3D Straight-line Drawings of k-trees *

Technical Report 2003-473, School of Computing, Queen's University

Emilio Di Giacomo † digiacomo@diei.unipg.it

Giuseppe Liotta† liotta@diei.unipg.it

Henk Meijer‡ henk@cs.queensu.ca

Abstract

This paper studies the problem of computing 3D crossing-free straight-line grid drawings of graphs such that the overall volume is small. We show that every 2-tree (and therefore every series-parallel graph) can be drawn on an integer 3D grid consisting of 15 parallel lines and having linear volume. We extend the study to the problem of drawing general k-trees on a set of parallel grid lines. Lower bounds and upper bounds on the number of such grid lines are presented. The results in this paper extend and improve similar ones already described in the literature.

1 Introduction

The increasing demand of visualization algorithms and software systems to draw and browse large networks, makes it relevant to investigate how much benefit can be obtained from the third dimension in order to represent the overall structure of a huge graph in a limited portion of a virtual 3D environment. This paper is devoted to the fundamental (but still quite open) problem of computing crossing-free straight-line three dimensional grid drawings of graphs such that the overall volume is small.

Cohen, Eades, Lin and Ruskey [4] showed that every graph admits crossing-free 3D drawing on an integer grid of $O(n^3)$ volume, and proved that this is asymptotically optimal. Calamoneri and Sterbini [2] showed that all 2-, 3-, and 4-colourable graphs can be drawn in a 3D grid of $O(n^2)$ volume with $O(n)$ aspect ratio and proved a lower bound of $\Omega(n^{1.5})$ on the volume of such graphs. For r-colourable graphs where r is a constant, Pach, Thiele and Tóth [13] showed a bound of $\theta(n^2)$ on the volume. Garg, Tamassia, and Vocca [12] showed that all 4-colorable graphs (and hence all planar graphs) can be drawn in $O(n^{1.5})$ volume and with $O(1)$ aspect ratio but by using a grid model where the coordinates of the vertices may not be integer. Chrobak, Goodrich, and Tamassia [3] gave an algorithm for constructing 3D convex drawings of triconnected planar graphs with $O(n)$ volume and non-integer coordinates.

Recent papers [6, 8, 9, 10, 11, 14] present drawings on integer grids of size $O(1) \times O(1) \times O(n)$. Felsner et al. [11] initiated the study of restricted integer grids, where all vertices are drawn on a small set of parallel grid lines, called *tracks*. In particular, they focused on the *box* and the 3-*prism*. A box is a grid consisting of four parallel lines, one grid unit apart from each other and a 3-prism uses three non-coplanar parallel lines. It is shown that all outerplanar graphs can be drawn on a 3-prism where the length of the lines is $O(n)$. This result gives the first algorithm to compute a crossing-free straight-line 3D grid drawing with linear volume for a non-trivial family of planar graphs. Moreover it is shown that there exist planar graphs that cannot be drawn on the 3-prism and that even a box does not support all planar graphs.

*Research partially supported by "Progetto ALINWEB: Algoritmica per Internet e per il Web", MIUR Programmi di Ricerca Scientifi ca di Rilevante Interesse Nazionale.

†Dipartimento di Ingegneria Elettronica e dell'Informazione, Università degli Studi di Perugia, Perugia, Italy.

‡Department of Computing and Information Science, Queen's University, Kingston, Ontario, Canada.

Dujmović, Morin, and Wood [8] show that if a graph G admits a drawing Γ on a grid consisting of a constant number of tracks, then G has a linear volume upper bound. This result suggests that the focus of the research should be on minimizing the number of tracks in a restricted integer grid, independent of the length of the tracks themselves. The *track number* of a graph is the minimum number of tracks that is required to compute such a drawing. In the same paper, Dujmović et al. show that the track number of a graph G is at most $pw(G)+1$, where $pw(G)$ is the pathwidth of G. Thus graphs with bounded pathwidth and n vertices have 3D straight-line grid drawings of $O(n)$ volume.

Wood [14] shows that series-parallel graph have constant track number and presents the first algorithm to compute 3D straight-line grid drawings of these graphs in linear volume. He further extends this results to graphs that have bounded tree-partition width (which includes those having bounded treewidth and bounded maximum degree). However, the hidden constants in these results are quite high. For example, the constant for series-parallel graphs is in the order 10^{16}.

Recently, Dujmović and Wood [9, 10] have extended and improved the above results by showing that every graph with bounded treewidth (i.e. every partial k-tree) has constant track number and therefore it admits a 3D straight-line grid drawing of linear volume. Motivated by the relevance of series-parallel graphs for graph drawing applications, Dujmović and Wood further investigate the track number and volume bounds of 2-trees (every 2-tree is a series-parallel graph and every series-parallel graph can be augmented to become a 2-tree). They show that the track number of a series-parallel graph is at most 18 and that a series-parallel graph has a 3D straight-line grid drawings of volume at most $36 \times 37 \times 37\lceil \frac{n}{18} \rceil$. For general k-trees (i.e. $k \geq 3$) however, the hidden constants are quite high and no lower bounds are presented. For series-parallel graphs, a lower bound of 5 on the track number is shown in [6].

In this paper we present new results on the track number of k-trees. Our contribution can be listed as follows.

- We present lower bounds on the track number of k-trees. For any given value of k we show a k-tree that requires at least $2k+1$ tracks. This result generalizes the lower bound on 2-trees showed in [6].
- The upper bound on the track number of 2-trees (and therefore of series-parallel graphs) is reduced from 18 to 15. As a consequence, the volume upper bound for series-parallel graphs is reduced by approximately thirty percent compared to that of [9, 10].
- By applying similar ideas as in [9, 10] we extend the drawing technique for 2-trees to general k-trees ($k \geq 3$). This gives rise to new upper bounds on the track number of k-trees. The new upper bounds are lower than in [9, 10], but still doubly exponential.

The remainder of this paper is organized as follows. Preliminary definitions can be found in Section 2. The lower bounds on the volume for drawing of k-trees is given in section 3. The drawing algorithm for 2-trees is presented in Section 4. The upper bounds for k-trees are given in Section 5. Some open problems are listed in Section 6. For reasons of space some details are omitted from this extended abstract and can be found in the Appendix.

2 Preliminaries

We assume familiarity with basic graph drawing terminology [5] and only recall those definitions about track assignment and drawings and about k-trees [1] that will be used throughout the paper.

2.1 Track layouts and drawings

Let $G = (V, E)$ be a graph. A *track assignment* of G consists of a partition $\{t_i \mid i \in I \subseteq \mathbb{N}\}$ of V, and of a total ordering $<_i$ of the vertices in each set t_i. Each set t_i is called a *track*. An *overlap* in a track assignment

consists of three vertices u, v, and w such that they are in the same track t_i, there exists the edge (u,w) and $u <_i v <_i w$. An *X-crossing* in a track assignment consists of two edges (u,w) and (v,z) such that u and v are in a same track t_i, w and z are in another track t_j ($i \neq j$), and $u <_i v$ and $z <_j w$. Figure 1(b) shows an example of track assignment for the graph in Figure 1(a). Vertices v_1, v_5 and v_2 form an overlap, as well as vertices v_3, v_6 and v_4. Edges (v_5,v_4), (v_2,v_3) form an X-crossing. Another X-crossing is formed by edges (v_6,v_8) and (v_4,v_7).

A *track layout* is a track assignment with no overlaps and no X-crossings. A track layout with k tracks is also called a *k-track layout*. Figure 1(c) shows a 3-track layout of the graph of Figure 1(a). The *track number* of a graph G, denoted by $tn(G)$, is the minimum k such that G has a k-track layout. A set of k tracks is also called a *k-prism*. In [9, 10] a track assignment is called a track layout if in addition there is no edge (u,v) such that u and v are on the same track. Since our upperbounds in Sections 4 and 5 do not use edges that lie on a track, our upperbounds are directly comparable to those in [9, 10].

In the rest of the paper a track layout will be specified by assigning to each vertex v two numbers: $track(v)$ is an integer that denotes the track to which v is assigned; $order(v)$ is an integer that denotes the ordering of v on $track(v)$. We say that $u <_i v$ if $track(u) = track(v) = i$ and $order(u) < order(v)$. We shall sometimes simplify the notation and write $u < v$ instead of $u <_i v$. Also we write $u \leq v$ to mean that either $u < v$ or u coincides with v.

