Unfolding Linear Tree Linkages of Low Monotonicity (Long Abstract)

Ching-Hao Liu and Sheung-Hung Poon
Department of Computer Science
National Tsing Hua University, Taiwan
{chinghao.liu,sheung.hung.poon}@gmail.com

Abstract

In this paper, we consider the problem of straightening k-monotone linear trees in two dimensions. First, we show that a 3-monotone linear tree can always be straightened in $O(n)$ moves and $O(n)$ time, and a 4-monotone linear tree can always be straightened in $O(n)$ moves and $O(n \log n)$ time, where n is the number of edges of the tree. Moreover, we provide two linear locked trees of monotonicity more than 4. First, by presenting a locked 5-monotone linear tree T of 14 edges, we determine that the minimum monotonicity of locked linear trees is 5. In addition, we observe that the backbone K of locked tree T contains one zigzag. Therefore, we further present a locked 6-monotone linear tree of 8 edges with straight backbone.

1 Introduction

A (*planar*) *linkage* is a straight-line drawing of a plane graph G in two dimensions with compatible edge lengths. A linkage L is called *linear* if all vertices and all edges of L lie on a line. For linear linkages, we use the *self-touching assumption* (refer to [1, 5] for the formal definition) as the theoretical basis: the edges of a linkage are allowed to touch or overlap, but not cross; that is, if the interiors of the edges intersect, then they must be parallel. A *k-monotone linear tree* is a tree linkage T with all vertices and all edges of T lying on a horizontal line such that every vertical line intersects the vertices or edges of T at most k times.

A *motion* for a linkage L is a continuous deformation of L on the plane during which no edge crossing is allowed and all edges of L should remain *rigid*, i.e. the interior of each edge is kept straight and each edge length is also preserved. Moreover, a motion is composed of single or simultaneous *moves*, each of which is a continuous monotonic change of an angle between two edges incident to some vertex. A *configuration* of a linkage L is a state of L which can be reached via a series of motions. A linear tree linkage T can be *straightened* if there exists a series of motions to reconfigure any initial configuration of T to the linear configuration with all root-to-leaf paths going to the right (called the *canonical configuration*); otherwise, T cannot be straightened and is thus called *locked*. A fundamental problem we concern is whether there exists a series of motions to reconfigure linkage T between any two linear configurations. If we can show that T can be straightened, then it is clear that this problem has a feasible solution.

We now present the related work for linear linkages in two dimensions in the following. First, Abbott et al. [1] showed that a linear chain can always be reconfigured between any two configurations. Later, Ballinger et al. [3] presented a locked linear tree of monotonicity 7 and a locked equilateral tree. Thus we can see that both linear trees and equilateral trees can lock. However, Abel et al. [2] showed that a linear equilateral tree can always be straightened in polynomial moves and time.

Moreover, we survey the related work for monotone linkages in two dimensions. First, Biedl et al. [4] showed that a monotone polygon can always be convexified in $O(n^2)$ moves and time. Then, Kusakari et al. [8] showed that a monotone tree can always be straightened in $O(n)$ moves and $O(n \log n)$, and Kusakari [7] showed that radial monotone trees can lock. Moreover, Poon [9] showed that a 2-monotone orthogonal tree can always be straightened in $O(n^2)$ moves and time, and Ballinger et al. [3] showed that 3-monotone orthogonal trees can lock. Their results thus showed that the minimum monotonicity of locked orthogonal trees is 3. In addition, Ballinger et al. [3]

also provided a locked linear tree of monotonicity 7. However, it remained open the questions of the straightenability of k-monotone linear trees and of determining the minimum monotonicity of locked linear trees. These motivate our work in this paper. In this paper, we not only show that a k-monotone tree for $k \leq 4$ can always be straightened, but also determine that the minimum monotonicity of locked tree linkages is 5. Hence, our results completely settle such a question.

