Drawing Ordered $(k-1)$–Ary Trees on k–Grids

Wolfgang Brunner and Marco Matzeder

University of Passau
94030 Passau, Germany
{brunner,matzeder}@fim.uni-passau.de

Abstract. We explore the complexity of drawing ordered $(k-1)$–ary trees on grids with k directions for $k \in \{4, 6, 8\}$ and within a given area. This includes, e. g., ternary trees drawn on the orthogonal grid. For aesthetically pleasing tree drawings on these grids, we additionally present various restrictions similar to the common hierarchical case. First, we generalize the $\mathcal{NP}$–hardness of minimal width in hierarchical drawings of ordered trees to $(k-1)$–ary trees on k–grids and then we generalize the Reingold and Tilford algorithm to k–grids.

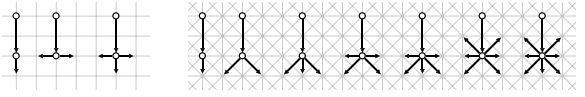
1 Introduction

Drawing trees is an important area in graph drawing. There are three main styles: hierarchical, radial and orthogonal [14]. For hierarchical tree drawings the linear time algorithm by Reingold and Tilford [16] is commonly used. These drawings fulfill various aesthetic criteria: vertices on the same level of the tree are placed on a horizontal line, children maintain their order, parents are centered above their children, a certain minimum horizontal distance is guaranteed, edges do not cross, and isomorphic subtrees are identically drawn up to translation. The resulting drawings require $\mathcal{O}(n^2)$ area. However, the problem of determining the minimum width of such drawings of ordered binary trees is $\mathcal{NP}$–hard [1,18]. For unordered trees the $\mathcal{NP}$–hardness was shown by Marriott and Stuckey [15].

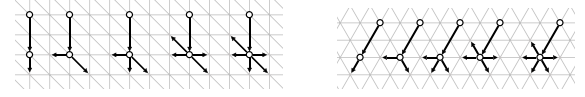
Another drawing style are radial drawings. Eades [8] presented an $\mathcal{O}(n)$ time algorithm where the root is placed at the center of the drawing. The subtrees are placed into sectors around the root, whose widths are determined by the number of their leaves. The vertices are placed onto concentric circles and the parents are centered with respect to their children. In contrast to hierarchical drawings, this algorithm positions the vertices on real coordinates.

The third approach are drawings of trees on grids [6,7,12,17]. Vertices are positioned on integer coordinates and the edges are either orthogonal straight lines, arbitrary polylines, or orthogonal polylines. In the following we consider drawings with edges as straight lines. In the case of unordered ternary trees on the orthogonal grid it turns out to be a $\mathcal{NP}$–hard problem to decide whether or not there is a drawing with unit edge length or within given area [5]. The same holds true for 5–ary trees on the hexagonal grid with six directions [3]. Additionally, the logic engine introduced by Eades and Whitesides [10] can be

U. Brandes and S. Cornelsen (Eds.): GD 2010, LNCS 6502, pp. 105–116, 2011.

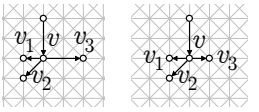


(a) $k = 4$ (b) $k = 8$

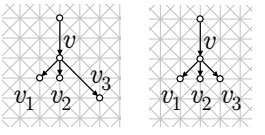


(c) $k = 6$ (asymmetric) (d) $k = 6$ (symmetric)

Fig. 1. Patterns on the k–grids



(a) O (b) OL



(c) OP (d) OLP

Fig. 2. Drawing styles

used to prove these $\mathcal{NP}$–hardness results. If only three directions on the orthogonal grid are allowed for straight line edges (west, south, and east), the algorithms use tree folding techniques to achieve tree drawings with a small area [6,7,12,17]. Chan et. al. proved an area bound of $\mathcal{O}(n \log n)$ for upward drawings and of $\mathcal{O}(n \log \log n)$ for non–upward drawings (all four orthogonal directions) of binary trees [6]. However, such drawings are not really pleasing since subtrees may be arbitrarily nested. In h–v layouts using two directions on the orthogonal grid complete binary trees only need $\mathcal{O}(n)$ area and arbitrary binary trees need $\Theta(n \log n)$ area [7]. For binary trees on the orthogonal grid Eades et. al. [9] developed an $\mathcal{O}(n^2)$ time algorithm for non order–preserving h–v drawings on minimal area.

Ordered binary trees can be drawn on the orthogonal grid within $\mathcal{O}(n^{1.5})$ area and $\Theta(n^2)$ area is a tight bound for ternary trees [11]. In this work we draw ordered $(k-1)$–ary trees on k–grids with $k \in \{4, 6, 8\}$ directions, e. g., ternary trees on the orthogonal grid with four possible directions. We prove that it is a $\mathcal{NP}$–hard problem to decide whether or not an ordered $(k-1)$–ary tree has an order–preserving drawing on the k–grid within a given area. Moreover, we introduce further aesthetics transferred from the hierarchical case [16] leading to more comprehensible and symmetric tree drawings on k–grids. In most cases we show that the decision problem remains $\mathcal{NP}$–hard.

