L)$ is right extendable too.
If there does not exist a k'-simple subtree of u then by Lemma 3.2 all subtrees have vertex separation at most $k-1$.
Define

$$le(L) = ri(L) = (u)\&L_{\pi(1)}\&L_{\pi(2)}\&\cdots\&L_{\pi(d)}$$

where π is a permutation over $(1,\ldots,d)$ and $L_1,\ldots,L_d$ are optimal layouts of the subtrees $T[u_1],\ldots,T[u_d]$ of u, see Figure 3.1 (d). In the same way as before it can be shown that $le(L) = ri(L)$ is a left as well as a right extendable layout of T. □

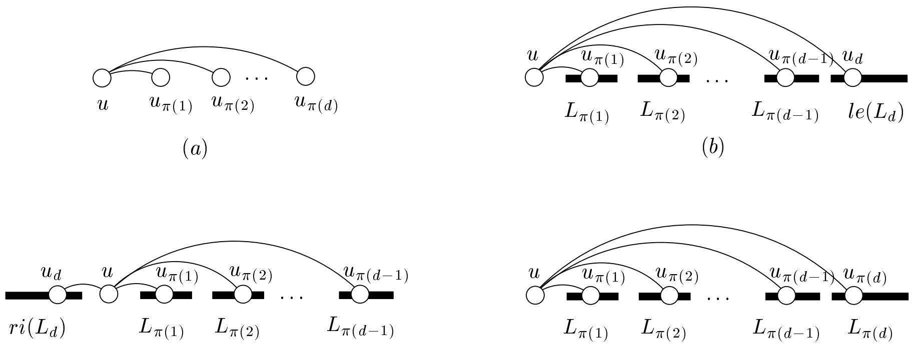

FIGURE 3.1. *Construction of left and right extendable layouts of a k'-simple tree. The thick lines represent layouts.*

The next lemma follows immediately.
LEMMA 3.6. *Let T be a k'-simple tree rooted at u and $u_1,\ldots,u_d$ the sons of u.*

(i) *If T has a k'-simple subtree, say $T[u_d]$, then for every permutation π over* $(1, \ldots, d-1)$

$$le(L) = (u)\&L_{\pi(1)}\&L_{\pi(2)}\& \cdots \&L_{\pi(d-1)}\&le(L_d)$$

is a left extendable layout and

$$ri(L) = ri(L_d)\&(u)\&L_{\pi(1)}\&L_{\pi(2)}\& \cdots \&L_{\pi(d-1)}$$

is a right extendable layout of T where $le(L_d)$ and $ri(L_d)$ are left and right extendable layouts of $T[u_d]$ and $L_1, \ldots, L_{d-1}$ are optimal layouts of the other subtrees $T[u_1], \ldots, T[u_{d-1}]$ of u,

(ii) *If T has no k'-simple subtree then for every permutation π over* $(1, \ldots, d)$

$$le(L) = ri(L) = (u)\&L_{\pi(1)}\&L_{\pi(2)}\& \cdots \&L_{\pi(d)}$$

is a left as well as a right extendable layout of T where $L_1, \ldots, L_d$ are optimal layouts of the subtrees $T[u_1], \ldots, T[u_d]$ of u.

The lemma provides a construction scheme for a left and a right extendable layout of a k'-simple tree rooted at u using the optimal layouts of the subtrees of u especially the left and right extendable layout of the k'-simple subtree.

The next lemma shows a similar construction for an optimal layout of a k-simple tree.

LEMMA 3.7. *Let T be a k-simple tree rooted at u and $T[u_{d-1}]$ and $T[u_d]$ be the k'-simple subtrees of u. For every permutation π over* $(1, \ldots, d-2)$

$$L = ri(L_{d-1})\&(u)\&L_{\pi(1)}\&L_{\pi(2)}\& \cdots \&L_{\pi(d-2)}\&le(L_d)$$

is an optimal layout of T where $ri(L_{d-1})$ is a right extendable layout of $T[u_{d-1}]$, $le(L_d)$ is a left extendable layout of $T[u_d]$, and $L_1, \ldots, L_{d-2}$ are optimal layouts of the subtrees $T[u_1], \ldots, T[u_{d-2}]$.

Proof. By Lemma 3.3 exactly two subtrees of u, say $T[u_{d-1}]$ and $T[u_d]$, are k'-simple and all other subtrees have vertex separation at most $k-1$. Consider layout L shown in Figure 3.2. The vertex separation of every non k'-simple subtree $T[u_{\pi(i)}]$ in L is at most $k-1$ and u is the only vertex to the left of $T[u_{\pi(i)}]$ which is connected to $T[u_{\pi(i)}]$ or to a vertex to the right of $T[u_{\pi(i)}]$. Furthermore the connections between u and the roots of the k'-simple trees do not increase the vertex separation of T in L since $ri(L_{d-1})$ is right extendable and $le(L_d)$ is left extendable. Thus the vertex separation of T with respect to L is k and thus optimal. □

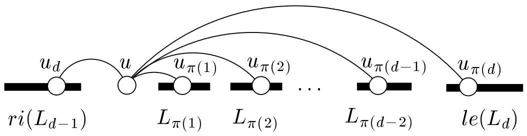

FIGURE 3.2. *Construction of the optimal layout of a k-simple tree. The thick lines represent layouts.*

Similar as before we can show that $L' = ri(L_d)\&(u)\&L_{\pi(1)}\&L_{\pi(2)}\& \cdots \&L_{\pi(d-2)}\& le(L_{d-1})$ is also an optimal layout of T. For our intention the layout L is sufficient.

Until now we have shown how to construct optimal layouts of simple trees by combining optimal layouts of the subtrees of their roots. In the next subsection we extend the construction to arbitrary trees.

3.2. Arbitrary trees and their optimal layouts. The main idea is to partition a tree into smaller parts such that all of them are simple trees.