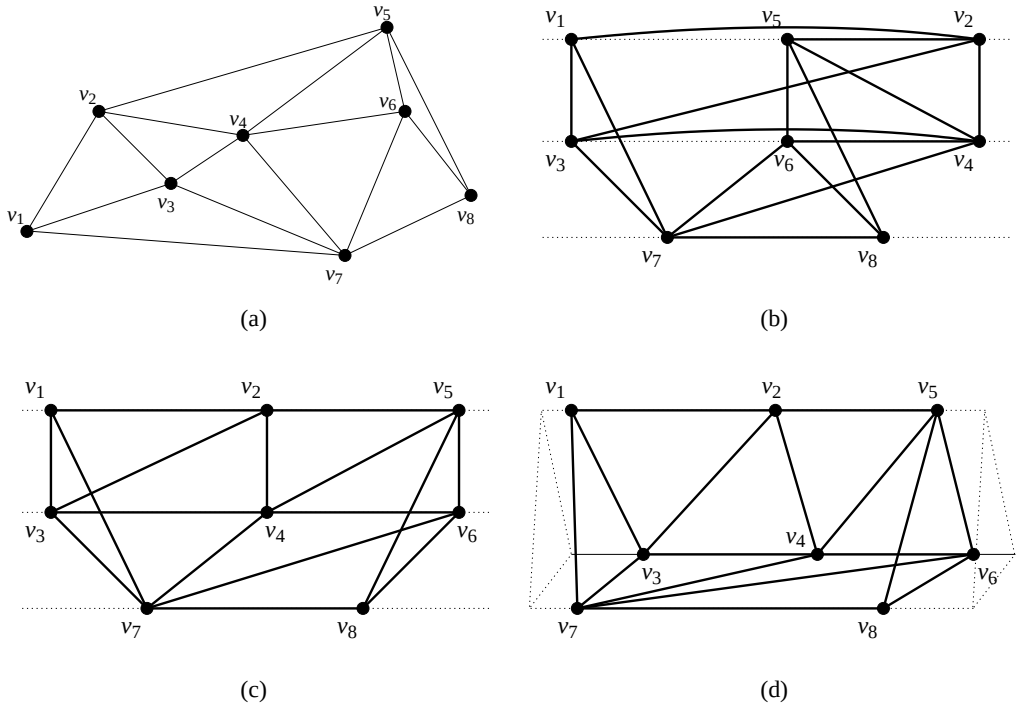


Figure 1: (a) A graph G. (b) A track assignment of G. (c) A track layout of G. (d) A track drawing of G.

A *track line* is a straight line of a 3D grid parallel to the x-axis. A *track drawing* of a graph G on k track lines is a 3D straight-line crossing-free grid drawing of G such that each vertex of G is drawn as a point at integer coordinates on one of the k track lines. A track drawing on k track lines is also called *k-track drawing*. The drawing in Figure 1(d) is a 3-track drawing of the graph of Figure 1(a).

In [8] the relationships between track layouts, track drawings, and their volume upper bounds are studied. The following theorem holds whether edges are allowed to lie on tracks or not.

Theorem 1 *[8] Let G be a graph with n vertices such that $tn(G) = t$. Then:*

- *G admits a t-track drawing whose volume is $t \times p_t \times p_t \cdot n'$;*
- *G admits a 2t-track drawing whose volume is $2t \times p_{2t} \times p_{2t} \cdot \lceil \frac{n}{t} \rceil$;*

where p_t is the smallest prime number greater than t, p_{2t} is the smallest prime number greater than 2t and n' is the maximum number of vertices on a single track.

We recall that the volume of a drawing Γ is measured as the number of grid points contained in or on a *bounding box* of Γ, *i.e.* the smallest axis-aligned box enclosing Γ.

2.2 k-trees

A *k-tree* for some $k \in \mathbb{N}$ is defined recursively as follows. The clique of size k is a k-tree, and the graph obtained from a k-tree by adding a new vertex adjacent to each vertex of a clique of size k is also a k-tree. The maximum size of a clique in a k-tree is $k+1$. A clique of size k is also called a *k-clique*. A *partial k-tree* is a subgraph of a k-tree. Figure 2(a) shows a 2-tree. Note that a 1-tree is a tree.

Let $G = (V(G), E(G))$ be a graph and let $T = (V(T), E(T))$ be a rooted tree. Let $\{T_\mu \subseteq V(G) \mid \mu \in V(T)\}$ be a set of subsets of $V(G)$ indexed by the nodes of T. The pair $(T, \{T_\mu \mid \mu \in V(T)\})$ is a *tree partition* of G if [7]:

- $\forall \mu, \nu \in V(T)$, if $\mu \neq \nu$ then $T_\mu \cap T_\nu = \emptyset$;
- $\forall (u,v) \in E(G)$, either
  - $\exists$ a node $\mu \in V(T)$ with $u, v \in T_\mu$
  - $\exists$ an edge $(\mu, \nu) \in E(T)$ such that $u \in T_\mu$ and $v \in T_\nu$.

Let μ be an element of $V(T)$ in a tree partition of G. The *pertinent graph* of μ is the subgraph of G induced by the vertices in T_μ; the pertinent graph of μ is denoted as G_μ. Figure 2 (b) shows a tree partition of the 2-tree in Figure 2 (a). Each node of T is represented as a shaded area and contains its pertinent graph. Let $e = (u,v) \in E(G)$ be such that there exists an edge $(\mu, \nu) \in E(T)$ with $u \in T_\mu$ and $v \in T_\nu$. Edge e is called *jumping edge*. For example, edge $e = (u,v)$ in the graph of Figure 2 (a) is a jumping edge; the corresponding nodes μ and ν of T are highlighted in Figure 2 (b). If μ is the parent of ν in T, then u is called *parent vertex* of e and v is the *child vertex* of e.

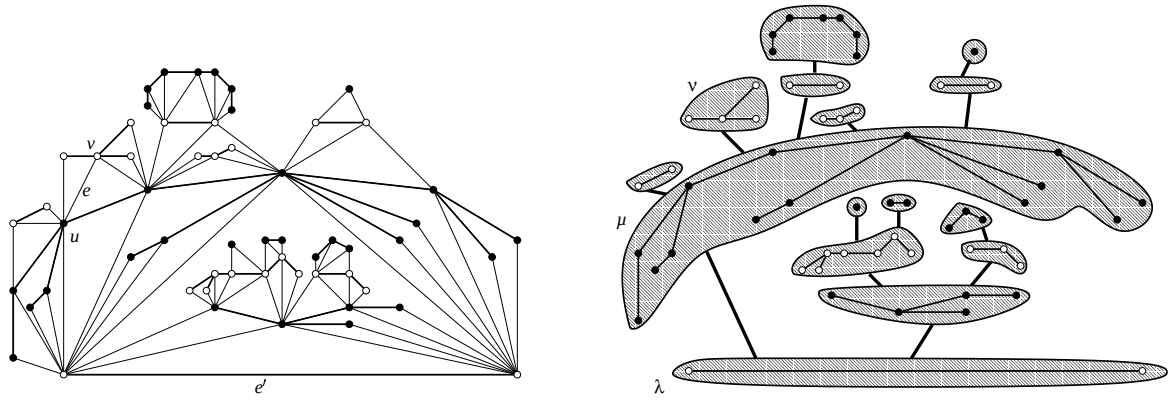
In [9, 10] it is shown that if G is a k-tree, then G admits a tree partition $(T, \{T_\mu \mid \mu \in V(T)\})$ such that for each node μ of T:

- the pertinent graph G_μ is a connected partial $(k-1)$-tree; and
- if μ is a non-root node and λ is the parent of μ, then the set of vertices in T_λ with a neighbour in T_μ form a clique of G, that will be denoted as C_μ.

Clique C_μ is called *parent clique* of μ and has size k. For example, if G is a 2-tree, each parent clique is an edge. Referring to Figure 2 (b), we have that the pertinent graph of node μ is a 1-tree. In the figure, λ is the parent of μ and the parent clique C_μ of μ is a clique of size 2, namely edge e'.

In the remainder of this paper, we assume that a tree partition $(T, \{T_\mu \mid \mu \in V(T)\})$ always has the properties above. We will use T rather than $(T, \{T_\mu \mid \mu \in V(T)\})$ to denote the tree partition.

Let G be a k-tree, an *equipped tree partition* T of G is a tree partition such that each node μ is equipped with a track layout of its pertinent graph G_μ.



(a) (b)

Figure 2: (a) A 2-tree and (b) its tree partition.

3 Lower bound for the track number of k-trees

A lower bound on the track number of 2-trees is presented in [6], where it is shown that there exist series-parallel graphs that do not admit a 4-track drawing (we recall that every series-parallel graph is a partial 2-tree [1]). In this section we show that the track number of a k-tree is at least $2k+1$.

A trivial lower bound on the track number of a k-tree is k. Indeed, a k-tree can contain a $(k+1)$-clique and in any track layout of a clique at most one track can have two vertices and no track can have more than two. We now prove a $(2k+1)$ lower bound on the track number of k-trees.