2 Preliminaries

The *orthogonal grid* is the infinite plane graph whose vertices have integer coordinates and whose edges link pairs of vertices at unit distance either vertically or horizontally. For all vertices (x, y) we extend the orthogonal grid with four directions to the *hexa grid* with six directions by adding undirected edges between

(x, y) and $(x+1, y-1)$, see the underlying grid in Fig. 1(c). A further extension is achieved by introducing undirected edges between integer coordinates (x, y) and $(x+1, y+1)$ resulting in the *octa grid* with eight directions, see Fig. 1(b). We call these three grids *k–grids* with $k \in \{4, 6, 8\}$. In the literature the hexa grid is called hexagonal grid [2,13] and triangular grid [19] as well. The *distance* between vertices u and v with coordinates (u_x, u_y) and (v_x, v_y) on a k–grid is defined by $\|(u, v)\| = \max(|u_x - v_x|, |u_y - v_y|)$. A *path* $v_1 \rightsquigarrow v_n$ is a sequence of vertices $(v_1, \ldots, v_n)$ in a graph with edges (v_i, v_{i+1}) and $i \in \{1, \ldots, n-1\}$. A *straight path* is a path where all edges have the same direction.

Let $T = (V, E)$ be a *tree* and $v \in V$. We use T_v for the subtree with root v. The *depth* of a vertex v is the number of edges of the path from the root to v. The *height* of T is the depth of the deepest vertex. We call two vertices *siblings* if they have the same parent. Trees with outdegree up to d are called *d–ary trees*. A tree *embedding* $\Gamma_k(T)$ of a tree T on a k–grid is a mapping Γ_k which specifies for each vertex $v \in V$ distinct integer coordinates $\Gamma_k(v) = (x, y)$ on a k–grid. Γ_k maps an edge $e \in E$ on a straight path $\Gamma_k(e)$ of the k–grid whose endpoints are the mappings of vertices linked by e. We use the terms *drawing* and embedding synonymously. The *length* of an edge $e \in E$ is the distance between its incident vertices and the *length* of a straight path p is the sum of its edge lengths. We call a vertex v of a $(k-1)$–ary tree on a k–grid *full* if it has $k-1$ children. The relative direction of each outgoing edge of a full vertex is determined, due to the given order of the tree. We call a path of full vertices *full path*.

Similar to the hierarchical drawing style of trees [16,18,20], we produce *planar* drawings preserving a given *minimal distance* and *isomorphic subtrees*. We discuss drawings of ordered trees on k–grids which are *order–preserving* and satisfy the given aesthetics. A tree drawing is *locally uniform* if for each vertex its outgoing edges have the same length, see Fig. 2(b). In a *pattern drawing* of a $(k-1)$–ary tree on the k–grid, the edge directions of the outgoing edges of each vertex have a prescribed angle with respect to the direction of the incoming edge, see Fig. 2(c). All patterns for the various k–grids are shown in Fig. 1. The patterns in Fig. 1(c) do not seem to be symmetric, but changing the underlying grid by rotating two axes yields a more pleasing symmetric drawing, see Fig. 1(d).

In the following we define various restrictions for order–preserving tree drawings on k–grids. Figure 2 presents an ordered tree T with vertices v and its children v_1, v_2 and v_3 in various order–preserving drawing styles. In Fig. 2(a) we show a drawing of T where no additional restrictions are given, called *O_k–drawing*. If a drawing is additionally local uniform on the k–grid, we call it an *OL_k–drawing*, see Fig. 2(b). We call an order–preserving pattern drawing on a k–grid *OP_k–drawing*. For an example see Fig. 2(c). If we combine these two properties, we obtain order–preserving locally uniform pattern drawings, called *OLP_k–drawings*, see Fig. 2(d). In such drawings the children of a vertex v are positioned axially symmetrical with respect to the incoming edge of v, which is analogous to placing the parent centered over its children in the hierarchical case. In Sect. 4 we introduce a drawing algorithm generating OLP_k–drawings.

3 $\mathcal{NP}$–Hardness Results

We show that drawing $(k-1)$–ary trees within a given area is $\mathcal{NP}$–hard for O_k–, OL_4–, OP_k–, and OLP_k–drawings. Clearly all these problems are in $\mathcal{NP}$.

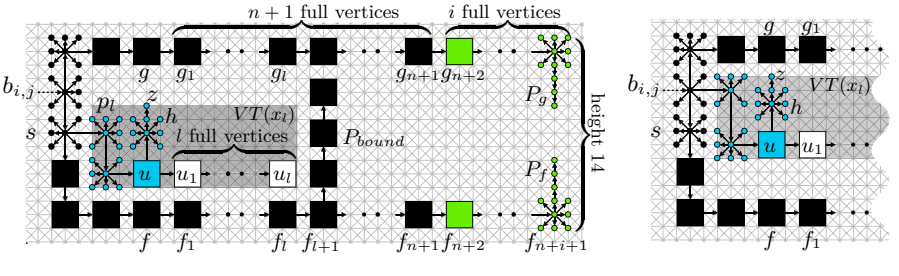
3.1 O–Drawings

In this section we prove the complexity of order–preserving drawings.