DEFINITION 3.8. *Let T be a tree rooted at u and $l = (k_1, k_2, \ldots, k_p^*)$ the label of u in T. Let $v_1, \ldots, v_p$ be the k_i-critical vertices of T according to Definition 2.5. The parts $\widetilde{T}_{k_i}$, $1 \le i \le p$, of T are defined as follows, see Figure 3.3.*

(i) *$\widetilde{T}_{k_1}$ is the subtree $T[v_1]$,*

(ii) *$\widetilde{T}_{k_i}$ is the subtree of $T[u; v_1, \ldots, v_{i-1}]$ rooted at v_i for $1 \le i \le p$.*

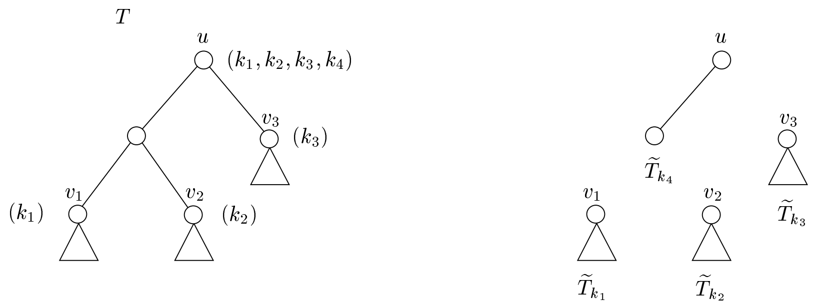

FIGURE 3.3. *The parts $\widetilde{T}_{k_i}$ of T*

LEMMA 3.9. *Let T be a tree rooted at u and $l = (k_1, k_2, \ldots, k_p^*)$ the label of u. Then all parts $\widetilde{T}_{k_1}, \ldots, \widetilde{T}_{k_p}$ of T are simple trees. Moreover for every $1 \le i \le p-1$ the part $\widetilde{T}_{k_i}$ is k_i-simple.*

Proof. The proof is by induction on p.

For $p = 1$ the part $\widetilde{T}_{k_p}$ is the tree T which is simple by definition.

For the induction step consider the part $\widetilde{T}_{k_1}$ of T. By Definition 2.5 the root v_1 of $\widetilde{T}_{k_1}$ has two subtrees with vertex separation k_1. So $vs(\widetilde{T}_{k_1}) \ge k_1$. On the other hand $vs(\widetilde{T}_{k_1}) \le k_1$ since the vertex separation of T is k_1. Thus $vs(\widetilde{T}_{k_1}) = k_1$ and by Definition 2.5 it is k_1-simple.

Now consider $T[u; v_1]$. By Definition 2.5 the label of u in $T[u; v_1]$ is $(k_2, \ldots, k_p^*)$. By the inductive hypothesis all parts $\widetilde{T}_{k_2}, \ldots, \widetilde{T}_{k_p}$ are simple trees and for every $2 \le i \le p-1$ the part $\widetilde{T}_{k_i}$ is k_i-simple. Hence all parts $\widetilde{T}_{k_1}, \ldots, \widetilde{T}_{k_p}$ are simple trees and for every $1 \le i \le p-1$ the part $\widetilde{T}_{k_i}$ is k_i-simple. ☐

If $\widetilde{T}_{k_p}$ is k_p'-simple its left and right extendable layout constructed as in Lemma 3.6 is denoted by $le(\widetilde{L}_{k_p})$ and $ri(\widetilde{L}_{k_p})$. If $\widetilde{T}_{k_i}$ is k_i-simple then its optimal layout constructed as in Lemma 3.7 is denoted by $\widetilde{L}_{k_i}$.

Next we assume that every layout is partitioned into two sublists by one of its vertices called the *focus* of the layout. The first sublist $pre(L)$, the prefix of L, consists of the focus and the vertices of L which are left to the focus. The second sublist $suf(L)$, the suffix of L, consists of the vertices of L which are right to the focus.

We define the focuses of the layouts of the parts $\widetilde{T}_{k_i}$ of T as follows: If $\widetilde{T}_{k_i}$ is k_i-simple the focus of the layout $\widetilde{L}_{k_i}$ is the root of $\widetilde{T}_{k_i}$. If $\widetilde{T}_{k_i}$ is k_i'-simple (this is only possible for $i = p$) the focuses of the layouts $le(\widetilde{L}_{k_i})$ and $ri(\widetilde{L}_{k_i})$ are their first vertices.

Next we define an operation $\oplus$ over such layouts. Given two layouts L and L' and their focuses, the layout $L \oplus L'$ is defined by $L \oplus L' = pre(L)\&L'\&suf(L)$. Furthermore the focus of $L \oplus L'$ is the focus of L'. Notice that operation $\oplus$ is transitive (but not commutative).

Now using the operation $\oplus$ we show that an optimal layout of an arbitrary tree can be constructed using the optimal layouts of its parts.

LEMMA 3.10. *Let T be a tree rooted at u and $l = (k_1, k_2, \ldots, k_p^*)$ the label of u.*

(i) *If k_p is critical in l then $L = \widetilde{L}_{k_1} \oplus \widetilde{L}_{k_2} \oplus \cdots \oplus \widetilde{L}_{k_p}$ is an optimal layout of T,*

(ii) *If k_p is not critical in l then $L = \widetilde{L}_{k_1} \oplus \cdots \oplus \widetilde{L}_{k_{p-1}} \oplus le(\widetilde{L}_{k_p})$ is an optimal layout of T.*

Proof. The proof is by induction on p.

For the basis of the induction let $p = 1$. By definition we have $T = \widetilde{T}_{k_p}$ and the claim trivially holds.