Definition 1 *Let $\Gamma(G)$ be a track layout of a graph G and let C be a k-clique of G. We say that C* covers *a subset Θ of k tracks in $\Gamma(G)$ if C has one vertex in each track of Θ.*

In other words, if a clique C covers a set of tracks Θ, for any two tracks of Θ there is a pair of vertices of C connected by an edge.

Definition 2 *Let $\Gamma(G)$ be a track layout of a graph G and let H be a subgraph of G. H* covers *a subset Θ of the tracks in $\Gamma(G)$, if H contains a clique C that covers Θ. We say that C is the* covering clique *of H.*

Definition 3 *Let $\Gamma(G)$ be a track layout of a graph G and let H_0 and H_1 be two subgraphs of G that cover the same tracks. H_0 is said to be to the left of H_1 if for each pair of vertex $v \in H_0$ and $w \in H_1$ that are in the same track t, $v <_t w$ or $v = w$. If H_0 is to the left of H_1 we write $H_0 \leq H_1$. We say $H_0 < H_1$ if $H_0 \leq H_1$ and $H_0 \neq H_1$.*

Notice that if C_0 and C_1 are two vertex-disjoint cliques that cover the same tracks then either $C_0 < C_1$ or $C_1 < C_0$.

Lemma 2 *Let G be a graph that contains three vertex-disjoint cliques C_0, C_1 and C_2. Let $\Gamma(G)$ be a track layout of G such that C_0, C_1 and C_2 each cover the same set of tracks Θ. Let $c_0 \in C_0$, $c_1 \in C_1$ and $c_2 \in C_2$ be three vertices that are in a same track. Let v be a vertex of G not in C_0, C_1 or C_2 that is adjacent to c_0, c_1 and c_2. Then v belongs to a track not in Θ.*

Sketch of Proof. Suppose as a contradiction that v belongs to a track t_0 of Θ. Let t_1 be the track of Θ containing c_0, c_1 and c_2. If $t_0 = t_1$ then two of the three edges (c_0, v), (c_1, v), and (c_2, v) would overlap. Assume $t_0 \neq t_1$. Since C_0, C_1 and C_2 cover Θ there exist three edges $e_0 = (c_0, w_0)$, $e_1 = (c_1, w_1)$ and $e_2 = (c_2, w_2)$ such that w_0, w_1 and w_2 are in t_0. Since C_0, C_1 and C_2 are vertex-disjoint we may assume, without loss of generality, that $C_0 < C_1 < C_2$. Then $c_0 < c_1 < c_2$ and $w_0 < w_1 < w_2$. If $v < w_0$ then edges

(v,c_1) and (c_0,w_0) would form an X-crossing. If $w_0 < v < w_1$ then edges (v,c_2) and (c_1,w_1) would form an X-crossing. If $w_1 < v < w_2$ then edges (v,c_0) and (c_1,w_1) would form an X-crossing. Finally, if $v > w_2$ then edges (v,c_1) and (c_2,w_2) would form an X-crossing. It follows that v cannot be in one of the tracks of Θ. ■

Let $\widetilde{G}$ be a k-tree consisting of a k-clique C, and of $2k$ vertices each adjacent to all vertices of C. Graph $\widetilde{G}$ is called *base k-tree*; the vertices of clique C of $\widetilde{G}$ are called *white vertices* and are denoted as c_0, ..., c_{k-1}; the other vertices of $\widetilde{G}$ are called *black vertices* and are denoted as v_0, ..., v_{2k-1}. Figure 3 shows an example of a base 3-tree with white and black vertices highlighted.

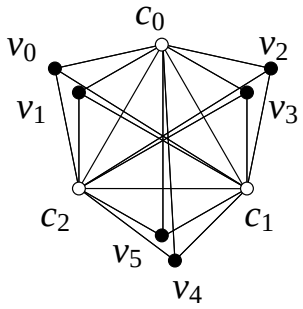


Figure 3: A base 3-tree.

Lemma 3 *Let $\widetilde{G}$ be a base k-tree. Then $tn(\widetilde{G}) > k$.*

Proof. The graph induced by the white vertices c_0, ..., c_{k-1} plus a black vertex v_i forms a $(k+1)$-clique. As already observed, any track layout of a $(k+1)$-clique requires at least k tracks. It follows that $tn(\widetilde{G}) \geq k$. We now prove that $tn(\widetilde{G}) > k$. Since the white vertices form a k-clique then in any track layout of $\widetilde{G}$ there are at least $k-1$ tracks that contain a white vertex. Let Θ be the set of tracks that contain at least one white vertex. The following facts hold for Θ.

Fact 1 For each track $t \in \Theta$, t can contain at most two black vertices. If not the track assignment would contain an overlap.

Fact 2 At most one track of Θ can contain two black vertices. Let t_0 and t_1 be two tracks of Θ and let c_0 and c_1 be the two white vertices in t_0 and t_1, respectively. Assume that t_0 contains two black vertices v_i and v_j and t_1 contains two black vertices v_h and v_l. In order to avoid overlaps in t_0, c_0 must be between v_i and v_j; assume $v_i < c_0 < v_j$. Analogously c_1 must be between v_h and v_l; assume $v_h < c_1 < v_l$. Then edges (c_0,v_h) and (c_1,v_i) and also edges (c_0,v_l) and (c_1,v_j) form X-crossings, a contradiction.

Assume for a contradiction that $tn(\widetilde{G}) = k$. Two cases are possible:

1. Each white vertex is in a distinct track. By *Fact 1* each track in Θ contains at most two black vertices, and since there are $2k$ black vertices, then we have exactly two vertices in each track-set, which contradicts *Fact 2*.
2. There exists one track t_0 containing two white vertices c_0 and c_1. Let t_1 be the track that does not contain white vertices. By *Fact 1* t_1 contains at least two black vertices v_i and v_j. Assume without loss of generality that $c_0 < c_1$ in t_0 and that $v_i < v_j$ in t_1 Then edges (c_0,v_j) and (c_1,v_i) form an X-crossings.

It follows that $tn(\widetilde{G}) > k$ ■

Lemma 4 *Let $\widetilde{G}$ be a base k-tree. In any τ-track layout such that $k+1 \leq \tau \leq 2k$, $\widetilde{G}$ covers $k+1$ track-sets and a covering clique of $\widetilde{G}$ is induced by all white vertices plus one black vertex.*

Sketch of Proof. Let $\Gamma(\widetilde{G})$ be a τ-track layout of $\widetilde{G}$ ($k+1 \leq \tau \leq 2k$) and let t be a track of $\Gamma(\widetilde{G})$. Suppose that two white vertices c_0 and c_1 belong to t. In this case none of the black vertices can belong to t or else there would be a three-cycle defined by vertices in the same track and hence an overlap. It follows that all black vertices are in tracks different from t. Since there are $2k$ black vertices and there are at most $2k-1$ tracks different from t, at least two black vertices, say v_i and v_j ($0 \leq i, j \leq 2k-1$), belong to a same track. Assume without loss of generality that $c_0 < c_1$ and that $v_i < v_j$. Edges (c_0, v_j) and (c_1, v_i) form an X-crossing. It follows that c_0 and c_1 cannot both belong to t and thus the clique formed by the white vertices covers a set Θ of k tracks.

We prove now that at least one black vertex must belong to a track not in Θ. Suppose that the k tracks of Θ contain all black vertices. Since there are $2k$ black vertices, two cases are possible.

1. There exist three black vertices v_i, v_j and v_h, in the same track t. Let c be the white vertex in track t. Two of the three edges (c, v_i), (c, v_j), and (c, v_h) overlap, which is impossible.
2. There exist four black vertices such that two of them are in a track and the other two are in another track. Let v_i, v_j, v_h, and v_l be such black vertices and assume that v_i and v_j are in a track t_0 and that v_h and v_l are in a track t_1 ($t_0 \neq t_1$). Let c_0 and c_1 be the white vertices of C that are in tracks t_0 and t_1, respectively. In order to avoid overlaps in t_0, c_0 must be between v_i and v_j; assume $v_i < c_0 < v_j$. Analogously c_1 must be between v_h and v_l; assume $v_h < c_1 < v_l$. But then edges (c_0, v_h) and (c_1, v_i) and also edges (c_0, v_l) and (c_1, v_j) form an X-crossing, which is impossible.