Theorem 1. *Let $k \in \{4, 6, 8\}$, T be a rooted $(k-1)$–ary tree, and $A \geq 1$. Determining the existence of an O_k–drawing $\Gamma_k(T)$ with area at most A is $\mathcal{NP}$–hard.*

We reduce *3–SAT* by creating a $(k-1)$–ary tree for a given expression in *3–CNF* similarly to the hierarchical case [1,18] in polynomial time. Let the boolean expression E consist of r clauses $F_1, \ldots, F_r$ with n variables $x_1, \ldots, x_n$. Each clause F_i with $i \in \{1, \ldots, r\}$ has three literals, such that $F_i = (y_{i,1} \vee y_{i,2} \vee y_{i,3})$. A literal $y_{i,j}$ with $j \in \{1, 2, 3\}$ in the i–th clause is either a variable x_l or its negation $\overline{x_l}$. In the following, we treat the case $k = 8$, i. e., we construct a 7–ary tree on the octa grid. To simplify the representation of the tree we replace some full vertices and their adjacent leaves by big vertices, called *box vertices* as can be seen in Fig. 3(a) and 3(b). Any label of a box vertex or near a full vertex and its leaves identifies the center vertex.

Variable tree. For a variable x_l of E with $l \in \{1, \ldots, n\}$, we define a unique 7–ary tree, called *variable tree* $VT(x_l)$ as the shaded part in Fig. 3(a). It is the induced subtree of the root p_l having a full path $p_l \rightsquigarrow u$. The full path $u_1 \rightsquigarrow u_l$ is appended to u having length l depending on the index of x_l. We append a further full vertex h above u and a single vertex z at the top of the full vertex h. Note that (the essential part of) the variable tree $VT(x_l)$ of Fig. 3(a) is shown in Fig. 3(b) as well, but there the vertical outgoing edge of p_l has length four instead of three.



(a) $y_{i,j} = x_l$ (b) $y_{i,j} = \overline{x_l}$

Fig. 3. Literal tree $LT(y_{i,j})$ of $y_{i,j}$ with $j = 3$

Literal tree. The j–th literal $y_{i,j}$ in the i–th clause is either x_l or $\overline{x_l}$. We define a literal tree $LT(y_{i,j})$, which contains the corresponding variable tree $VT(x_l)$. The skeleton of a literal tree (see Fig. 3(a)) is the full path $b_{i,j} \rightsquigarrow f_{l+1}$. At f_{l+1} we append a path of three full vertices P_{bound} in vertical direction and a horizontal full path $f_{l+2} \rightsquigarrow f_{n+i+1}$. A similar symmetrical construction is added to the top from $b_{i,j}$ to g_{n+i+1} without the full path P_{bound}.

The full path P_{bound} ensures that if the skeleton of the literal tree is drawn within minimal width, the variable tree is also drawn within minimal width. At the child above the last vertex f_{n+i+1} we append a path P_f of j single vertices (analogous P_g below g_{n+i+1}) as can be seen in Fig. 3(a). Due to this construction, all these subtrees are non isomorphic.

If $y_{i,j} = x_l$, the root p_l of variable tree $VT(x_l)$ is appended to the vertex s, see Fig. 3(a). Otherwise it is appended to $b_{i,j}$, see Fig. 3(b), where just the essential part of the corresponding literal tree is shown.

Minimal height and width of a literal tree: Let the incoming edge of $b_{i,j}$ face to the east. Fig. 3(a) shows a drawing of $LT(y_{i,j})$ with $y_{i,j} = x_l$ of minimal height 14. Note that the directions of the edges are determined by the incoming edge of the root $b_{i,j}$, except for the edges of P_f and P_g. If all vertical edges have minimal length, the drawing has minimal height 14, otherwise the height may increase.

The minimal height of the drawing of $LT(y_{i,j})$ with $y_{i,j} = \overline{x_l}$ is also 14, as can be seen in Fig. 3(b). Observe that if the vertical outgoing edge of p_l gets length three, the height of the drawing increases due to z. For both cases $y_{i,j} = x_l$ and $y_{i,j} = \overline{x_l}$ the drawing of $LT(y_{i,j})$ has minimal width $8 + 3(n + i + 1)$ due to the paths $f \rightsquigarrow f_{n+i+1}$ and $g \rightsquigarrow g_{n+i+1}$.

Clause tree. For a clause $F_i = (y_{i,1} \vee y_{i,2} \vee y_{i,3})$, we define a 7–ary *clause tree* $CT(F_i)$ with root r_i. Each clause tree $CT(F_i)$ has the paths $r_i \rightsquigarrow c_i$, $c_i \rightsquigarrow e_i$, and $r_i \rightsquigarrow d_i$, see Fig. 4. $CT(F_i)$ contains three literal trees $LT(y_{i,1})$, $LT(y_{i,2})$ and $LT(y_{i,3})$ with roots $b_{i,1}$, $b_{i,2}$ and $b_{i,3}$, appended at the corresponding vertices $a_{i,1}$, $a_{i,2}$ and $a_{i,3}$. For an example in Fig. 4, the edge $(a_{1,3}, b_{1,3})$ connects the first clause tree F_1 with its third literal tree $LT(y_{1,3})$.