For the induction step consider the tree $T[u; v_1]$ where v_1 is the root of the part $\widetilde{T}_{k_1}$. By Definition 2.5 the label of u in $T[u; v_1]$ is $(k_2, \ldots, k_p^*)$.
Assume that k_p is critical in l. By the inductive hypothesis the layout $L' = \widetilde{L}_{k_2} \oplus \widetilde{L}_{k_3} \oplus \cdots \oplus \widetilde{L}_{k_p}$ is an optimal layout of $T[u; v_1]$ with vertex separation $k_2 < k_1$.
Now consider the layout $L = \widetilde{L}_{k_1} \oplus \widetilde{L}_{k_2} \oplus \cdots \oplus \widetilde{L}_{k_p}$ of T. By the definition of $\oplus$ and its transitivity we have $L = pre(\widetilde{L}_{k_1}) \& (\widetilde{L}_{k_2} \oplus \widetilde{L}_{k_3} \oplus \cdots \oplus \widetilde{L}_{k_p}) \& suf(\widetilde{L}_{k_1})$. Since $L' = \widetilde{L}_{k_2} \oplus \widetilde{L}_{k_3} \oplus \cdots \oplus \widetilde{L}_{k_p}$ we obtain that $L = pre(\widetilde{L}_{k_1}) \& L' \& suf(\widetilde{L}_{k_1})$.
We show that L is an optimal layout of T. By construction v_1 belongs to $pre(\widetilde{L}_{k_1})$ and v_1 is the only vertex to the left of $T[u; v_1]$ in L which is connected to $T[u; v_1]$ or to a vertex to the right of $T[u; v_1]$. Since the vertex separation of $T[u; v_1]$ with respect to L' is less than k_1 the vertex separation of T with respect to L does not exceed k_1. Hence L is optimal.
If k_p is not critical in l then a similar argument shows that $L = \widetilde{L}_{k_1} \oplus \cdots \oplus \widetilde{L}_{k_{p-1}} \oplus le(\widetilde{L}_{k_p})$ is an optimal layout of T. ☐

Notice if k_p is not critical in l then $L' = \widetilde{L}_{k_1} \oplus \cdots \oplus \widetilde{L}_{k_{p-1}} \oplus ri(\widetilde{L}_{k_p})$ is also an optimal layout of T. But for our intention it is sufficient to consider L.

Lemma 3.10 supplies a flexible list representation of an optimal layout of a tree. We will call this list the *coarse layout* of T.

DEFINITION 3.11. *Let T be a tree rooted at u and $l = (k_1, k_2, \ldots, k_p^*)$ the label of u in T. The coarse layout of T is defined as*

$$R = \begin{cases} (\widetilde{L}_{k_1}, \widetilde{L}_{k_2}, \ldots, \widetilde{L}_{k_p}) & \textit{if } k_p \textit{ is critical in } l, \\ (\widetilde{L}_{k_1}, \widetilde{L}_{k_2}, \ldots, \widetilde{L}_{k_{p-1}}, (le(\widetilde{L}_{k_p}), ri(\widetilde{L}_{k_p}))) & \textit{otherwise.} \end{cases}$$

Given the coarse layout of T its optimal layout can be constructed using Lemma 3.10. Hence in order to construct an optimal layout of a tree it is sufficient to construct its coarse layout.

Next we show how to construct the coarse layout of a tree T rooted at u using the label of u and the coarse layouts of its subtrees. We do this in two steps.
In the first step we identify the parts of T which are also parts of the subtrees of u. In fact all parts of T except $\widetilde{T}_{k_p}$ are exactly the parts of the subtrees of u with vertex separation larger than k_p. The optimal layouts of these parts are adopted from the coarse layout of T.
In the second step using the remaining parts of the coarse layouts of the subtrees of u we show how to construct $\widetilde{L}_{k_p}$ (if k_p is critical) or $le(\widetilde{L}_{k_p})$, $ri(\widetilde{L}_{k_p})$ (if k_p is not critical). Then the coarse layout of T is complete.

LEMMA 3.12. *Let T be a tree rooted at u, $l = (k_1, k_2, \ldots, k_p^*)$ the label of u, and $u_1, \ldots, u_d$ the sons of u. Then*

(i) *the parts* $\widetilde{T}_{k_1}, \widetilde{T}_{k_2}, \ldots, \widetilde{T}_{k_{p-1}}$ *of* T *are exactly the parts of* $T[u_1], \ldots, T[u_d]$ *with vertex separation larger than* k_p,

(ii) *no subtree* $T[u_i]$, $1 \leq i \leq d$, *has a* k_p*-simple part.*

Proof. The proof proceeds by induction on the length p of the label l.

For the basis of the induction let $p = 1$. In this case T and $T[u_1], \ldots T[u_d]$ have vertex separation at most k_p. Thus they have no parts with vertex separation larger than k_p. Furthermore by Lemmas 3.2 and 3.3 the subtrees $T[u_1], \ldots T[u_d]$ of u either have vertex separation less than k_p or are k_p'-simple. Since neither trees with vertex separation less than k_p nor k_p'-simple trees have k_p-simple parts, it follows that no subtree $T[u_i]$ of u has a k_p-simple part and the claim holds.

For the induction step let $p > 1$. Consider the k_1-simple part $\widetilde{T}_{k_1}$ of T. Its root v_1 is the k_1-critical vertex in T. By Definition 2.5 $v_1 \neq u$ and therefore v_1 belongs to some subtree of u, say $T[u_1]$. Clearly $vs(T[u_1]) = k_1$ and therefore $\widetilde{T}_{k_1}$ is also the k_1-simple part of $T[u_1]$. Now consider tree $T[u; v_1]$. By Definition 2.5 u has the label $(k_2, \ldots, k_p)$ in $T[u; v_1]$. By the inductive hypothesis the parts $\widetilde{T}_{k_2}, \ldots, \widetilde{T}_{k_{p-1}}$ of $T[u; v_1]$ are exactly the parts of the subtrees of $T[u; v_1]$ with vertex separation larger than k_p and no subtree of $T[u; v_1]$ has a k_p-simple part. Hence the claim holds for this case and the proof is complete. □

As a consequence we obtain the next lemma.

LEMMA 3.13. *Let* T *be a tree rooted at* u, $l = (k_1, k_2, \ldots, k_p^*)$ *the label of* u, *and* $u_1, \ldots, u_d$ *the sons of* u. *Then*

(i) *the layout elements* $\widetilde{L}_{k_1}, \widetilde{L}_{k_2}, \ldots, \widetilde{L}_{k_{p-1}}$ *of the coarse layout of* T *are exactly the layout elements of the coarse layouts of* $T[u_1], \ldots, T[u_d]$ *which correspond to a vertex separation larger than* k_p,

(ii) *no coarse layout of* $T[u_i]$, $1 \leq i \leq d$, *contains a layout corresponding to a* k_p*-simple tree.*

Of course for the construction of the coarse layout of T we have to sort these $p-1$ layouts $\widetilde{L}_{k_1}, \widetilde{L}_{k_2}, \ldots, \widetilde{L}_{k_{p-1}}$ according to the integers $k_1, \ldots, k_{p-1}$. For an efficient sorting we make use of the following fact. Let k_j be an element of l such that $(\widetilde{L}_{k_1}, \widetilde{L}_{k_2}, \ldots, \widetilde{L}_{k_j})$ is a prefix of the coarse layout of the subtree of u with the largest vertex separation. Since $\widetilde{L}_{k_1}, \widetilde{L}_{k_2}, \ldots, \widetilde{L}_{k_j}$ are sorted in the coarse layout of this subtree we only need to sort the remaining $p - j - 1$ elements $\widetilde{L}_{k_{j+1}}, \widetilde{L}_{k_{j+2}}, \ldots, \widetilde{L}_{k_{p-1}}$ occurring in the coarse layouts of $T[u_1], \ldots, T[u_d]$.