2 Preliminaries

In this section, we define technical terms relating to the structure of k-monotone linear trees and unfolding operations. Let $T = (V, E)$ be a k-monotone linear tree (see tree T in Figure 1), where the clockwise cyclic ordering of the incident edges around each vertex in T is given. In most figures of this paper, overlapping edges of linear trees are drawn as slightly separated straight line segments or curved line segments, in order for showing the combinatorial structure of the overlapping edges.

Backbone and Main Subtrees. The *backbone* K of tree T is the chain between a vertex with the smallest x-coordinate in T (called the *left-end* of K or the *root* of T) and a vertex with the largest x-coordinate in T (called the *right-end* of K). See backbone $K = C[s, t]$ in Figure 1, where $C[s, t]$ denotes the chain between vertices s and t. If we remove all edges of K from T, then we call the remaining components that are not isolated vertices the *main subtrees* of T. See subtree T_i rooted at vertex u in Figure 1 for illustration.

Now, the components in a main subtree T_i in T are defined as follows. The *root* of T_i (see vertex u in Figure 1) is the common vertex of T_i and backbone K. Let C be the chain between the left-end and the right-end of T_i (see chain $C[p, q]$ in Figure 1). The *(main) stem* of T_i (see chain $C[u, v]$ in Figure 1) is the shortest chain connecting root u and C. The *branching vertex* v of T_i (see vertex v in Figure 1) is the common vertex of the stem of T_i and C. Moreover, the *left (main) branch* of T_i (see chain $C[v, p]$ in Figure 1) is the chain between v and the left-end of T_i, and the *right (main) branch* of T_i (see chain $C[v, q]$ in Figure 1) is the chain between v and the right-end of T_i.

Covering Relations. Next, we define the *covering relations* for straight chains in tree T. Let C be a straight chain in T above K, and let $\mathcal{R}(C)$ be the set of all rays starting from a point on C shooting in the upward direction. Straight chain C is *covered* if some rays in $\mathcal{R}(C)$ topologically intersect some non-vertical edges of T; otherwise, C is *uncovered*. Notice that we use the term "topologically" to mean that although two objects occupy the same geometrical position, under the self-touching assumption, their positions can still be differentiated. Moreover, if all rays in $\mathcal{R}(C)$ topologically intersect some non-vertical edges of T, then C is called *fully covered*. If some rays in $\mathcal{R}(C)$ topologically intersect some non-vertical edges of T, but some do not, then C is called *partly covered*.

Let $C_1, \ldots, C_q$ for $q \geq 3$ be a set of edge-disjoint straight chains in T. For every i, $1 \leq i \leq q-1$, if C_i is covered by C_{i+1}, then C_1 is *indirectly covered* by C_q. Note that C_1 can be covered and indirectly covered by C_q at the same time. For the covering relations over straight chains in T below K, we define them in a symmetric fashion as above.

Zigzags. A *regular zigzag* in tree T (see chain $xijk$ in Figure 1) is a Z-shape chain, and a *reverse zigzag* in tree T (see chain $jk\ell q$ in Figure 1) is a mirrored Z-shape chain. The three edge-disjoint straight chains of Z are called the *upper*, *middle*, and *lower chains* of Z in top-to-bottom order. Here we define a *zigzag* in a more strict fashion, with considering two more properties, i.e., the upper and lower chains are maximal straight chains in tree T, and they should be at least as long as the middle chain. The *left* and *right corners* of Z are the left- and right-ends of the middle chain of Z, respectively. The *left* and *right chains* of Z are the upper and lower chains of Z that connect to the right and left corners of Z, respectively.

Binding, Verticalizing, Horizontalizing, and Straightening Operations. To *bind* overlapping edges lying on a line is to glue them together to form a combined edge. To *upward-* and *downward-verticalize* a straight chain $C = C[u, v]$, where vertex v is a leaf, around vertex u to lie in the upward or downward direction is to rotate C around leaf u while keeping C rigid, until C points to the upward or downward direction, respectively. Clearly, the operations of *upward-* and *downward-horizontaling* a straight chain C can also be defined in a similar fashion. In addition, to *straighten* a subtree T_i is to reconfigure T_i such that all root-to-leaf chains in T_i become straight chains that run along a specific direction.