Minimal height and width of a clause tree: Let the incoming edge of r_i face to the east. As explained above, all important edge directions are fixed in the drawing then. Each clause tree $CT(F_i)$ has a vertical path $c_i \rightsquigarrow e_i$, which determines the minimal height 49, see Fig. 4. If each literal tree in a clause tree has at least height 15, its height is at least $2 + 1 + 15 + 1 + 15 + 1 + 15 = 50$. If at least one literal tree is drawn with height 14 and the remaining two with height 15, the clause tree can be drawn with height 49. The topmost horizontal path $r_i \rightsquigarrow c_i$ of full vertices determines the minimal width $14 + 3(n + i + 1)$.

Tree $T(E)$. For a boolean expression $E = F_1 \wedge \ldots \wedge F_r$ we define a 7–ary tree $T(E)$ with clause trees $CT(F_1), \ldots, CT(F_r)$. The root r_0 of $T(E)$ is the parent of the root r_1 of the first clause tree. Each clause tree $CT(F_i)$ is connected with $CT(F_{i+1})$ ($i \in \{1, \ldots, r-1\}$) by an edge (c_i, r_{i+1}). At the last clause tree $CT(F_r)$ we add a leaf c_{r+1} to the center vertex c_r directing to the east.

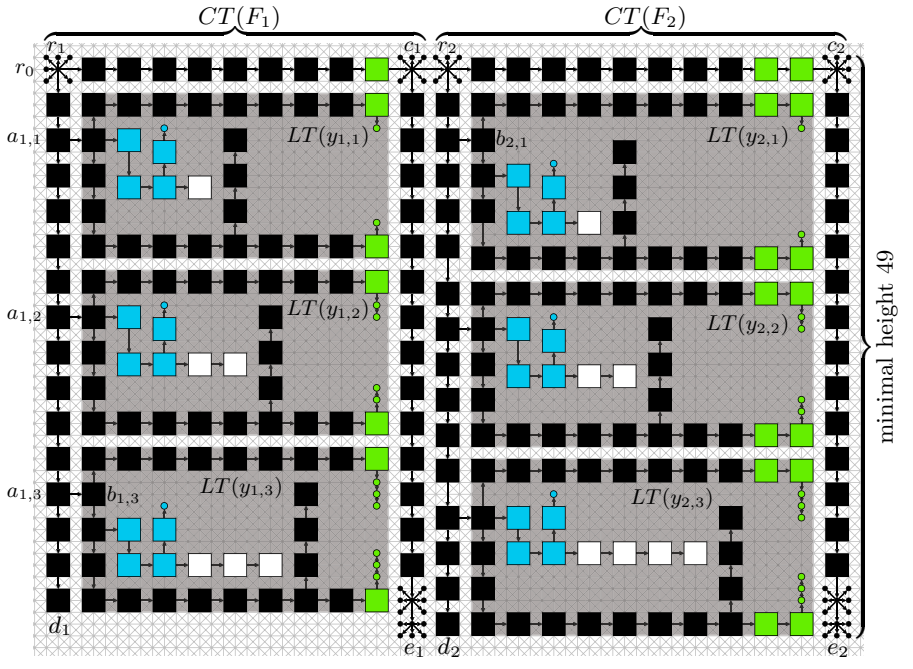


Fig. 4. 7–ary tree $T(E)$ on the octa grid of boolean expression $E = (\overline{x_1} \vee \overline{x_2} \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_4})$ with assignment $\tau(x_1) = \tau(x_2) = false$ and $\tau(x_3) = \tau(x_4) = true$

Minimal height and width of the tree $T(E)$: Let the edge (r_0, r_1) face to the east. The construction generates a horizontal straight path $r_0 \rightsquigarrow c_{r+1}$ at the top ensuring minimal width of $\sum_{i=1}^{r}(14 + 3(n + i + 1)) + (r - 1) = 1.5r^2 + (3n + 19.5)r - 1$. The height of $\Gamma(T(E))$ is the maximal height of the clause trees because the clause trees are placed side by side with aligned top.

Lemma 1. *Let E be a boolean expression in 3–CNF with r clauses and n variables. E is satisfiable if and only if there exists a drawing $\Gamma(T(E))$ of the 7–ary tree $T(E)$ on the octa grid with area at most $A = 49 \cdot W$ with $W = 1.5r^2 + (3n + 19.5)r - 1$.*

Proof. Let E be an arbitrary boolean expression in *3–CNF* with n variables $x_1, \ldots, x_n$ and r clauses $F_1, \ldots, F_r$. We construct the 7–ary tree $T(E)$ as described above.

'⇒': Let $\tau : \{x_1, \ldots, x_n\} \to \{true, false\}$ be a satisfying assignment for the boolean expression E. We construct a tree drawing $\Gamma(T(E))$ on the octa grid with area at most A in the following way. If for a variable x_l the assignment $\tau(x_l) = true$ is given, the variable tree $VT(x_l)$ is drawn as in Fig. 3(a) such that all edges are drawn with minimal length. Otherwise for $\tau(x_l) = false$, we draw all edges with minimal length, except the vertical outgoing edge of p_l, which is drawn with length four (see drawing of the variable tree in Fig. 3(b)).