Notice that the smaller k_j, the larger j and therefore the smaller the number $p - j - 1$ of the elements of the coarse layouts of $T[u_1], \ldots, T[u_d]$ which must be sorted. We show how to find such an element k_j which is small enough for our later complexity considerations.

First we need the following lemma.

LEMMA 3.14. *Let* T *be a tree rooted at* u *and* $u_1, \ldots, u_d$ *the sons of* u *and* $l_i = (k_{i,1}, k_{i,2}, \ldots, k_{i,p_i}^*)$ *the label of* u_i *in* $T[u_i]$ *for* $1 \leq i \leq d$.
Let $k_{t,1} = \max_{1 \leq i \leq d} \{k_{i,1}\}$, *i.e.* l_t *is the label containing the largest element over all* $k_{i,1}$.
Finally let $k_{s,1} = \max_{1 \leq i \neq t \leq d} \{k_{i,1}\}$ *if* $d > 1$, *otherwise* $k_{s,1} = 0$, *i.e. if* $d > 1$ *then* l_s *is the label containing the second largest element over all* $k_{i,1}$.
If there are consecutive elements $k_{t,j}$ *and* $k_{t,j+1}$ *in* l_t *with* $k_{t,j} > \max\{k_{t,j+1}, k_{s,1}\} + 1$ *then* $vs(T) = k_{t,1}$.

Proof. The proof is by induction on the index j.

For the basis of the induction let $j = 1$. Let T' be the tree obtained from T by removing the part of $T[u_t]$ with vertex separation $k_{t,1}$. Clearly all subtrees of u

in T' have vertex separation at most $k_{t,1}-2$. Let L'_i be an optimal layout of the subtree $T'[u_i]$, $1 \leq i \leq d$. Define $L' = (u)\&L'_1\&L'_2\& \cdots \&L'_d$. An easy inspection shows that the vertex separation of T' with respect to L' is at most $k_{t,1}-1$. By Lemma 3.9 the part $\widetilde{T}_{k_{t,1}}$ of $T[u_t]$ is a $k_{t,1}$-simple tree. Now consider the layout $L = pre(\widetilde{L}_{k_{t,1}})\&L'\&suf(\widetilde{L}_{k_{t,1}})$ where $\widetilde{L}_{k_{t,1}}$ is the optimal layout of $\widetilde{T}_{k_{t,1}}$. The same argument as in Lemma 3.10 shows that the vertex separation of T with respect to L is $k_{t,1}$.

For the induction step let $j > 1$. First remove from T the part $\widetilde{T}_{k_{t,1}}$ of $T[u_t]$. Let T' be the obtained tree. By the inductive hypothesis $vs(T') = k_{t,2} < k_{t,1}$. Let L' be an optimal layout of T'. In the same way as before it can be shown that the vertex separation of T with respect to $L = pre(\widetilde{L}_{k_{t,1}})\&L'\&suf(\widetilde{L}_{k_{t,1}})$ is $k_{t,1}$. Hence $vs(T) = k_{t,1}$ and the claim holds. □

Lemma 3.15. *Let T be a tree rooted at u, $l = (k_1, k_2, \ldots, k_p^*)$ the label of u, and $u_1, \ldots, u_d$ the sons of u. Let $l_i = (k_{i,1}, k_{i,2}, \ldots, k^*_{i,p_i})$ be the label of u_i in $T[u_i]$ for $1 \leq i \leq d$. Finally let $k_{t,1} = \max\limits_{1\leq i\leq d} \{k_{i,1}\}$ and $k_{s,1} = \max\limits_{1\leq i\neq t\leq d} \{k_{i,1}\}$ if $d > 1$, otherwise $k_{s,1} = 0$.*
Then $(\widetilde{L}_{k_1}, \widetilde{L}_{k_2}, \ldots, \widetilde{L}_{k_j}) = (\widetilde{L}_{k_{t,1}}, \widetilde{L}_{k_{t,2}}, \ldots, \widetilde{L}_{k_{t,j}})$ where $k_{t,j}$ and $k_{t,j+1}$ are the smallest consecutive elements of l_t with $k_{t,j} > \max\{k_{t,j+1}, k_{s,1}\} + 1$.

Proof. The proof is by induction on p.

For the basis of the induction let $p = 1$. If T is k_p-simple then by Lemma 3.3 $l_t = (k'_p)$ and there are no consecutive elements in l_t. Thus the claim trivially holds. If T is k'_p-simple then there are two cases.
If T has a k'_p-simple subtree then $l_t = (k'_p)$ and as before the claim trivially holds.
If T has no k'_p-simple subtree then by Lemma 3.2 all subtrees of T have vertex separation at most $k_p - 1$. We show that in this case there are no consecutive elements $k_{t,j}$ and $k_{t,j+1}$ in l_t with $k_{t,j} > \max\{k_{t,j+1}, k_{s,1}\} + 1$. Suppose that l_t has more than two elements and that $k_{t,j}$ and $k_{t,j+1}$ exist. By Lemma 3.14 $vs(T) = k_{t,1} \leq k_p - 1$ which is a contradiction because $vs(T) = k_p$. Thus in this case the claim holds.