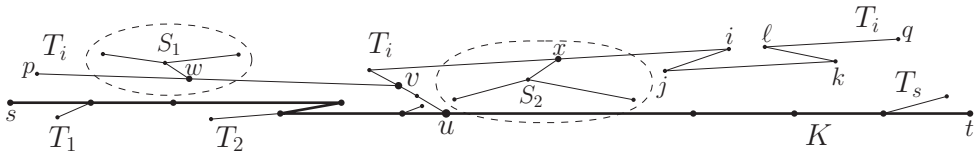

Figure 1: A 4-monotone linear tree T with zigzagged backbone K (bold edges).

3 3-monotone linear trees

In this section, we present the algorithm Unfold3MonoTree consisting of four steps for straightening a 3-monotone linear tree T with backbone K. In Steps 1 to 3, we straighten all main subtrees above K to lie in the upward direction. Then in Step 4, we first straighten all main subtrees below K to lie in the downward direction in a similar fashion as in Steps 1 to 3. Then we pull straight all zigzags of K and we rightward-horizontalize all main subtrees in T in sequence. This completes the overview of this algorithm. We now briefly describe the first three steps as follows. More detailed steps are omitted due to lack of space.

A *hair* is a straight chain rooted on K, a branch, or a stem, extending to a leaf, and without additional edges incident to its internal vertices. Note that when we say to verticalize or horizontalize a hair, we use the convention that we perform the corresponding operation around its non-leaf endpoint. In Step 1, we manipulate all hairs rooted on each main branch B above K. That is, we first upward-verticalize all hairs rooted on the upper side of each branch B, and then we bind all hairs rooted on the lower side of each branch B to B. In Step 2, we first upward-verticalize all the main branches above K that are not fully covered by any other main branch, and then we upward-verticalize all the remaining fully-covered main branches above K. As a result, all main branches above K are upward-verticalized. Besides, we can see that the remaining parts that are not upward-verticalized are all main stems above K and all the bound inner hairs rooted on the main branches above K. Thus in Step 3, we first upward-verticalize all main stems above K, and then we release and upward-verticalize all the bound inner hairs rooted on the main branches above K. As a result, all main subtrees above K are straightened to lie in the upward direction.

We note that in the whole process, no edge crossing can occur. Since we obtain the verticalizing ordering according the tree structure, this algorithm runs in linear time. Lastly, we obtain the following theorem.

Theorem 1 *A 3-monotone linear tree can be straightened in $O(n)$ moves and $O(n)$ time.*

4 4-monotone linear trees

In this section, we consider the straightening of 4-monotone linear trees. In Section 4.1, we solve a specific case in which the backbones are straight. In Section 4.2, we extend the result to solve the general case.

4.1 4-monotone linear trees with straight backbone

Now, we consider the straightening of a 4-monotone linear tree T with straight backbone K. A secondary subtree in a main subtree T_i is called *inner* if it lies wholly between backbone K and the union of both branches of T_i (see subtree S_2 in Figure 1); otherwise, it is called *outer* (see subtree S_1 in Figure 1). Clearly, applying Unfold3MonoTree to tree T is not enough to solve this problem since inner secondary subtrees can entangle with main subtrees, which makes this problem nontrivial. With a careful design of basic operations, we come up with a sophisticated algorithm Unfold4MonoTreeSB consisting of six steps to straighten the given tree T. The key idea of this algorithm is to find and verticalize *uncovered neat backward-hair brushes* (defined below; see Figure 2(a)) in an iterative fashion until all main subtrees are verticalized. We can realize this idea due to the fact that the verticalization ordering for the *sticks* of these brushes (defined below) we select follows the verticalization orders (i) and (ii), mentioned in the following Step 4. After all main subtrees are verticalized, it is not hard to see that we can proceed to straighten all verticalized main subtrees.