For every satisfied literal in E we construct a drawing with height 14, else 15. Let $y_{i,j}$ be the j–th literal in the i–th clause of E. If $y_{i,j} = x_l$ and $x_l = true$, we draw the literal tree $LT(y_{i,j})$ as shown in Fig. 3(a), where each edge has minimal length and, thus, the literal tree has minimal height 14. If $y_{i,j} = x_l$ and $x_l = false$, we draw the outgoing edge of p_l in $VT(x_l)$ with length four, such that a vertical edge of the skeleton of $LT(y_{i,j})$ has to be drawn with length four. Therefore, the height of the literal tree is 15. If $y_{i,j} = \overline{x_l}$ and $x_l = true$, in the corresponding variable tree $VT(x_l)$ all edges are drawn with minimal length. In this case the drawing of the literal tree needs height 15, due to z. The last of the four cases is $y_{i,j} = \overline{x_l}$ and $x_l = false$, where we draw the literal tree $LT(y_{i,j})$ as in Fig. 3(b) with height 14.

As one of the three literals of each F_i is satisfied, the corresponding literal tree is drawn with height 14 and the other two literal trees have height at most 15, such that the height of the drawing of $CT(F_i)$ is 49. Hence, the described construction produces drawings of clause trees with height 49 and width $14 + 3(n + i + 1)$. The resulting drawing of $T(E)$ is constructed within a rectangle with height at most 49 and width at most W and, hence, area $49 \cdot W$.

'$\Leftarrow$': Let $\Gamma'(T(E))$ be a drawing with area at most $A = 49 \cdot W$. Suppose, the outgoing egde of r_0 of the drawing of $T(E)$ faces to the east determining all essential edge directions. Due to the vertical and horizontal paths in $\Gamma'(T(E))$, the drawing has at least height 49 and at least width W. Thus, the height is exactly 49 and the width exactly W. As a consequence each edge of the path $r_1 \rightsquigarrow c_r$ has minimal length three. For each F_i this determines the horizontal distance between the vertical paths $r_i \rightsquigarrow d_i$ and $c_i \rightsquigarrow e_i$ of $CT(F_i)$. Thus, the width of this clause tree is $14 + 3(n + i + 1)$.

In the following we construct a satisfying assignment $\tau'(E)$ from this drawing $\Gamma'(T(E))$. As explained above in each clause tree $CT(F_i)$ at least one literal tree $LT(y_{i,j})$ has to have height 14. We assign its variable x_l the boolean value satisfying the literal $y_{i,j}$. This is well defined because it is not possible that the literal trees of a non–negated and a negated literal with the same variable x_l both have height 14 due to the subtree isomorphism property.

Consequently if a literal tree $LT(y_{i,j})$ with $y_{i,j} = x_l$ is drawn with height 14, we assign $\tau'(x_l) = true$. For a drawing of a literal tree $LT(y_{i,j})$ with $x_{i,j} = \overline{x_l}$ and height 14, the outgoing edge of p_l has at least length four. We assign $\tau'(x_l) = false$ constructing a satisfiying assignment of the corresponding literal. Since each clause tree $CT(F_i)$ contains a literal tree with height 14, the assignment τ' satisfies each clause F_i. Thus, E is satisfied. If there are remaining variables x_l with no assignment yet, we assign an arbitrary boolean value. □

For an example, see Fig. 4, where the drawing of the 7–ary tree on the octa grid within height 49 and width $W = 1.5 \cdot 2^2 + (3 \cdot 4 + 19.5) \cdot 2 - 1 = 68$ is given for a boolean expression with two clauses ($r = 2$) and four variables ($n = 4$). The variable $x_1 = false$ appears in the literal $y_{1,1}$ of the first clause F_1 and in $y_{2,1}$ in F_2. The corresponding literal tree of the satisfied literal $y_{1,1} = \overline{x_1}$ has height 14. $LT(y_{2,1})$ has height 15 corresponding to the not satisfied literal $y_{2,1}$. All literals in F_1 are satisfied and all literal trees in $CT(F_1)$ have height 14. Nevertheless,

the minimal height of 49 of $CT(F_1)$ is ensured by the vertical path $c_1 \rightsquigarrow e_1$ of box vertices. F_2 has exactly one satisfied literal $y_{2,2}$ and therefore one literal tree $LT(y_{2,2})$ with height 14.

We have shown that for $k = 8$ the problem of determining the existence of a tree drawing on the octa grid on given area is $\mathcal{NP}$–hard. To show the $\mathcal{NP}$–hardness for $k = 6$ and $k = 4$, the construction of the tree $T(E)$ and the values of the heights and widths have to be adjusted. The idea of the constructions remains mainly the same. This completes the proof of Theorem 1.

3.2 *OL*–Drawings

Theorem 2. *Let T be a rooted ternary tree and $A \geq 1$. Determining the existence of an OL_4–drawing $\Gamma_4(T)$ with area at most A is $\mathcal{NP}$–hard.*

Proof. (Sketch). The construction for OL_4–drawings of ternary trees on the orthogonal grid is similar to the construction of the O_4–drawings from above. The local–uniformity property enforces inserting interspaces between vertical paths of the tree leading to other dimensions. □

For OL_6– and OL_8–drawings this approach does not work because the children of two adjacent locally uniform full vertices would overlap.