It follows that at least one black vertex v_i is in a track not in Θ. Since each black vertex is adjacent to all white vertices, it follows that $\widetilde{G}$ covers $k+1$ tracks and the subgraph induced by the white vertices plus v_i is a covering clique. ■

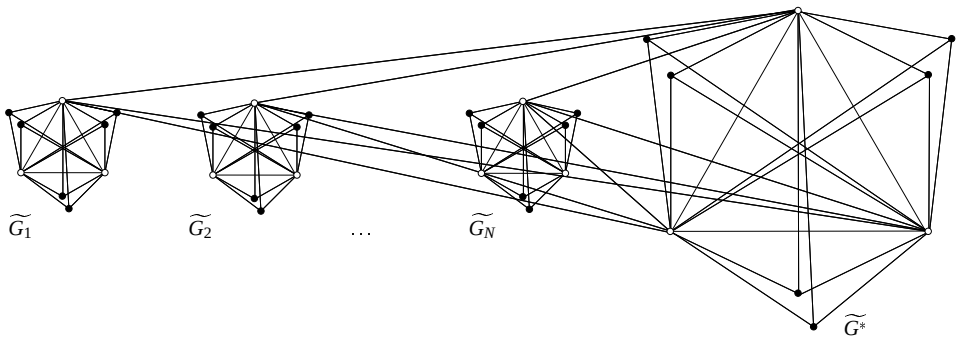


Figure 4: The graph G of Theorem 5 for $k = 3$.

Theorem 5 *There exists a k-tree G such that* $tn(G) \geq 2k+1$.

Sketch of Proof. W e first construct a particular partial k-tree G and then show that $tn(G) \geq 2k+1$.

Graph G is constructed as follows. Let $\widetilde{G_1}, \widetilde{G_2}, \ldots, \widetilde{G_N}$ be N copies of a base k-tree such that $N = 2(k+1)\binom{2k}{k+1} + 1$. We call these copies *small graphs*. Let $\widetilde{G^*}$ be another base k-tree, called the *big graph*. For each small graph $\widetilde{G_i}$ ($i = 1, \ldots, N$) let c_i be a distinguished white vertex , called *pivot vertex* of $\widetilde{G_i}$. For each $\widetilde{G_i}$, we connect its pivot vertex c_i to all white vertices of the big graph $\widetilde{G^*}$. Figure 4 shows the construction of G for $k = 3$. It is not hard to see that we can ad edges to G such that it becomes a k-tree.

Since a base k-tree is a subgraph of G, by Lemma 3 $tn(G) > k$. Assume that $k+1 \leq tn(G) \leq 2k$ and consider a τ-track layout of G ($k+1 \leq \tau \leq 2k$). By Lemma 4 each small graph covers $k+1$ tracks. Since there are N small graphs in G, at least $2(k+1)+1$ of them cover the same set Θ of $k+1$ tracks and at least three of them have their pivot vertex in the same track t. Let G_i, G_j and G_k be such small graphs and let

c_i, c_j and c_k be the three pivot vertices in track t. Since c_i, c_j and c_k are white, by Lemma 4 they are in the covering cliques of G_i, G_j, and G_k, respectively. Let C_i, C_j and C_k be such covering cliques and let v be a white vertex of the big graph $\widetilde{G^*}$. Since v is adjacent to c_i, c_j and c_k and since the covering cliques C_i, C_j, and C_k are vertex-disjoint, we can apply Lemma 2 and conclude that v is not in a track of Θ. Also, by Lemma 4 no two white vertices of $\widetilde{G^*}$ are in the same track. It follows that any track layout of G requires at least $k+1$ tracks for the small graphs and k more tracks for the white vertices of the big graph and therefore $tn(G) \geq 2k+1$, contradicting the assumption that $tn(G) = 2k$. ■

4 Upperbound on the track number of 2-trees

Dujmović and Wood [9, 10] prove that all 2-trees have a drawing on 18 tracks. They achieve this by drawing the trees G_μ on 6 different sets of 3 lines each. In this section we show that by drawing the trees G_μ in a particular order, we can draw them on 5 sets of 3 lines each, giving a drawing of 15 track lines.

Let G be a 2-tree and let T be an equipped tree partition of G. Let μ be a node of T and let G_μ be the pertinent graph of μ. By the definition of tree partition, we have that G_μ is a tree (see for example Figure 2). We arbitrarily root G_μ at a vertex and colour the edges with two colours as follows. All edges incident to the root are coloured black. All remaining edges are coloured black or white in such a way that any path from the root to a leaf of G_μ consists of alternating black and white edges.

Since T is an equipped tree partition, each node μ of T is associated with a track layout. We assume that the track layout of G_μ is computed by using the algorithm presented in [11] that is based on the idea of "wrapping" a tree around a 3-prism. Shortly speaking, the wrapping idea is as follows. Perform a BFS visit of the tree starting from the root and for each visited vertex v set $track(v) = (d \mod 3)$ and set $order(v) = (n_v + 1)$ where d is the distance from the root and n_v is the number of vertices visited before v. Note that the computed track layout of G_μ is such that: (i) $0 \leq track(v) < 3$ for each vertex v of G; (ii) $0 \leq order(v) \leq n-1$; and (iii) no edge has both vertices assigned to the same track. Let $e_0 = (u_0, v_0)$ and $e_1 = (u_1, v_1)$ be two edges of G_μ. We say that e_0 is *to the left of* e_1 (and that e_1 is *to the right of* e_0) if either the distance of e_0 from the root is less than the distance of e_1 from the root; or e_0 and e_1 have the same distance from the root, and $u_0 \leq u_1$, $v_0 \leq v_1$ and $e_0 \neq e_1$.

Let N_T be the number of nodes of T. We now define a total ordering for the nodes of T such that each node μ of T is given a number, denoted as $visitorder(\mu)$, in the range $[0, N_T - 1]$. This is achieved by performing a particular version of a breadth first search of T, in which the children of a node μ are grouped according to the colour of their parent clique and within each group they are sorted according to the left-to-right ordering of their parent cliques. A pseudo-code description of such an ordering procedure is given in Algorithm 2TVISITORDER() that can be found in the Appendix.

We now present our algorithm to compute a track layout of a 2-tree G. Similar to the approach of Dujmović and Wood [9, 10], the algorithm receives as input G and an equipped tree partition T of G. We assign the 3-track layout of each G_μ in T to one of five different 3-prisms chosen according to the order defined by Algorithm 2TVISITORDER(). It follows that the total number of tracks that are used is 15. A detailed description of our strategy is given in Algorithm 2TTRACKLAYOUT(). In the pseudo-code, P_0, P_1, P_2, P_3, and P_4 denote five 3-prisms; the tracks of each 3-prism P_j ($j = 0, \ldots, 4$) are numbered $3j$, $3j+1$ and $3j+2$.

Since each vertex v of G is given a distinct pair $(track(v), order(v))$, we have that Algorithm 2TTRACK-LAYOUT() defines a track assignment. The next lemmas prove that such an assignment is a track layout.

Lemma 6 *Let G be a 2-tree and let T be an equipped tree partition of G. Algorithm* 2TTRACKLAYOUT*() computes a track assignment without overlaps.*

Sketch of Proof. An edge e cannot have its two vertices on the same track. Indeed, if e is a jumping edge its

2TTRACKLAYOUT(G,T)
Input: A 2-tree G and an equipped tree partition T of G.
Output: A track layout of G on 15 tracks.
foreach vertex μ of T
compute $visitorder(\mu)$;
endfor
Let ρ be the root of T;
$\Delta \leftarrow |G_\rho|$
for $i = 1$ **to** $N_T - 1$
Let μ be the node of T such that $visitorder(\mu) = i$;
Let λ be the parent of μ;
Let P_j be the prism whose tracks contains the vertices of G_λ;
if the colour of C_μ is black
$h \leftarrow 1$;
else
$h \leftarrow 2$;
endif
$p \leftarrow (j+h) \mod 5$;
foreach vertex v of G_μ
$track(v) \leftarrow track(v) + 3p$;
$order(v) \leftarrow order(v) + \Delta$;
endfor
$\Delta \leftarrow \Delta + |G_\mu|$

Algorithm 1: Algorithm 2TTRACKLAYOUT()

vertices are on different tracks because they are on different prisms. If e is not a jumping edge it must belong to a pertinent graph G_μ of a node μ of T and its vertices are on different tracks because the track layout of G_μ is computed with the wrapping technique of [11]. It follows that the track assignment computed by Algorithm 2TTRACKLAYOUT() cannot have overlaps. ■

Lemma 7 *Let G be a 2-tree, let $\Gamma(G)$ be a track assignment of G computed by Algorithm* 2TTRACKLAYOUT*(). Let $e_0 = (u_0, v_0)$ be a jumping edge such that u_0 is its parent vertex. Let $e_1 = (u_1, v_1)$ be a jumping edge such that u_1 is on the same track as u_0 and v_1 is on the same track as v_0. Then u_1 is the parent vertex of e_1.*

Sketch of Proof. Let P_i be the 3-prism that contains u_0 and u_1. Let P_j be the 3-prism that contains v_0 and v_1. Since u_0 and v_0 are in pertinent graphs G_{μ_0} and G_{μ_1} respectively, such that μ_1 is a child of μ_0 in T, it follows that $j = (i+1) \mod 5$ or $j = (i+2) \mod 5$. Suppose as a contradiction that u_1 is the child vertex of e_2. By the same argument we get that $i = (j+1) \mod 5$ or $i = (j+2) \mod 5$, either one of which is impossible. ■

Lemma 8 *Let G be a 2-tree and let T be an equipped tree partition of G. Algorithm* 2TTRACKLAYOUT*() computes a track assignments without X-crossings.*

Sketch of Proof. Let $e_0 = (u_0, v_0)$ and $e_1 = (u_1, v_1)$ be two edges of G. If e_0 and e_1 form an X-crossing in the track assignment, then the two vertices of e_0 are on the same two tracks as the two vertices of e_1. Therefore both edges must be either jumping or non-jumping.