Definition 2 *A* neat backward-hair brush R *is a straight chain, called the* stick $K(R)$ *of R (see $C[u,v]$ in Figure 2(a)), attached with hairs $H(R)$*

(see hairs f_1, f_2, f_3 in Figure 2(a)), in a main subtree of tree T such that the following three properties are satisfied:

- ***(One-end-free Stick)*** *Stick $K(R)$ is* one-end-free *if by treating as being invisible all hairs in $H(R)$, all verticalized subtrees rooted on $K(R)$, and all the edges rooted on the upper side of $K(R)$ that cover $K(R)$, then $K(R)$ is a hair with its leaf v, called the* free-end *of $K(R)$. Moreover, the* pivot *of $K(R)$ is its other endpoint.*
- ***(Backward Hairs)*** *All hairs in $H(R)$ lie between stick $K(R)$ and backbone K and point backwards with respect to the direction of $K(R)$. Thus hairs $H(R)$ are called the* backward hairs *of R.*
- ***(Fully-covered Hairs)*** *All hairs in $H(R)$ are fully covered by stick $K(R)$.*

Moreover, a brush R is *uncovered* if its stick $K(R)$ is uncovered. Note that when we say to verticalize or horizontalize a brush R, we use the convention that we perform the corresponding operation around the pivot of its stick $K(R)$.

The six steps of Unfold4MonoTreeSB is described as follows. In Step 1, we first compute a *trapezoidal map* $M(T)$ of tree T (see [6] for the definition) for obtaining covering relations, which will be used in the following steps. Then we modify all main subtrees in T via five kinds of binding operations (the corresponding properties we aim to obtain are omitted due to lack of space) in order to obtain a simpler linear tree structure T'. See Figure 3(b) for illustration.

In Step 2, we first compute the *core tree* T^* by removing from T' all right inner secondary branches corresponding to all left main branches in T', and all left inner secondary branches corresponding to all right main branches in T'. See Figure 3(c). Note that the removed branches will serve as backward hairs. In an onion-peeling order, we perform the *brush stick selection* on each main subtree T_i of T^* to select a set of *candidate sticks* $\mathcal{C}$ from T_i. See Figure 3(d). More precisely, we recursively extract from the current subtree T_i' of T_i a maximal straight chain C with one endpoint being the vertex with minimum *distance to root* $d_r(\cdot)$ such that C does not cover any edge in T_i', until T_i' becomes empty. The *distance to root* $d_r(C)$ of a chain $C[u, v]$ in T_i as the minimum value of $d_r(u)$ and $d_r(v)$. Then we claim the following lemma, in order for showing the key lemma of this algorithm, namely Lemma 4.

Lemma 3 (Covering of selected sticks) *Let C, C' be two sticks in $\mathcal{C}$ intersecting a root-to-leaf chain in a main subtree T_i of tree T^*. If $d_r(C) < d_r(C')$, then C' covers or indirectly covers C.*

In Step 3, for each brush R whose stick is in $\mathcal{C}$, we compute its backward hairs $H(R)$ and *interfering hairs* $I(R) \subset \mathcal{C}$ (see hairs g_1, g_2, g_3 in Figure 2(a)), whose hairs are covered by $K(R)$ and cover or indirectly cover some backward hairs in $H(R)$. Thus $\mathcal{C}$ can be divided into two categories, say $\mathcal{C}_s$ (sticks that do not serve as interfering hairs; called *final sticks*) and $\mathcal{C}_h$ (sticks that serve as interfering hairs). Then we obtain *final brushes* $\mathcal{R}$, consisting of all brushes whose sticks are in $\mathcal{C}_s$, which can be proven to be all neat backward-hair ones, whose proof is omitted due to lack of space.