3.3 *OP*– and *OLP*–Drawings

Theorem 3. *Let $k \in \{4, 6, 8\}$, T be a rooted $(k-1)$–ary tree, and $A \geq 1$. Determining the existence of an OP_k– or an OLP_k–drawing $\Gamma_k(T)$ with area at most A is $\mathcal{NP}$–hard.*

Proof. The construction for OP_k–drawings for $k \in \{4, 6, 8\}$ is identical to the O_k cases as these drawings already are pattern drawings.

To prove the cases OLP_k for $k \in \{6, 8\}$ we modify the reduction of Sect. 3.1 slightly. The edges connecting full vertices are splitted by one vertex. The outgoing edge of this vertex can have arbitrary length without contradicting the aesthetic local uniformity since it has no siblings. In contrast to Sect. 3.1 and 3.2 the directions of the edges are completely defined by the patterns (see Fig. 1). Thus, for each new vertex the outgoing edge has the same direction as the incoming edge. As a consequence of the construction, the resulting drawing has the same dimensions and the proof is done analogously.

For OLP_4–drawings the proof is identical to the proof of Theorem 2, because an order–preserving local uniform pattern drawing is already given there. □

4 Heuristic

We present a heuristic in Algorithm 1 which produces drawings of $(k-1)$–ary trees on the k–grid. It is an improvement of our algorithm presented in [3] where enclosing hexagons for distance calculations on the hexa grid were used. Our new

algorithm generally creates smaller drawings using more precise contours following the Reingold and Tilford algorithm [16].

To describe the heuristic we need some definitions. $\mathcal{D}_k$ is a set of *direction vectors* dependent on the current k–grid having $2k$ direction vectors. Let $\mathcal{D}_4$ be the set $\{(0,\pm1),(\pm1,\pm1),(\pm1,0)\}$, $\mathcal{D}_6 = \mathcal{D}_4 \cup \left\{\pm(\frac{1}{2},1),\pm(1,\frac{1}{2})\right\}$, and $\mathcal{D}_8 = \mathcal{D}_6 \cup \left\{\pm(\frac{1}{2},-1),\pm(1,-\frac{1}{2})\right\}$. For the vector $(d_1,d_2) = D \in \mathcal{D}_k$ we define the *orthogonal vector* $D_o = (-d_2,d_1)$. A *direction contour* $C_{v,D}$ with $D \in \mathcal{D}_k$ is a (non–strictly) monotonic increasing polyline with respect to D_o of $\Gamma(T_v)$. It is the set of vertices and segments of edges visible along the direction vector $D \in \mathcal{D}_k$. If two neighboring segments/points do not have the same endpoint, they are connected by a segment parallel to D. For an example see the direction contour $C_{v_4,D_{3,4}}$ in Fig. 5(a). The direction contour $C_{l,D}$ of a leaf l trivially consists of the coordinates of l itself for all $D \in \mathcal{D}_k$. We call the set of all direction contours $C_{v,D}$ with $D \in \mathcal{D}_k$ the *contour* C_v of the subtree T_v.

We use a preprocessing step creating a planar drawing on the k–grid. The outgoing edges of each vertex $v \in V$ get the initial length $3^{height(T)-depth(v)-1}$ as in [3], which guarantees planarity for all k–grids. The recursive Algorithm 1 is called with the root v of T_v as input. The output of each recursion step is the contour of C_v of T_v and the updated edge lengths in the drawing of T_v. If v is a leaf, its trivial contour is returned in line 1. The loop in line 1 realizes the recursive call of $getContour(v_i)$ computing the contours C_{v_i} of the subtrees T_{v_i} for each child v_i. The outgoing edges of v are contracted in line 1, see details in Sect. 4.1. Finally, the contour C_v of T_v is created by merging the contours of the subtrees of the children of v in line 1, see details in Sect. 4.2.