Consider first the case that they are both non-jumping edges; e_0 and e_1 can either belong to the same

pertinent graph G_μ or they are edges of two different pertinent graphs G_μ and G_ν. If they formed an X-crossing in the first case then there would be an X-crossing in the track layout of G_μ equipping T, which is impossible. If there were an X-crossing in the second case, then either $u_0 < u_1$ and $v_1 < v_0$ or $u_1 < u_0$ and $v_0 < v_1$. But if $\mathit{visitorder}(\mu) < \mathit{visitorder}(\nu)$ then e_0 is to the left of e_1 or e_1 is to the left of e_0 by construction. Again a contradiction.

It remains to consider the case that e_0 and e_1 are two jumping edges. We assume without loss of generality that u_0 is the parent vertex of e_0 and that u_0 and u_1 are on the same track. By Lemma 7, u_1 is a parent vertex of e_1. Let μ_0, μ_1, ν_0 and ν_1 be the nodes whose pertinent graphs contain u_0, u_1, v_0 and v_1, respectively. It follows that u_0 is a vertex in the parent clique C_{ν_0} of ν_0 and that u_1 is a vertex in the parent clique C_{ν_1} of ν_1.

If $\mu_0 = \mu_1$ and $C_{\nu_0} = C_{\nu_1}$ then $u_0 = u_1$ because each clique has only one vertex on a single track. In this case a crossing is not possible because e_0 and e_1 have a vertex in common. If $\mu_0 = \mu_1$ and C_{ν_0} is to the left of C_{ν_1} then $u_0 < u_1$ and $v_0 < v_1$. Therefore an X-crossing is not possible. If $\mu_0 \neq \mu_1$, assume without loss of generality that $\mathit{visitorder}(\mu_0) < \mathit{visitorder}(\mu_1)$. The ordering of the nodes of T is such that $\mathit{visitorder}(\nu_0) < \mathit{visitorder}(\nu_1)$. This implies that $u_0 < u_1$ and $v_0 < v_1$. Hence an X-crossing is impossible. ■

Lemmas 6 and 8 imply that Algorithm 2TTRACKLAYOUT() computes a track layout of a 2-tree. Since the algorithm uses five 3-prisms, the total number of tracks used is 15. The following theorem summarizes this discussion and uses Theorem 1 for the volume upper bound.

Theorem 9 *Let G be a 2-tree. Then $tn(G) \leq 15$. Also, G admits a track drawing with volume at most $30 \times 31 \times 31\lceil \frac{n}{15} \rceil$.*

Theorem 9 immediately extends to partial 2-trees.

Corollary 10 *Let G be a series-parallel graph. There exists an algorithms that computes a 3D straight-line grid drawing of G with volume at most $30 \times 31 \times 31\lceil \frac{n}{15} \rceil$.*

We observe that the multiplicative constant factor in the volume upper bound of Theorem 9 and of Corollary 10 improves that in [9, 10] by approximately thirty percent.

5 Upperbound on the track number of k-trees

The algorithm for drawing k-trees in [9, 10] is recursive. Since we can now draw 2-trees on 15 lines, we can apply the same ideas to find new upper bounds on the track number of k-trees.

Theorem 11 *[9, 10] Every k-tree has track number at most t_k given by the following recursive equation:*

$$\begin{array}{ll} t_k = 3c_{k-1,k}t_{k-1} & (k \in \mathbb{N}) \\ c_{k,i} = c'_{k,i} + c''_{k,i} & (k \in \mathbb{N}, 1 \leq i \leq k+1) \\ c'_{k,i} = 3c_{k-1,k}c_{k-1,i} & (k \in \mathbb{N}, 1 \leq i \leq k) \\ c''_{k,i} = 3c^2_{k-1,k}\sum_{j=1}^{i-1} c_{k-1,j}c_{k-1,i-j} & (k \in \mathbb{N}, 1 \leq i \leq k+1) \end{array} \tag{1}$$

where $t_0 = 1$, $c_{0,1} = 1$, and $c'_{k,k+1} = 0$.

By using the same technique as we used in Section 4 we can find a new recursive equation that improves the upper bounds of the above theorem. For reasons of space, we only sketch our approach. A detailed description can be found in the Appendix. We start by introducing some definitions.

Definition 4 *Let C_0 and C_1 be two cliques of G and let $\Gamma(G)$ be a track-layout of G. C_0 and C_1 are of the same type if they cover the same subset of tracks in $\Gamma(G)$.*

Definition 5 *Let G be a k-tree, let T be an equipped tree partition of G and let μ be a node of T.*

- *The* t-number of μ *is the number of tracks used in the track layout of G_μ and is denoted as t_μ.*
- *The* c-number of μ *is the number of types of cliques of size k in the track layout of G_μ, and is denoted as c_μ.*
- *The* t-number of T *is equal to* $\max_{\mu \in V(T)} t_\mu$*, and is denoted as t_T.*
- *The* c-number of T *is equal to* $\max_{\mu \in V(T)} c_\mu$*, and is denoted as c_T.*

Similar to [9, 10], we use a recursive technique based on an equipped tree partition of a k-tree G. The pertinent graph G_μ of any node μ of the tree-partition T is a partial $(k-1)$-tree. G_μ is augmented to a k-tree and a track layout of G_μ with at most t_T tracks is recursively computed. The maximum number of types of cliques in the track layout of the pertinent graph G_μ of any node μ is c_T. For each type of clique τ in G_μ the children of μ whose parent clique is of type τ have a track layout that uses a different t_T-prism.

The difference between our technique and the one in [9, 10] is on how these different t_T-prisms are chosen. In [9, 10] three groups of c_T different t_T-prisms are considered and the track layouts of the pertinent graphs of the nodes that have the same depth in T use the same group of t_T-prisms. The track layouts of the pertinent graphs of two nodes of T having different depth use the same group if their depths are congruent modulo 3. The total number of t_T-prisms that is needed is $3c_T$ and the total number of tracks is $3c_T t_T$.

We generalize the approach of Section 4 and use only $2c_T + 1$ different t_T-prisms. Firstly, the nodes of T are ordered by performing a particular version of a breadth first search, in which the children of a node μ are grouped according to the type of their parent clique and within each group they are sorted according to the left-to-right ordering of their parent cliques. The algorithm that computes a track layout of G visits the nodes of T according to their order. Let μ be the currently visited node of T and let G_μ be its pertinent graph. Let P_i be the t_T-prism used for the track layout of G_μ. The algorithm maintains the following invariants. The track layout of the pertinent graph of the parent of μ uses one of the c_T t_T-prisms P_j such that $j = (i-k) \mod (2c_T+1)$, $(1 \leq k \leq c_{k-1})$. The track layouts of the pertinent graphs of the children of μ use one of the c_T t_T-prisms P_j such that $j = (i+k) \mod (2c_T+1)$, $(1 \leq k \leq c_{k-1})$; for each child ν of μ, the choice of the t_T-prism to use for the track layout of G_ν depends on the type of its parent clique in G_μ.

Since $2c_T + 1$ is less than $3c_T$ for any value of c_T greater than 1, the number of tracks used in our technique is smaller than the one given by Theorem 11. The following result can be proved by extending the techniques presented in Section 4 (see Appendix for details).

Theorem 12 *Every k-tree G has track number at most t_k, where t_k is given by the following recursive equation:*

$$\begin{array}{ll} t_k = (2c_{k-1,k}+1)t_{k-1} & \\ c_{k,i} = c'_{k,i} + c''_{k,i} & (1 \leq i \leq k+1) \\ c'_{k,i} = (2c_{k-1,k}+1)c_{k-1,i} & (1 \leq i \leq k) \\ c''_{k,i} = (2c_{k-1,k}+1)c_{k-1,k}\sum_{j=1}^{i-1} c_{k-1,j}c_{k-1,i-j} & (1 \leq i \leq k+1) \end{array} \tag{2}$$

with $t_2 = 15$, $c_{2,1} = 15$, $c_{2,2} = 105$, $c_{2,3} = 180$ and $c'_{k,k+1} = 0$.