For the induction step let $p > 1$. By Lemma 3.14 $vs(T) = k_{t,1}$. Thus $k_{t,1} = k_1$. By Lemma 2.4 the roots of $\widetilde{T}_{k_{t,1}}$ and $\widetilde{T}_{k_1}$ coincide and therefore $\widetilde{L}_{k_{t,1}} = \widetilde{L}_{k_1}$. Let T' be the tree obtained from T by removing the part $\widetilde{T}_{k_1}$. By Definition 2.5 the label of u in T' is $(k_2, \ldots, k_p^*)$. By the inductive hypothesis $(\widetilde{L}_{k_2}, \ldots, \widetilde{L}_{k_j}) = (\widetilde{L}_{k_{t,2}}, \ldots, \widetilde{L}_{k_{t,j}})$. Thus $(\widetilde{L}_{k_1}, \widetilde{L}_{k_2}, \ldots, \widetilde{L}_{k_j}) = (\widetilde{L}_{k_{t,1}}, \widetilde{L}_{k_{t,2}}, \ldots, \widetilde{L}_{k_{t,j}})$ and the proof is complete. □

We continue with the second step and show how to construct the last element $\widetilde{L}_{k_p}$ or $(le(\widetilde{L}_{k_p}), ri(\widetilde{L}_{k_p}))$ of the coarse layout of T.

Lemma 3.16. *Let T be a tree rooted at u and $l = (k_1, k_2, \ldots, k_p^*)$ the label of u. Let $u_1, \ldots, u_d$ be the sons of u and $l_i = (k_{i,1}, k_{i,2}, \ldots, k^*_{i,p_i})$ the label of u_i in $T[u_i]$ for $1 \leq i \leq d$. Let R'_i be the list obtained from the coarse layout R_i of $T[u_i]$ by removing all components $\widetilde{L}_{k_{i,j}}$ with $k_{i,j} > k_p$.*
Then the element $\widetilde{L}_{k_p}$ or $(le(\widetilde{L}_{k_p}), ri(\widetilde{L}_{k_p}))$ of the coarse layout of T can be constructed using the elements of R'_i, $1 \leq i \leq d$.

Proof. Let $k_{i,j}$ be the smallest element of l_i larger than k_p for $1 \leq i \leq d$. If k_{i,p_i} is critical in l_i then $R'_i = (\widetilde{L}_{k_{i,j+1}}, \widetilde{L}_{k_{i,j+2}}, \ldots, \widetilde{L}_{k_{i,p_i}})$ otherwise $R'_i = (\widetilde{L}_{k_{i,j+1}}, \widetilde{L}_{k_{i,j+2}}, \ldots, \widetilde{L}_{k_{i,p_i-1}}, (le(\widetilde{L}_{k_{i,p_i}}), ri(\widetilde{L}_{k_{i,p_i}})))$. By Lemma 3.12($i$) $R'_1, R'_2, \ldots, R'_d$ are the coarse layouts of the subtrees of $\widetilde{T}_{k_p}$. There are two cases to consider.

Case 1: $\widetilde{T}_{k_p}$ is k'_p-simple. By Lemma 3.3 at most one subtree of $\widetilde{T}_{k_p}$ is k'_p-simple and all other subtrees have vertex separation $\leq k_p - 1$.

Assume $\widetilde{T}_{k_p}$ has a k'_p-simple subtree, say $\widetilde{T}_{k_p}[u_d]$. For $1 \le i \le d-1$ define $L'_i = \widetilde{L}_{k_{i,j+1}} \oplus \widetilde{L}_{k_{i,j+2}} \oplus \cdots \oplus \widetilde{L}_{k_{i,p_i}}$ if k_{i,p_i} is critical in l_i and $L'_i = \widetilde{L}_{k_{i,j+1}} \oplus \widetilde{L}_{k_{i,j+2}} \oplus \cdots \oplus \widetilde{L}_{k_{i,p_i-1}} \oplus le(\widetilde{L}_{k_{i,p_i}})$ if k_{i,p_i} is not critical in l_i. By Lemma 3.10 the layout L'_i is an optimal layout of $\widetilde{T}_{k_p}[u_i]$ for $1 \le i \le d-1$. Furthermore, since Lemma 3.6(i), we have $le(\widetilde{L}_{k_p}) = (u)\&L'_1\&L'_2\& \cdots \&L'_{d-1}\&le(L'_d)$ and $ri(\widetilde{L}_{k_p}) = ri(L'_d)\&(u)\&L'_1\&L'_2\&\cdots\&L'_{d-1}$. Notice that $(le(L'_d), ri(L'_d))$ is the only element of the coarse layout R'_d of the subtree $\widetilde{T}_{k_p}[u_d]$). Now assume that $\widetilde{T}_{k_p}$ has no k'-simple subtree. For $1 \le i \le d$ define L'_i as before. By Lemma 3.10 L'_i is an optimal layout of $\widetilde{T}_{k_p}[u_i]$ for $1 \le i \le d$. By Lemma 3.6(ii) we have $le(\widetilde{L}_{k_p}) = ri(\widetilde{L}_{k_p}) = (u)\&L'_1\&L'_2\&\cdots\&L'_d$.

Case 2: $\widetilde{T}_{k_p}$ is k_p-simple. By Lemma 3.3 exactly two subtrees of $\widetilde{T}_{k_p}$, say $\widetilde{T}_{k_p}[u_{d-1}]$ and $\widetilde{T}_{k_p}[u_d]$, are k'_p-simple and all other subtrees have vertex separation at most $k_p - 1$. For $1 \le i \le d-2$ let L_i be defined as before. By Lemma 3.10 the layout L'_i is an optimal layout of $\widetilde{T}_{k_p}[u_i]$ for $1 \le i \le d-2$. By Lemma 3.7 we have $\widetilde{L}_{k_p} = ri(L'_{d-1})\&(u)\&L'_1\&L'_2\&\cdots\&L'_{d-2}\&\ le(L'_d)$. Observe that $ri(L'_{d-1})$ is the second component of the only element of R'_{d-1} and $le(L'_d)$ is the first component of the only element of R'_d.

Hence the last element $\widetilde{L}_{k_p}$ or $(le(\widetilde{L}_{k_p}), ri(\widetilde{L}_{k_p}))$ of the coarse layout of T can be constructed using the elements of the coarse layouts of the subtrees of u. □

Now we are able to present our linear time algorithm constructing an optimal layout of an input tree.