Lemma 4 (Neat backward-hair brush lemma) *All brushes $R \in \mathcal{R}$ are neat backward-hair ones in T', and all sticks in $I(R)$ are fully covered by $K(R)$.*

Moreover, for every brush $R \in \mathcal{R}$, we assign each hair in $H(R)$ and $I(R)$ to point to the non-leaf endpoint of its corresponding hair in $I(R)$ and $H(R)$, respectively. See dotted arrows in Figure 2(b).

In Step 4, we first compute a linear verticalization ordering Ω for all brushes $R \in \mathcal{R}$, which are all neat backward-hair ones, such that the following two verticalization orders hold:

- **Verticalization order (i), based on covering relations over $\mathcal{C}_s$.** If stick $C \in \mathcal{C}_s$ is covered or indirectly covered by another stick $C' \in \mathcal{C}_s$, then C' should be verticalized earlier than C.
- **Verticalization order (ii), based on tree-structure relations over $\mathcal{C}_s$.** For two sticks $C, C' \in \mathcal{C}_s$ intersecting a root-to-leaf chain of a main subtree T_i, if $d_r(C) < d_r(C')$, then C' should be verticalized earlier than C.

According to Lemma 4 and this ordering, we can upward-verticalize on all brushes $R \in \mathcal{R}$ and all hairs in $H(R) \cup I(R)$, as shown in Figure 2, without any edge crossing.

Next, in Step 5, we first release all the bound structures of T and then perform an intuitive but complicated procedure, called the *stretching and straightening process* consisting of ten substeps, to straighten all verticalized main subtrees above K. Lastly, we straighten all main subtrees below K

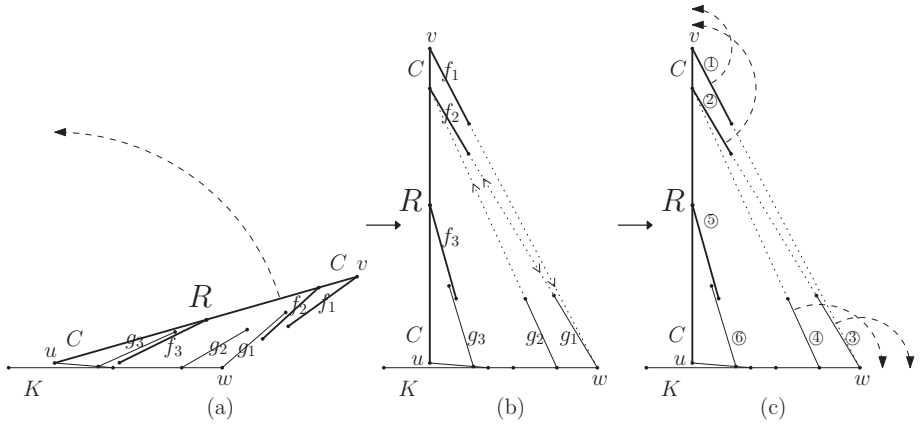

Figure 2: Upward-verticalizating an uncovered neat backward-hair brush R (bold edges) whose stick is chain $C = C[u, v]$ with the backward hairs $H(R) = \{f_1, f_2, f_3\}$ and the interfering hairs $I(R) = \{g_1, g_2, g_3\}$. (a) Initial configuration of brush R. (b) The configuration when R is upward-verticalized. During the motion from (a) to (b), all hairs in $H(R) \cup I(R)$ are kept pointing to their target points (dotted arrows). (c) Circled numbers represent the sequential peeling operations.