Furthermore, G admits a track drawing with volume at most $2t_k \times p \times p\lceil \frac{n}{t_k} \rceil$, where p is the smallest prime number greater than $2t_k$.

Table 1 compares Equation 2 of Theorem 12 with Equation 1 of Theorem 11, for values of k such that $2 \leq k \leq 6$. For example, when $k = 4$ the value of Equation 2 is about 4 times smaller than the one of Equation 1; for $k = 5$, the ratio becomes order of 10^3; for $k = 6$ the ratio becomes order of 10^{11}.

k	Eq.2	Eq.1
2	15	18
3	5415	7776
4	1.16e+13	5.15e+13
5	3.17e+47	1.96e+50
6	4.68e+175	7.73e+186

Table 1: A comparison between the track number given by Theorem 12 and Theorem 11.

6 Open Problems

Several interesting problems about 3D straight-line drawings of graphs and in particular about track drawings of k-trees remain open. We mention here three of them that naturally raise from the results in this paper and that can stimulate future research.

- Reduce the gap between lower bound of five and upper bound of fifteen on the track number of 2-trees.
- Improve the volume bounds for 3D straight-lines drawings of k-trees, especially for $k \geq 3$.
- Investigate the track number of other families of graphs.

References

[1] H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1-2):1–45, 1998.

[2] T. Calamoneri and A. Sterbini. Drawing 2-, 3- and 4-colorable graphs in $o(n^2)$ volume. In S. North, editor, *Graph Drawing (Proc. GD '96)*, volume 1190 of *Lecture Notes Comput. Sci.*, pages 53–62. Springer-Verlag, 1997.

[3] M. Chrobak, M. T. Goodrich, and R. Tamassia. Convex drawings of graphs in two and three dimensions (preliminary version). In *Proceedings of the twelfth annual symposium on Computational geometry*, pages 319–328. ACM Press, 1996.

[4] R. F. Cohen, P. Eades, T. Lin, and F. Ruskey. Three-dimensional graph drawing. *Algorithmica*, 17(2):199–208, 1997.

[5] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing*. Prentice Hall, Upper Saddle River, NJ, 1999.

[6] E. Di Giacomo, G. Liotta, and S. K. Wismath. Drawing series-parallel graphs on a box. In *Proc. of the 14th Canadian Conference on Computational Geometry (CCCG), University of Lethbridge, Alberta, Canada*, 2002.

[7] G. Ding and B. Oporowski. On tree-partitions of graphs. *Discrete Math.*, 149(1–3):45–58, 1996.

[8] V. Dujmovi´c, P. Morin, and D. R. Wood. Pathwidth and three-dimensional straight line grid drawings of graphs. In M. T. Goodrich and S. Koburov, editors, *Graph Drawing (Proc. GD '02)*, volume 2528 of *Lecture Notes Comput. Sci.*, pages 42–53. Springer-Verlag, 2002.

[9] V. Dujmovi´c and D. R. Wood. Tree-partitions of k-trees with application in graph layout. Technical Report TR-02-03, School of Computer Science, Carleton University, 2002.

[10] V. Dujmovi´c and D. R. Wood. Tree-partitions of k-trees with application in graph layout. In *Workshop on Graph Theoretic Concepts in Computer Science (Proc. WG '03)*, Lecture Notes Comput. Sci. Springer-Verlag, to appear.

[11] S. Felsner, G.Liotta, and S. K. Wismath. Straight line drawings on restricted integer grids in two and three dimensions. In P. Mutzel, M. Junger, and S. Leipert, editors, *Graph Drawing (Proc. GD '01)*, volume 2265 of *Lecture Notes Comput. Sci.* Springer-Verlag, 2001.

[12] A. Garg, R. Tamassia, and P. Vocca. Drawing with colors. In *Proc. 4th Annu. European Sympos. Algorithms*, volume 1136 of *Lecture Notes Comput. Sci.*, pages 12–26. Springer-Verlag, 1996.

[13] J. Pach, T. Thiele, and G. T´oth. Three-dimensional grid drawings of graphs. In G. Di Battista, editor, *Graph Drawing (Proc. GD '97)*, volume 1353 of *Lecture Notes Comput. Sci.*, pages 47–51. Springer-Verlag, 1997.

[14] D. R. Wood. Queue layouts, tree-width, and three-dimensional graph drawing. In M. Agrawal and A. Seth, editors, *Proc. 22nd Foundations of Software Technology and Theoretical Computer Science (FSTTCS '02)*, volume 2556 of *Lecture Notes Comput. Sci.*, pages 348–359. Springer-Verlag, 2002.

Appendix

Pseudo-Code of Algorithm 2TVISITORDER()

2TVISITORDER(G,T)
Input: A 2-tree G, and an equipped tree partition T of G.
Output: A total ordering of the nodes of T.
$Q \leftarrow$new queue();
Let ρ be the root of T;
$Q.enqueue(\rho)$;
$i \leftarrow 0$;
while $i < N_T$
$\mu \leftarrow Q.dequeue()$;
$visitorder(\mu) \leftarrow i$;
Let $e_0, e_1, \ldots, e_{h-1}$ be the sequence of black edges of G_μ sorted according to the left-to-right ordering, i.e. e_g is to the left of e_{g+1} $(0 \leq g \leq (h-2))$.
for $g = 0$ **to** $h-1$
Let $\nu_0, \nu_1, \ldots, \nu_{h-1}$ be the children of μ whose parent clique is e_g;
for $j = 0$ **to** $h-1$
$Q.enqueue(\nu_j)$
endfor
endfor
Let $e_0, e_1, \ldots, e_{h-1}$ be the sequence of white edges of G_μ sorted according to the left-to-right ordering, i.e. e_g is to the left of e_{g+1} $(0 \leq g \leq (h-2))$.
for $g = 0$ **to** $h-1$
Let $\nu_0, \nu_1, \ldots, \nu_{h-1}$ be the children of μ whose parent clique is e_g;
for $j = 0$ **to** $h-1$
$Q.enqueue(\nu_j)$
endfor
endfor
$i \leftarrow i+1$;
endwhile

Algorithm 2: Algorithm 2TVISITORDER()

Detailed proof of Theorem 12

Let G be a k-tree and let T be an equipped tree partition of G. We assume that in the track layout of the pertinent graph of each node of T, no edge has its two vertices on the same track. We will show later that this is not a restrictive assumption.

Let N_T be the number of nodes of T. We now define a total ordering for the nodes of T such that each node μ of T is given a number, denoted as $visitorder(\mu)$, in the range $[0, N_T - 1]$. This is achieved by performing a particular version of a breadth first search of T, in which the children of a node μ are grouped according to the type of their parent clique and within each group they are sorted according to the left-to-right ordering of their parent cliques. A precise description of such an ordering procedure is given in Algorithm KTVISITORDER(). For each node μ of T, we associate each type of k-clique of the track layout of G_μ with an integer in the range $[0, c_\mu - 1]$.

KTVISITORDER(G,T)
Input: A k-tree G, and an equipped tree partition T of G.
Output: A total ordering of the nodes of T.

```
Q ←new queue();
Let ρ be the root of T;
Q.enqueue(ρ);
i ← 0;
while i < N_T
    μ ← Q.dequeue();
    visitorder(μ) ← i;
    for j = 0 to c_μ − 1
        Let {C_0, C_1, ..., C_{h−1}} be the set of the k-cliques of the track layout of G_μ of type
        j such that C_g < C_{g+1} (0 ≤ g ≤ h − 2);
        for g = 0 to h − 1
            Let {ν_0, ν_1, ..., ν_{m−1}} be the children of μ such that their parent clique is C_g;
            for l = 1 to m
                Q.enqueue(ν_l);
            endfor
        endfor
    endfor
    i ← i + 1;
endwhile
```

Algorithm 3: Algorithm KTVISITORDER()

We are now ready to present our algorithm to compute a track layout of a k-tree G. Similar to the approach of Section 4, the algorithm receives as input G and an equipped tree partition T of G. We assign the t_T-track layout of each G_μ in T to one of $(2c_T+1)$ different t_T-prisms chosen according to the order defined by Algorithm KTVISITORDER(). A detailed description of our strategy is given in Algorithm KTTRACKLAYOUT(). In the pseudo-code $P_0, \ldots, P_{2c_T}$ denote $2c_T+1$ t_T-prisms and the tracks of each prism P_j ($j = 0,\ldots,2c_T$) are numbered $t_T\,j$, $t_T\,j+1$ and $t_T\,j+2$.