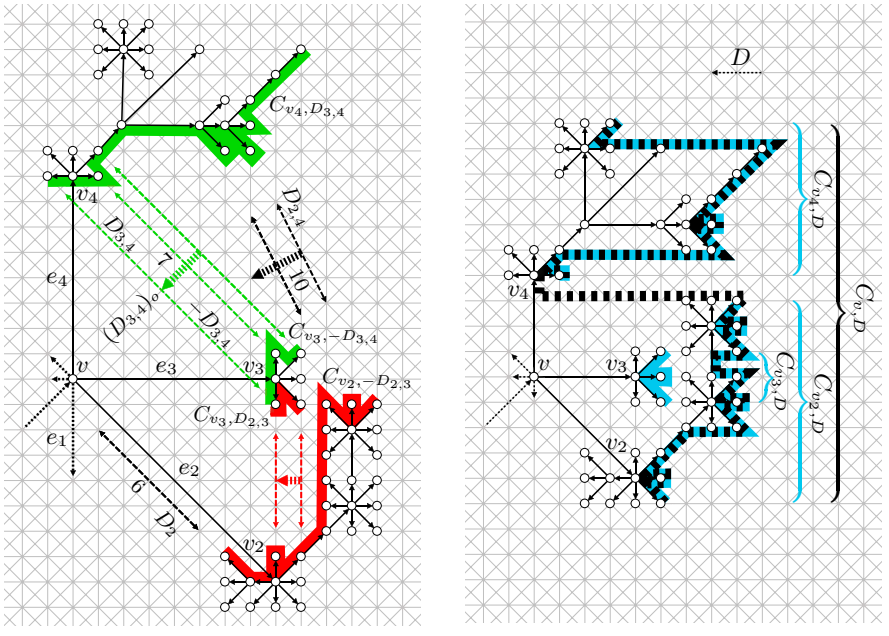
4.1 Contract Edges

To determine the maximal contraction value of the outgoing edges of v, we calculate the distances of all pairs of subtrees and the distances of subtrees to incident edges of v. Let v_i and v_j be children of v with incoming edges e_i and e_j, respectively. First, we calculate the minimal distance between the subtrees T_{v_i} and T_{v_j} using the contours C_{v_i} and C_{v_j}. The difference of the unit vectors $D = \overrightarrow{e_i}/\|\overrightarrow{e_i}\| - \overrightarrow{e_j}/\|\overrightarrow{e_j}\|$ determines the necessary direction contours $C_{v_i,D}$ and $C_{v_j,-D}$ facing each other. We calculate the minimal distance $minDist$ between these two direction contours using a scanline moving along D_o. Each bend of

Algorithm 1. getContour

Input: A vertex v of a planar drawing of a $(k-1)$–ary tree on a k–grid
Output: Contour C_v of v and the contracted edge lengths of T_v

```
1 if v is a leaf then return trivial contour of v
2 C ← ∅
3 foreach child v_i of v do C ← C ∪ {getContour(v_i)}
4 contractEdges(C, v)                          // contract outgoing edges of v
5 C_v ← mergeContours(C, v)
6 return C_v
```



(a) Distance calculation (b) Merge

Fig. 5. Contraction step of a drawing of a 7–ary tree on the octa grid

both direction contours is visited once to determine the distance to the other direction contour along the direction D. The complexity of this step is linear in the number of bends of the direction contours.

Proposition 1. *The number of bends in a direction contour $C_{v,D}$ of a subtree T_v and direction $D \in \mathcal{D}_k$ is linear in the number of vertices in the subtree T_v.*

The maximal possible contraction value of the two subtrees T_{v_i} and T_{v_j} is determined considering the behavior of the two vertices v_i and v_j while contracting the edges e_i and e_j simultaneously by one. If the distance between the two vertices is thereby reduced by two, e. g., T_{v_2} and T_{v_4} in Fig. 5(a), we could shorten e_i and e_j by $\lfloor (minDist - 1)/2 \rfloor$, otherwise the distance is reduced by one and we could shorten these edges by $minDist - 1$, e. g., T_{v_3} and T_{v_4}. This maximal contraction value is calculated for all pairs of subtrees of the children of v. For an example see Fig. 5(a) where the distance between the two contours $C_{v_3,-D_{3,4}}$ and $C_{v_4,D_{3,4}}$ is 7 and the edges e_3 and e_4 could be shortened by 6 without creating an intersection of the two subtrees. To calculate the distance between the subtrees T_{v_2} and T_{v_4} the direction contours $C_{v_2,-D_{2,4}}$ and $C_{v_4,D_{2,4}}$ are used. The distance between these direction contours is 10. Since the subtrees move towards each other, the maximal contraction value considering these two subtrees is $\lfloor \frac{(10-1)}{2} \rfloor = 4$. To prevent the subtrees of the children of v from

overlapping with its incident edges, we have to determine their distances as well. In this case the distance *minDist* is calculated between all incident edges of v and each direction contour C_{v_i,D_i} with direction $D_i = \overrightarrow{e_i} / \|\overrightarrow{e_i}\| \in \mathcal{D}_k$. The edge e_i could be shortened by $minDist - 1$. In Fig. 5(a) the distance between C_{v_2,D_2} and e_1 is 6. Thus, e_2 could be shortened by 5. The final contraction value is the minimum of all contraction values and we shorten the outgoing edges of v by this value. In Fig. 5(a) the outgoing edges can be shortened by 4, see Fig. 5(b).

4.2 Merge Contours

After contracting the edges the contour C_v has to be calculated for each direction $D \in \mathcal{D}_k$. Let $v_1, \ldots, v_i$ with $i \in \{1, \ldots, k-1\}$ be the children of v. Again, we use a scanline procedure along D_o. For the new contour $C_{v,D}$ we use the union of all "visible" parts of $C_{v_1,D}, \ldots, C_{v_i,D}$ and of the outgoing edges of v, as can be seen in Fig. 5(b). There, the merge of the contours $C_{v_2,D}$, $C_{v_3,D}$, $C_{v_4,D}$, the edges (v, v_2), (v, v_3), and (v, v_4) is shown. The merge step runs in linear time due to Proposition 1. As can be seen in Fig. 5(b), a part of the edge (v, v_4) becomes part of the dashed direction contour $C_{v,D}$. The direction contour of $C_{v_3,D}$ is not needed anymore because it is no longer visible along direction D.