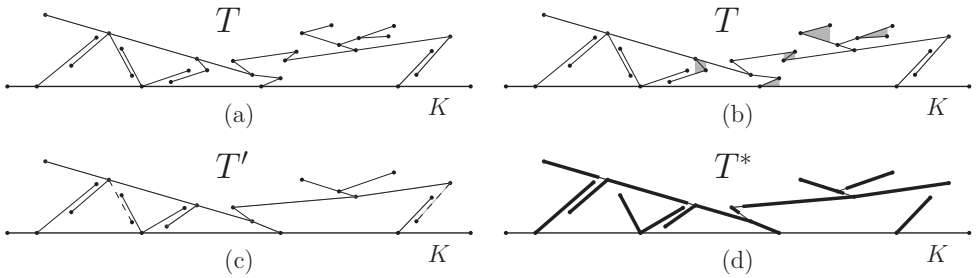

Figure 3: Steps 1 and 2 of UNFOLD4MONOTREESB. (a) Initial tree T. (b) Applying the five kinds of binding operations (shaded regions) to tree T to obtain T'. (c) Removing some inner secondary branches (dashed edges) from tree T' to obtain core tree T^*. (d) Selecting candidate sticks (bold chains) from all main subtrees of core tree T^*.

in a similar fashion as above. This completes the algorithm. More detailed steps and correctness proofs are omitted due to lack of space. Since computing trapezoidal map $M(T)$ takes $O(n \log n)$ time, we thus obtain the following lemma.

Lemma 5 *A 4-monotone linear tree with straight backbone can be straightened in $O(n)$ moves and $O(n \log n)$ time.*

4.2 4-monotone linear trees with zigzagged backbone

In this subsection, we present algorithm UNFOLD4MONOTREE consisting of four steps to straighten a general 4-monotone linear tree T with backbone K. Without loss of generality, we assume that K contains zigzags. As tree T has monotonicity 4, edges can pass through the openings of backbone zigzags, so that when we are going to move a such edge, we need to *open* the corresponding opening beforehand and move related edges above and below K simultaneously.

Now we sketch UNFOLD4MONOTREE as follows. This algorithm uses the procedure in UNFOLD4MONOTREESB as a subroutine, and we straighten the zigzags in K incrementally by considering covering relations as follows. We search for the first *openable zigzag* Z in K from left to right such that its middle chain, its right chain and the main subtrees rooted on its right chain can be bound, and then be verticalized without any edge crossing. Then we apply an *opening process* to Z and all the other unstraightened zigzags Z' in K on its left one by one from right to left (see Figures 4). After that, we further apply an *straightening process* to straighten these zigzags one by one from left to right (see Figures 5). Thus by repeating the above procedure, we eventually can

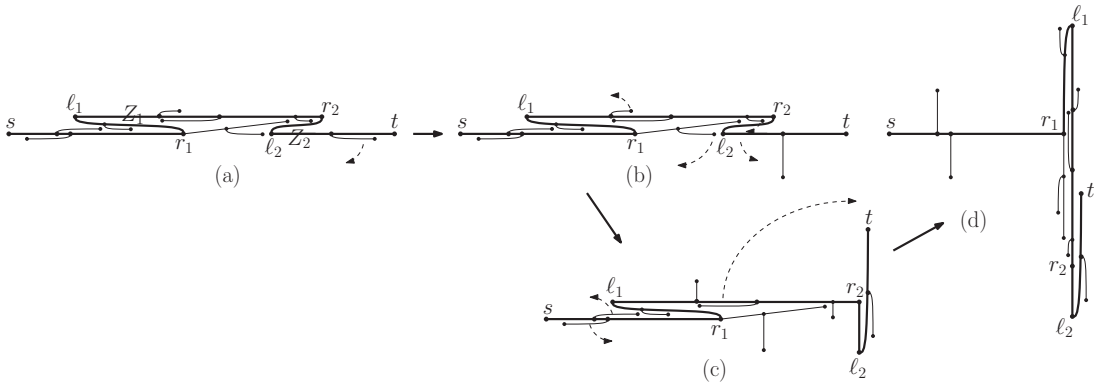

Figure 4: The opening process for all zigzags in $\mathcal{Z}' = \{Z_1, Z_2\}$.