Since each vertex v of G is given a distinct pair $(track(v), order(v))$, we have that Algorithm KTTRACKLAYOUT() defines a track assignment. The next lemmas prove that such an assignment is a track layout.

Lemma 13 *Let G be a k-tree and let T be an equipped tree partition of G. Algorithm* KTTRACKLAYOUT*() computes a track assignment without overlaps.*

Sketch of Proof. An edge e cannot have its two vertices on the same track. Indeed, if e is a jumping edge its vertices are on different tracks because they are on different prisms. If e is not a jumping edge it must belong to a pertinent graph G_μ of a node μ of T and its vertices are on different tracks because in the track layout of G_μ no edge has both endvertices in a same track. It follows that the track assignment computed by Algorithm 2TTRACKLAYOUT() cannot have overlaps. ■

Lemma 14 *Let G be a k-tree, let $\Gamma(G)$ be a track-layout of G computed by Algorithm* KTTRACKLAYOUT*(). Let $e_0 = (u_0, v_0)$ be a jumping edge such that u_0 is its parent vertex. Let $e_1 = (u_1, v_1)$ be a jumping edge such that u_1 is on the same track as u_0 and v_1 is on the same track as v_0. Then u_1 is the parent vertex of e_1.*

Sketch of Proof. Let G_{μ_0} be the pertinent graph containing u_0 and let P_i be the t_T-prism whose tracks contains the vertices of G_{μ_0}. Vertex v_0 is in the pertinent graph of one of the children of μ_0, thus it is on

KTTRACKLAYOUT(G,T)
Input: A k-tree G and an equipped tree partition T of G.
Output: A track layout of G on $(2c_T+1)t_T$ tracks.
KTVISITORDER(G,T);
Let ρ be the root of T;
$\Delta \leftarrow |G_\rho|$;
for $i = 1$ **to** $N_T - 1$
Let μ be the node of T such that $visitorder(\mu) = i$;
Let λ be the parent of μ;
Let P_j be the prism whose tracks contains the vertices of G_λ;
Let h be the type of C_μ;
$p \leftarrow (j+h) \mod (2c_T+1)$;
foreach vertex v of G_μ
$track(v) \leftarrow track(v) + t_T p$;
$order(v) \leftarrow order(v) + \Delta$;
endfor
$\Delta \leftarrow \Delta + |G_\mu|$;
endfor

Algorithm 4: Algorithm KTTRACKLAYOUT()

a track of one of the prisms in the set $P^+ = \{P_j \mid j = (i+k) \mod (2c_T+1) \ \ 1 \le k \le c_T\}$. Suppose as a contradiction that u_1 is the child vertex of e_1. Let G_{μ_1} be the pertinent graph containing u_1. Since u_1 is on the same track as u_0 then the vertices of G_{μ_1} are in the tracks of P_i. Vertex v_1 must be in one of the prisms of the set $P^- = \{P_j \mid j = (i-k) \mod (2c_T+1) \ \ 1 \le k \le c_T\}$. Since $P^+ \cap P^- = \emptyset$ then v_0 and v_1 are in different prisms and hence in different tracks. This contradicts the hypothesis. ■

Lemma 15 *Let G be a k-tree and let T be an equipped tree partition of G. Algorithm* KTTRACKLAYOUT*() computes a track assignments without X-crossings.*

Sketch of Proof. Let $e_0 = (u_0, v_0)$ and $e_1 = (u_1, v_1)$ be two edges of G. If e_0 and e_1 form an X-crossing in the track assignment, then the two vertices of e_0 are on the same two tracks as the two vertices of e_1. Therefore both edges must be either jumping or non-jumping.

Consider first the case that they are both non-jumping edges; e_0 and e_1 can either belong to the same pertinent graph G_μ or they are edges of two different pertinent graphs G_μ and G_ν. If they formed an X-crossing in the first case then there would be an X-crossing in the track layout of G_μ equipping T, which is impossible. If there were an X-crossing in the second case, then either $u_0 < u_1$ and $v_1 < v_0$ or $u_1 < u_0$ and $v_0 < v_1$. But if $visitorder(\mu) < visitorder(\nu)$ then G_μ is to the left of G_ν or G_ν is to the left of G_μ by construction. This implies that e_0 is to the left of e_1 or e_1 is to the left of e_0. Again a contradiction.

It remains to consider the case that e_0 and e_1 are two jumping edges. We assume without loss of generality that u_0 is the parent vertex of e_0 and that u_0 and u_1 are on the same track. By Lemma 7, u_1 is a parent vertex of e_1. Let μ_0, μ_1, ν_0 and ν_1 be the nodes whose pertinent graphs contain u_0, u_1, v_0 and v_1, respectively. It follows that u_0 is a vertex in the parent clique C_{ν_0} of ν_0 and that u_1 is a vertex in the parent clique C_{ν_1} of ν_1.

If $\mu_0 = \mu_1$ and $C_{\nu_0} = C_{\nu_1}$ then $u_0 = u_1$ because each clique has only one vertex on a single track. In this case a crossing is not possible because e_0 and e_1 have a vertex in common. If $\mu_0 = \mu_1$ and C_{ν_0} is to the left of C_{ν_1} then u_0 is to the left of u_1 and v_0 is to the left of v_1. Therefore an X-crossing is not possible. If $\mu_0 \neq \mu_1$, assume without loss of generality that $visitorder(\mu_0) < visitorder(\mu_1)$. The ordering given by

Algorithm KTVISITORDER() is such that $visitorder(v_0) < visitorder(v_1)$. This implies that u_0 is to the left of u_1 and v_0 is to the left of v_1. It follows that an X-crossing is impossible also in this case. ■

Lemma 16 *Let G be a 3-tree. There is an equipped tree partition T of G such that $t_T = 15$ and $c_T = 180$.*

Sketch of Proof. Let T be any tree partition of G. Since G is a 3-tree the pertinent graph G_μ of each node μ of T is a 2-tree and then by Theorem 9 it admits a 15-track layout. Therefore T can be equipped so that $t_T = 15$. The value of c_T is the maximum number of types of cliques of size 3 in the track layouts equipping T.

Consider a 2-tree G' and an equipped tree partition T' of G'. A 3-clique C cannot have all its vertices in the pertinent graphs $G_{\mu'}$ of a single node μ' of T' since $G_{\mu'}$ is a tree. Also, C cannot have vertices in the pertinent graphs of three different nodes, because the three different nodes would be mutually adjacent in T' and then there would be a cycle in T'. Thus the vertices of C are in the pertinent graphs of exactly two adjacent nodes.

In the track layout of G' computed as described in Section 4 the track layouts of the nodes of T' uses five different 3-prisms. Suppose the prisms are numbered from 0 to 4 and consider a prism P_i ($0 \leq i \leq 4$). All jumping edges whose parent vertices are in P_i have their child vertex either in $P_{(i+1) \mod 5}$ or in $P_{(i+2) \mod 5}$. Consider prisms P_i and $P_{(i+1) \mod 5}$. A 3-clique whose vertices are in P_i and $P_{(i+1) \mod 5}$ has one vertex in P_i and the other two in $P_{(i+1) \mod 5}$, or vice versa. The vertex that is in P_i can be in 3 possible different track. The two vertices in $P_{(i+1) \mod 5}$ can be in 3 possible different pair of track. It follows that there are 9 possible types of cliques having a vertex in P_i and the other two in $P_{(i+1) \mod 5}$. Symmetrically, there are 9 possible types of cliques having two vertices in P_i and the other one in $P_{(i+1) \mod 5}$. Therefore we have 18 possible types of 3-cliques with vertices in P_i and $P_{(i+1) \mod 5}$. By an analogous reasoning we have 18 possible types of 3-cliques with vertices in P_i and $P_{(i+2) \mod 5}$. For each P_i we have 36 possible types of 3-cliques containing jumping edges whose parent vertices are in P_i. Since the number of prisms is 5, we have that $c_T = 5 \cdot 36 = 180$. ■

The following lemma present a simple proof that every k-tree has a track layout on a constant number of tracks. This number is bigger than what needed and is bigger, as k grows, than the values given in [9, 10]. A better bound will be computed in a following lemma by using a more refined analysis similar to the one presented in [9, 10].

In the following t_k denotes the track number of a k-tree, i.e. the maximum number of tracks such that every k-tree admits a track layout on them. Also, c_k denotes the maximum number of types of cliques of size $k+1$ in any track layout of a k-tree.

Lemma 17 *Let G be a k-tree with $k \geq 3$. G has track number t_k that is given by the following recursive equation:*

$$t_k \leq (2c_{k-1} + 1)t_{k-1}$$
$$c_k \leq \binom{t_k}{k+1}$$

with $t_2 = 15$ and $c_2 = 180$.