3.3. Linear time layout Algorithm. Consider a tree T rooted at u. We can assume that using the function *compute_labels* of [7] every vertex v of T is marked by the last element of its label. For example let $l = (k_1, \ldots, k_p^*)$ be the label of v. Then if k_p is critical in l the mark of v is k_p otherwise k'_p.

Let $u_1, \ldots, u_d$ be the sons of u. Our aim is to compute an optimal layout L of T where L is a list of the vertices of T. Our algorithm works as follows. First it recursively computes the coarse layouts R_i of the subtrees $T[u_i]$. Then using the mark of u and the coarse layouts R_i it computes the coarse layout R of T. At the end the algorithm computes L from R.

To achieve linear time we need a suitable data structure for the coarse layout R of T. We represent R by a list Λ according to the block representation of the label l of u given in [7].

Let $\lambda = ((m_1, n_1), (m_2, n_2), \ldots, (m_{r-1}, n_{r-1}), (m_r, n_r))$ be the block representation of l where $m_r = n_r$; otherwise we split the last interval (m_r, n_r) into two intervals $(m_r, n_r - 1)$ and (n_r, n_r).

Consider the coarse layout R of T. If m_r is critical in l then R has the form $(\widetilde{L}_{m_1}, \ldots, \widetilde{L}_{n_1},\ \ldots\ , \widetilde{L}_{m_{r-1}}, \ldots, \widetilde{L}_{n_{r-1}}, \widetilde{L}_{m_r})$ otherwise the form $(\widetilde{L}_{m_1}, \ldots, \widetilde{L}_{n_1},\ \ldots\ , \widetilde{L}_{m_{r-1}}, \ldots, \widetilde{L}_{n_{r-1}}, (le(\widetilde{L}_{m_r}), ri(\widetilde{L}_{m_r})))$.

Λ is a list of r records, one for every interval (m_i, n_i). Every record consists of four fields. The first two are of type integer and the last two are pointers, see Figure 3.4.

For $1 \le i \le r-1$, see Figure 3.4 (a), the first field of the i-th record is reserved for m_i, the second for n_i, the third for a pointer to the layout $\widetilde{L}_{m_i,n_i} = \widetilde{L}_{m_i} \oplus \widetilde{L}_{m_i-1} \oplus \cdots \oplus \widetilde{L}_{n_i}$, and the fourth for a pointer to the focus f_i of $\widetilde{L}_{m_i,n_i}$.

For $i = r$ there are two cases. If m_r is critical in l, see Figure 3.4 (b), then the r-th record has the same structure as before, but now both integer fields are reserved for m_r since in this case $\widetilde{L}_{m_r,m_r} = \widetilde{L}_{m_r}$.

If m_r is not critical in l, see Figure 3.4 (c), then the first two fields of the r-th record are reserved for m'_r, the third field is reserved for a pointer to $le(\widetilde{L}_{m_r})$, and the fourth for a pointer to $ri(\widetilde{L}_{m_r})$ where $le(\widetilde{L}_{m_r})$ and $ri(\widetilde{L}_{m_r})$ are the left and right extendable layouts of the m'_r-simple part $\widetilde{T}_{m_r}$ of T. In this case we need no pointers to the focuses of $le(\widetilde{L}_{m_r})$ and $ri(\widetilde{L}_{m_r})$ because the focuses are always the first vertices.

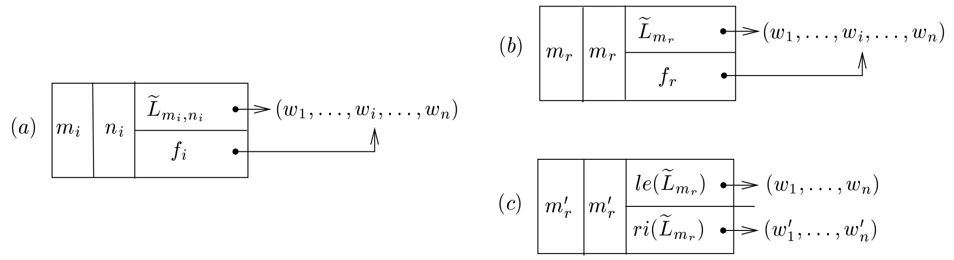

FIGURE 3.4. *The records of an $L-list$ Λ*

We say that Λ is of type *L_list*. This data structure allows the construction of the prefix and suffix of every $\widetilde{L}_{m_i,n_i}$ in constant time.
The next observation concerns coarse layouts and their *L_list* representations.

OBSERVATION 3.17. *Let T be a tree, R a coarse layout of T, and Λ the L_list representation of R. The elements of R are basically the layouts in the layout fields of the records of Λ. The only difference is that some elements $\widetilde{L}_{m_i}, \widetilde{L}_{m_i-1}, \ldots, \widetilde{L}_{n_i}$ of R are represented in a layout field of Λ by $\widetilde{L}_{m_i,n_i} = \widetilde{L}_{m_i} \oplus \widetilde{L}_{m_i-1} \oplus \cdots \oplus \widetilde{L}_{n_i}$. In fact this is done for every block (m_i, n_i) of λ except possibly for the last block (m_r, n_r). For the last block (m_r, n_r) only the elements $\widetilde{L}_{m_r}, \widetilde{L}_{m_r-1}, \ldots, \widetilde{L}_{n_r-1}$ of R are represented by $\widetilde{L}_{m_r,n_r-1} = \widetilde{L}_{m_r} \oplus \widetilde{L}_{m_r-1} \oplus \cdots \oplus \widetilde{L}_{n_i-1}$.*

Now we present some elementary subroutines used by our main algorithm.

function *grind_coarse_layout*$(\Lambda : L\_list) : layout$;
It computes the optimal layout of a tree from the *L_list* representing its coarse layout according to Lemma 3.10.

function *split_L_list*$(\Lambda : L\_list; k : integer) : (L\_list, L\_list)$;
It splits an L_list $\Lambda = ((m_1, n_1, X_1, Y_1), \ldots, (m_r, n_r, X_r, Y_r))$ into the *L_lists* $R = ((m_1, n_1, X_1, Y_1), \ldots, (m_j, n_j, X_j, Y_j))$ and $R' = ((m_{j+1}, n_{j+1}, X_{j+1}, Y_{j+1}), \ldots, (m_r, n_r, X_r, Y_r))$ where n_j is the smallest integer over all $n_i > k$ where $1 \le i \le r$.