4.3 Time Complexity

The calculation of the order–preserving locally uniform pattern drawing with Algorithm 1 has time complexity of $\mathcal{O}(n^2)$. In every recursion step of Algorithm 1 the running time is in $\mathcal{O}(n)$ based on the scan line procedures of the contraction step (Sect. 4.1) and the merge step (Sect. 4.2). The recursive method *getContour*(v) is called for each $v \in V$ exactly once.

Using techniques similar to those used in the Reingold and Tilford algorithm [16] and a more sophisticated merge step, the time complexity of the algorithm can be improved to $\mathcal{O}(n)$. Such an implementation is realized in *Gravisto* [4].

5 Conclusion

We have shown the $\mathcal{NP}$–hardness for several problems of drawing trees on k–grids within a given area for O_k–, OP_k–, OL_4– and OLP_k–drawings. Furthermore we introduced a heuristic producing OLP_k–drawings of ordered $(k-1)$–ary trees on a k–grid guaranteeing the isomorphic subtree property. For the calculation we use contours similar to *threads* of the algorithm of Reingold and Tilford [16].

For unordered trees we conjecture that the Bhatt and Cosmadakis technique [5] or the logic engine [10] can be used to prove the $\mathcal{NP}$–hardness of drawing unordered trees on the octa grid within a given area or with unit edge length.

References

1. Akkerman, T., Buchheim, C., Jünger, M., Teske, D.: On the complexity of drawing trees nicely: Corrigendum. Acta Inf. 40(8), 603–607 (2004)
2. Aziza, S., Biedl, T.C.: Hexagonal grid drawings: Algorithms and lower bounds. In: Graph Drawing, pp. 18–24 (2004)

3. Bachmaier, C., Brandenburg, F.J., Brunner, W., Hofmeier, A., Matzeder, M., Unfried, T.: Tree drawings on the hexagonal grid. In: Tollis, I.G., Patrignani, M. (eds.) GD 2008. LNCS, vol. 5417, pp. 372–383. Springer, Heidelberg (2009)
4. Bachmaier, C., Brandenburg, F.J., Forster, M., Holleis, P., Raitner, M.: Gravisto: Graph visualization toolkit. In: Graph Drawing, pp. 502–503 (2004)
5. Bhatt, S.N., Cosmadakis, S.S.: The complexity of minimizing wire lengths in VLSI layouts. Inf. Process. Lett. 25(4), 263–267 (1987)
6. Chan, T.M., Goodrich, M.T., Kosaraju, S.R., Tamassia, R.: Optimizing area and aspect ratio in straight-line orthogonal tree drawings. Comput. Geom. Theory Appl. 23(2), 153–162 (2002)
7. Crescenzi, P., Di Battista, G., Piperno, A.: A note on optimal area algorithms for upward drawings of binary trees. Comput. Geom. Theory Appl. 2, 187–200 (1992)
8. Eades, P.: Drawing free trees. Bulletin of the Institute of Combinatorics and its Applications 5, 10–36 (1992)
9. Eades, P., Lin, T., Lin, X.: Minimum size h-v drawings. In: Advanced Visual Interfaces, pp. 386–394 (1992)
10. Eades, P., Whitesides, S.: The logic engine and the realization problem for nearest neighbor graphs. Theor. Comput. Sci. 169(1), 23–37 (1996)
11. Frati, F.: Straight-line orthogonal drawings of binary and ternary trees. In: Hong, S.-H., Nishizeki, T., Quan, W. (eds.) GD 2007. LNCS, vol. 4875, pp. 76–87. Springer, Heidelberg (2008)
12. Garg, A., Rusu, A.: Straight-line drawings of binary trees with linear area and arbitrary aspect ratio. J. Graph Algo. App. 8(2), 135–160 (2004)
13. Kant, G.: Hexagonal grid drawings. In: Mulkers, A. (ed.) Live Data Structures in Logic Programs. LNCS, vol. 675, pp. 263–276. Springer, Heidelberg (1993)
14. Kaufmann, M., Wagner, D. (eds.): Drawing Graphs. LNCS, vol. 2025. Springer, Heidelberg (2001)
15. Marriott, K., Stuckey, P.J.: NP-completeness of minimal width unordered tree layout. J. Graph Algorithms Appl. 8(2), 295–312 (2004)
16. Reingold, E.M., Tilford, J.S.: Tidier drawing of trees. IEEE Trans. Software Eng. 7(2), 223–228 (1981)
17. Shin, C.S., Kim, S.K., Chwa, K.Y.: Area-efficient algorithms for straight-line tree drawings. Comput. Geom. Theory Appl. 15(4), 175–202 (2000)
18. Supowit, K.J., Reingold, E.M.: The complexity of drawing trees nicely. Acta Inf. 18, 377–392 (1982)
19. Tamassia, R.: On embedding a graph in the grid with the minimum number of bends. SIAM J. Comput. 16(3), 421–444 (1987)
20. Walker, J.Q.W.: A node-positioning algorithm for general trees. Softw. Pract. Exper. 20(7), 685–705 (1990)