Sketch of Proof. We prove the statement by induction on k. The base case is $k = 3$. Let G be a 3-tree and let T be an equipped tree partition of G. By executing Algorithm KTTRACKLAYOUT() on G we obtain a track layout of G with $t_3 \leq (2c_2 + 1)t_2$, by Lemma 15. By Lemma 16 $t_2 = 15$ and $c_2 = 180$. Suppose the statement is true for $k-1$, and let G be a k-tree. Let T be a tree partition of G. By induction we can equip T with the track layout of the pertinent graph of each node μ of T. Notice that in Algorithm KTVISITORDER() and Algorithm KTTRACKLAYOUT(), the input is assumed to be a non-partial k-tree, while the pertinent graph

of each node μ is a partial k-tree. Thus in order to recursively apply algorithm to the pertinent graph of each node μ, it has to be augmented to a non-partial k-tree.

We have $t_T = t_{k-1}$ and $c_T = c_{k-1}$. By executing Algorithm KTTRACKLAYOUT() on G we obtain a track layout of G on $t_k \leq (2c_{k-1}+1)t_{k-1}$, by Lemma 15. Concerning the value of c_k, recall that c_k is the number of types of cliques of size $k+1$ in the track layout of G. This number is at most the number of possible choices of $k+1$ tracks among t_k and hence

$$c_k \leq \binom{t_k}{k+1}.$$

■

At the beginning of this Section we made the assumption that the track layouts that equip the nodes of the tree partition of G are such that no edges has its vertices in the same track. We observe that this is not a restrictive assumption. Namely, as shown in the proof of Lemma 13 if the track layouts that equip T have the property that no edge has both vertices in a track, then also the track layout of G has the same property. As described in the proof of Lemma 17 the track layout of G is computed recursively. The base step of the recursion is $k = 2$. By Lemma 13 the property above holds for the track layout of 2-trees. Therefore we may assume that the same property holds for tha track layout of any k-tree.

The bound given in Lemma 17 for c_k is largely overestimated. This lead to an upper bound for the track number of a k-tree that is not the best possible. By using a more refined analysis, similar to the one presented in [9, 10], it is possible to further reduce the bound on c_k. In the following Lemma $c_{k,i}$ denotes the maximum number of types of cliques of size i in any track layout of a k-tree. A clique of size i can have all the vertices in the pertinent graph of a single node of T or in the pertinent graph of two adjacent vertices of T. The clique having all vertices in the pertinent graph of a single node of T are called *intra-node cliques*, while those having the vertices in the pertinent graph of two adjacent vertices of T are called *inter-node cliques*. The number of intra-node cliques of size i is denoted as $c'_{k,i}$ while the number of inter-node cliques is denoted as $c''_{k,i}$. According to these definition c_k is equal, by definition, to $c_{k,k+1}$.

Lemma 18 *Let G be a k-tree with $k \geq 3$. G has track number at most t_k where t_k is given by the following recursive equation:*

$$\begin{aligned}
t_k &= (2c_{k-1,k}+1)t_{k-1} \\
c_{k,i} &= c'_{k,i} + c''_{k,i} && (1 \leq i \leq k+1) \\
c'_{k,i} &= (2c_{k-1,k}+1)c_{k-1,i} && (1 \leq i \leq k) \\
c''_{k,i} &= (2c_{k-1,k}+1)c_{k-1,k}\textstyle\sum_{j=1}^{i-1} c_{k-1,j}c_{k-1,i-j} && (1 \leq i \leq k+1)
\end{aligned}$$

with $t_2 = 15$, $c_{2,1} = 15$, $c_{2,2} = 105$, $c_{2,3} = 180$ and $c'_{k,k+1} = 0$.

Sketch of Proof. A clique cannot have vertices in the pertinent graphs of more than two different nodes, because these nodes would be mutually adjacent in T and then there would be a cycle in T. Thus $c_{k,i}$ is given by the sum of the intra-node cliques and the inter-node cliques, i.e. $c_{k,i} = c'_{k,i} + c''_{k,i}$.

Computation of $c'_{k,i}$ An intra-node clique can have size at most k because the pertinent graph of each node is a partial $k-1$-tree. Therefore $c'_{k,k+1} = 0$. For any $i \leq k$, an intra-node clique C of size i in G is also a clique of size i in G_μ, for some node μ of T. For any node μ of T, the number of types of cliques of size i in G_μ is $c_{k-1,i}$ by definition. Since the nodes of T are distributed on $(2c_{k-1,k}+1)$ prisms then each type of cliques is repeated $(2c_{k-1,k}+1)$ times and therefore $c'_{k,i} = (2c_{k-1,k}+1)c_{k-1,i}$.

Computation of $c''_{k,i}$ For any $i \leq k$, an inter-node clique C of size i has a set of j $(1 \leq j \leq i-1)$ vertices in the pertinent graph G_μ of a node μ of T and a set of $i-j$ vertices in the pertinent graph G_ν of a node ν of T adjacent to μ. The vertices in the two sets are connected by jumping edges. Consider a prism

P_i $(0 \leq i \leq (2c_{k-1,k}+1))$. All jumping edges whose parent vertices are in P_i have their child vertex in one of the prisms $P_{(i+1) \mod (2c_{k-1,k}+1)}$, $P_{(i+2) \mod (2c_{k-1,k}+1)}$, $\ldots$, $P_{(i+c_{k-1,k}) \mod (2c_{k-1,k}+1)}$. Consider prisms P_i and $P_{(i+1) \mod (2c_{k-1,k}+1)}$. There are $c_{k-1,j}$ types of cliques of size j in P_i and $c_{k-1,i-j}$ types of cliques of size $i-j$ in $P_{(i+1) \mod (2c_{k-1,k}+1)}$. Thus the number of types of inter-node i-cliques with vertices on prisms P_i and $P_{(i+1) \mod (2c_{k-1,k}+1)}$ is

$$\sum_{j=1}^{i-1} c_{k-1,j} c_{k-1,i-j}.$$

There is the same number of types of inter-node i-cliques with vertices in P_i and in each of the $P_{(i+2) \mod (2c_{k-1,k}+1)}$, $\ldots$, $P_{(i+c_{k-1,k}) \mod (2c_{k-1,k}+1)}$. Therefore, for each P_i we have

$$c_{k-1,k} \sum_{j=1}^{i-1} c_{k-1,j} c_{k-1,i-j}$$

possible types of inter-node i-clique containing jumping edges whose parent vertices are in P_i. Since the number of prisms is $(2c_{k-1,k}+1)$, we have that

$$c''_{k,i} = (2c_{k-1,k}+1) c_{k-1,k} \sum_{j=1}^{i-1} c_{k-1,j} c_{k-1,i-j}.$$

The values of t_2 and $c_{2,3}$ follow immediately from Lemma 16. The value of $c_{2,1}$ is clearly 15 while the value of $c_{2,2}$ can be computed as follows. A clique of size two, i.e. an edge, in a 2-tree is either an intra-node clique or an inter-node clique. The types of intra-node cliques are 3 for each node, because there is no edge having two vertices in a same track, and since the track layout of each node is in the tracks of one among five 3-prisms then each type is replicated 5 times and hence the number of possible types of intra-node cliques is 15. The inter-node cliques have their vertices in two different 3-prisms. For any pair of prisms the number of possible types of such cliques is 9. The jumping edges with the parent vertex in a specific prism, say P_i, has the child vertex either in $P_{(i+1) \mod 5}$ or in $P_{(i+2) \mod 5}$, hence we have 18 possible types of cliques with the parent vertex in P_i. Since the prisms are five, we have $5 \cdot 18 = 90$ possible types of inter-node cliques, and therefore $c_{2,2} = 90 + 15 = 105$. ■

The results of this section are summarized by the following theorem.

Theorem 12 *Every k-tree G has track number at most t_k, where t_k is given by the following recursive equation:*

$$\begin{array}{ll} t_k = (2c_{k-1,k}+1)t_{k-1} & \\ c_{k,i} = c'_{k,i} + c''_{k,i} & (1 \leq i \leq k+1) \\ c'_{k,i} = (2c_{k-1,k}+1)c_{k-1,i} & (1 \leq i \leq k) \\ c''_{k,i} = (2c_{k-1,k}+1)c_{k-1,k} \sum_{j=1}^{i-1} c_{k-1,j} c_{k-1,i-j} & (1 \leq i \leq k+1) \end{array} \tag{2}$$

with $t_2 = 15$, $c_{2,1} = 15$, $c_{2,2} = 105$, $c_{2,3} = 180$ and $c'_{k,k+1} = 0$.

Furthermore, G admits a track drawing with volume at most $2t_k \times p \times p\lceil \frac{n}{t_k} \rceil$, where p is the smallest prime number greater than $2t_k$.