function *compact*$(R : list\ of\ records) : L\_list$;
It transforms a list of records $(m_1, n_1, X_1, Y_1), \ldots, (m_r, n_r, X_r, Y_r)$ sorted in decreasing order according to the integers $m_1, \ldots, m_r$ into an *L_list*. It replaces every segment $(m_k, n_k, X_k, Y_k), (m_{k+1}, n_{k+1}, X_{k+1}, Y_{k+1}), \ldots, (m_l, n_l, X_l, Y_l)$ with $n_i = m_{i+1} + 1$ for $k \le i < l$ by the record (m_k, n_l, X, Y) where $X = X_k \oplus X_{k+1} \oplus \cdots \oplus X_l$ and $Y = Y_l$.

function *construct_last_component*$(u : vertex; \Lambda_1, \ldots, \Lambda_d : L\_list;) : L\_list$;
It computes the last component of the *L_list* representing a coarse layout of the tree $T[u]$ according to Lemma 3.16. Λ_i, $1 \le i \le d$, is the L_list representation of R'_i in Lemma 3.16.

The realizations and the correctness of the first and the last functions above are evident since the corresponding lemmas, Observation 3.17, and the transitivity of $\oplus$.

Now we come to the main part of our algorithm.

{ Compute an optimal layout of T }
function *optimal_layout*(T : *tree*) : *layout*;
  Λ := *compute_coarse_layout*(T);
  return *grind_coarse_layout*(Λ);

{ Compute a coarse layout of T }
function *compute_coarse_layout*(T : *tree*) : *L_list*;
  { Let u be the root of T }
  if (T contains only u) **then** $\Lambda := ((0', 0', (u), (u))$;
                return Λ
              else { Let $u_1, \ldots, u_d$ be the sons of u in T };
                for all u_i **do** Λ_i := *compute_coarse_layout*($T[u_i]$);
                return *combine_coarse_layouts*($u, \Lambda_1, \ldots, \Lambda_d$);

{ Compute the coarse layout Λ of $T[u]$ by combining Λ_i }
function *combine_coarse_layouts*(u : *vertex*; $\Lambda_1, \ldots, \Lambda_d$: *L_list*) : *L_list*;
  { Let m_i be the first integer field of the first record in Λ_i }
  $m_t := \max\limits_{1\le i\le d} \{m_i\}$;
  if $(d > 1)$ **then** $m_s := \max\limits_{1\le i\ne t\le d} \{m_i\}$ **else** $m_s := 0$;
  { Excise $\bar{\Lambda}_t$ from Λ_t according to Lemma 3.15 }
  $(\bar{\Lambda}_t, \Lambda_t)$:= *split_L_list*($\Lambda_t, m_s + 1$);
  { Let n be the mark of u }
  { Excise R_i from Λ_i according to Lemma 3.13 }
  for $i := 1$ **to** d **do** (R_i, R_i') := *split_L_list*(Λ_i, n);
  sort all records of $R_1, \ldots, R_d$ according to their first integer fields;
  R := list containing all sorted records in decreasing order;
  Λ' := *compact*(R);
  { Compute the last component of Λ according to Lemma 3.16 }
  R' := *construct_last_component*($u, R_1', \ldots, R_d'$);
  { Compute the whole coarse layout of $T[u]$ }
  $\Lambda := \bar{\Lambda}_t \& \Lambda' \& R'$;
  return Λ;

Next we show the correctness of our algorithm.

LEMMA 3.18. *Given an input tree T compute_coarse_layout correctly computes an L_list representation of a coarse layout of T.*

Proof. Let u be the root of T, $u_1, \ldots, u_d$ the sons of u and $l = (k_1, k_2, \ldots, k_p^*)$ the label of u in T. The proof is by induction on the height h of T.

For $h = 0$, T consists only of the root u. In this case *compute_coarse_layout*(T) results in $\Lambda := ((0', 0', (u), (u)))$ and the claim trivially holds.

For the induction step let T be of height $h + 1$. By the inductive hypothesis the executions *compute_coarse_layout*($T[u_i]$) correctly compute the *L_list* representations Λ_i of the coarse layouts of $T[u_i]$.

Consider the execution $\Lambda := \mathit{combine\_coarse\_layouts}(u, \Lambda_1, \ldots, \Lambda_d)$. We show that Λ is a correct representation of a coarse layout of T.
The function $\mathit{split\_L\_list}(L_t, m_s+1)$ splits Λ_t into a sublist $\bar{\Lambda}_t$ and a new sublist Λ_t. Let (m_r, n_r, X, Y) be the last record of $\bar{\Lambda}_t$. By construction n_r is the smallest integer $k_{t,j}$ of the label $l_t = (k_{t,1}, k_{t,2}, \ldots, k^*_{t,p_t})$ with $k_{t,j} > \max\{k_{t,j+1}, m_s\} + 1$. By Lemma 3.15 the excised *L_list* $\bar{\Lambda}_t$ can be directly adopted from the *L_list* Λ.
Consider the execution $\mathit{split\_L\_list}(\Lambda_i, n)$, $1 \le i \le d$, where $n = k^*_p$ by construction. It splits Λ_i into sublists R_i and R'_i. Let (m_r, n_r, X, Y) be the last record of R_i.
By construction n_r is the smallest integer of the label l_i which is larger than n. This is obvious if n does not occur in l_i and a direct consequence of Lemma 3.13(ii) otherwise. By Lemma 3.13(i) and the transitivity of $\oplus$ the *L_list* Λ' constructed by the function *compact* can be directly adopted from Λ. Finally by Lemma 3.16 and the transitivity of $\oplus$ the *L_list* R' constructed by $\mathit{construct\_last\_component}(u, R'_1, \ldots, R'_d)$ contains the remaining record of Λ. Hence $\Lambda = \bar{\Lambda}_t \& \Lambda' \& R'$ is a correct *L_list* representation of a coarse layout of T. □

By Lemma 3.18 and the correctness of $\mathit{grind\_L\_list}(\Lambda)$ we conclude the following theorem.

THEOREM 3.19. *The function optimal_layout correctly computes an optimal layout of an input tree.*

Next we show that the time complexity of our algorithm is linear.

LEMMA 3.20. *Let T be a tree rooted at u. The time required by the function compute_coarse_layout is linear in the size of T.*

Proof. Let $u_1, \ldots, u_d$ be the sons of u.
First observe that $\mathit{compute\_coarse\_layout}(T)$ has the same structure as the function $\mathit{compute\_label}(T[u])$ of [7]. Therefore it is sufficient to show that our function *combine_coarse_layouts* requires no more time than the function *combine_labels* of [7], see Lemma 2.7. That means it is sufficient to show that the time required by $\mathit{combine\_coarse\_layouts}(T)$ is $O(d + 2m_s + \sum_{1 \le i \ne s,t}^{d} m_i)$.
Consider the execution $\mathit{combine\_coarse\_layouts}(u, \Lambda_1, \ldots, \Lambda_d)$. The computation of m_t and m_s can be done in $O(d)$. The time required by $(\bar{\Lambda}_t, \Lambda_t) := \mathit{split\_L\_list}(\Lambda_t, m_s + 1)$ clearly is $O(m_s + 1)$. The execution time of $(R_i, R'_i) := \mathit{split\_L\_list}\ (\Lambda_i, n)$ is $O(m_s + 1)$ for $i = t$ and $O(m_i + 1)$ for $1 < i \ne t \le d$. The sorting of all records of $R_1, \ldots, R_d$ according to the integers in their first fields can be done in $O(m_s + 1)$ using bucket sort. To see this observe that these integers are (i) all at most $m_s + 1$ except possibly one (this would occur in the first field of the first record of R_t) since splitting of Λ_t and (ii) all pairwise different since Lemma 3.13(i).
For the same reason $\mathit{compact}(R)$ can be done in $O(m_s + 1)$.
The execution time of $R' := \mathit{construct\_last\_component}(u, R'_1, \ldots, R'_d)$ clearly is $O(d + 2m_s + \sum_{1 \le i \ne s,t}^{d} m_i)$.
Hence the time required by the function *combine_coarse_layouts* is $O(d + 2m_s + \sum_{1 \le i \ne s,t}^{d} m_i)$ and the proof is complete. □

The time needed by *optimal_layout* to compute an optimal layout of T is the time needed by *compute_coarse_layout* plus the time needed by *grind_coarse_layout*. By Lemma 3.20 and Corollary 2.2 we conclude the following theorem.

THEOREM 3.21. *The time complexity of the function optimal_layout is linear in the size of the input tree.*

4. Conclusion. We have presented an algorithm computing optimal linear layouts of trees in linear time. As a consequence the construction of optimal edge search strategies, optimal node search strategies, and optimal interval augmentations of trees can be done in linear time.

We are interested in extending our result to k-trees where k is a constant. In this case it has been shown in [2] that the above constructions are possible in polynomial time. However the exponent of the polynom is too large to make their algorithm practicable.

5. Acknowledgment. I wish to thank J. Ellis for his comments.

REFERENCES

[1] H. Bodlaender, *A linear time algorithm for finding tree-decompositions of small treewidth*, SIAM J. Comput., 25 (1996), pp. 1305–1317.

[2] H. Bodlaender and T. Kloks, *Efficient and constructive algorithms for the pathwidth and treewidth of graphs*, J. Algorithms, 21 (1996), pp. 358–402.

[3] H. Bodlaender, T. Kloks, and D. Kratsch, *Treewidth and pathwidth of permutation graphs*, SIAM J. Disc. Math., 8 (1995), pp. 606–616.

[4] H. Bodlaender and R. Möhring, *The pathwidth and treewidth of cographs*, SIAM J. Disc. Math., 6 (1993), pp. 181–188.

[5] M. Chung, F. Makedon, I. Sudborough, and J. Turner, *Polynomial algorithms for the min-cut linear arrangement problem on degree restricted trees*, SIAM J. Comput., 14 (1985), pp. 158–177.

[6] S. Cook and R. Sethi, *Storage requirements for deterministic polynomial finite recognizable languages*, J. Comput. System Sci., 13 (1976), pp. 25–37.

[7] J. Ellis, I. Sudborough, and J. Turner, *The vertex separation and search number of a graph*, Inform. Comput., 113 (1994), pp. 50–79.

[8] M. Fellows and M. Langston, *Nonconstructive advances in polynomial-time complexity*, Inform. Process. Lett., 26 (1987), pp. 157–162.

[9] ———, *Nonconstructive tools for proving polynomial-time decidability*, J. Assoc. Comput. Mach., 35 (1988), pp. 727–739.

[10] M. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.

[11] J. Gustedt, *On the pathwidth of chordal graphs*, Disc. Appl. Math., 45 (1993), pp. 233–248.

[12] L. Kirousis and C. Papadimitriou, *Interval graphs and searching*, Disc. Math., 55 (1985), pp. 181–184.

[13] ———, *Searching and pebbling*, Theor. Comp. Science, 47 (1986), pp. 205–218.

[14] A. LaPaugh, *Recontamination does not help to search a graph*, J. Assoc. Comput. Mach., 40 (1993), pp. 243–245.

[15] T. Lengauer, *Black-white pebbles and graph separation*, Acta Informat., 16 (1981), pp. 465–475.

[16] F. Makedon and I. Sudborough, *On minimizing width in linear layouts*, Disc. Appl. Math., 23 (1989), pp. 243–265.

[17] N. Megiddo, S. Hakimi, M. Garey, D. Johnson, and C. Papadimitriou, *The complexity of searching a graph*, J. Assoc. Comput. Mach., 35 (1988), pp. 18–44.

[18] F. Monien and I. Sudborough, *Min cut is NP-complete for edge weighted trees*, Theor. Comp. Science, 58 (1988), pp. 209–229.

[19] T. Parsons, *Pursuit-evasion in a graph*, in Theory and Applications of Graphs, Y. Alavi and D. Lick, eds., Springer-Verlag, New York/Berlin, 1976, pp. 426–441.

[20] M. Yannakakis, *A polynomial algorithm for the min-cut linear arrangement of trees*, J. Assoc. Comput. Mach., 32 (1985), pp. 950–988.