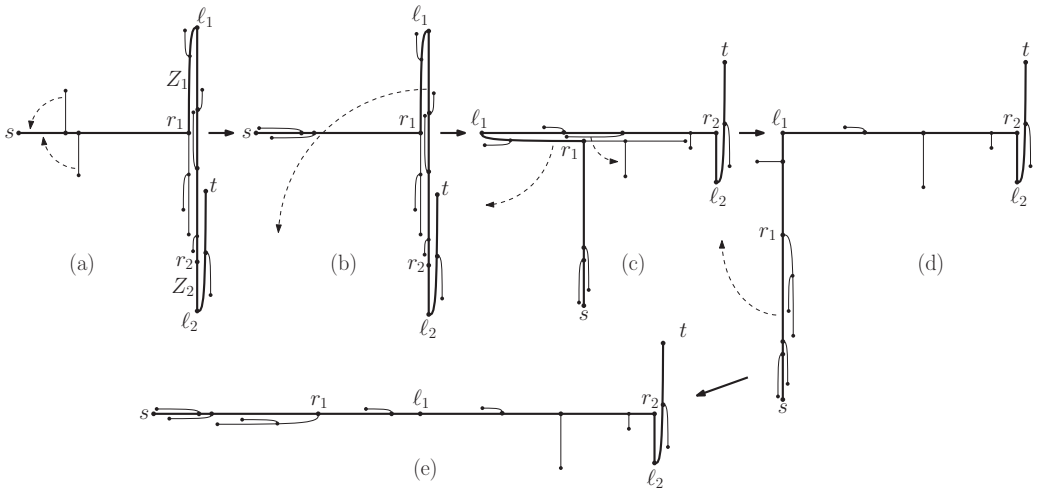

Figure 5: The straightening process for zigzag $Z_1 = C[s, r_2] \in \mathcal{Z}'$. (a) is the continuation from Figure 4(d).

verticalize all main subtrees and also straighten all zigzags of K. More detailed steps and correctness proofs are omitted due to lack of space. Since computing trapezoidal map $M(T)$ and some stick sets, used in opening and straightening processes, both run in $O(n \log n)$ time, we thus obtain the following theorem.

Theorem 6 *A* 4*-monotone linear tree can be straightened in* $O(n)$ *moves and* $O(n \log n)$ *time.*

5 Locked linear trees of monotonicity more than 4

In this section, we present two new locked linear trees of monotonicity more than 4. To show the lockedness, we make use of the following lemma as the theoretical basis.

Lemma 7 ([5]) *If a self-touching configuration is rigid, then it is strongly locked.*

At the beginning, we present a 5-monotone linear tree of 14 edges, whose self-touching configuration is as shown in Figure 6. Clearly, the linear geometry of this tree is a straight horizontal line with five distinct vertices. We then claim that the tree is rigid and thus, by Lemma 7, strongly locked, whose proof is omitted due to lack of space.

Theorem 8 *The* 5*-monotone linear tree of* 14 *edges in Figure 6 is strongly locked.*

Furthermore, we observe that the backbone of the above locked linear tree contains only one zigzag, and the smallest monotonicity ever known for a locked linear tree of 8 edges is 7. Thus we present a 6-monotone linear tree of 8 edges with straight backbone (see Figure 7), and we claim the tree is rigid and thus, by Lemma 7, strongly locked, whose proof is also omitted due to lack of space.

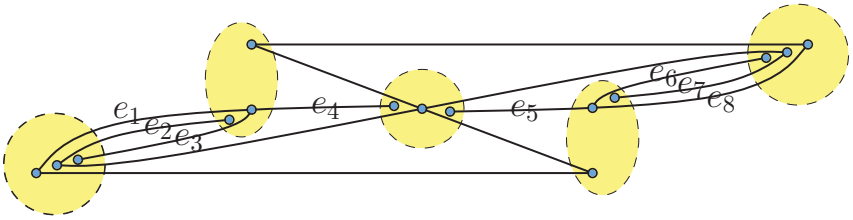

Figure 6: A locked 5-monotone linear tree of 14 edges. Vertices surrounded by dashed circles are tighter than drawn.

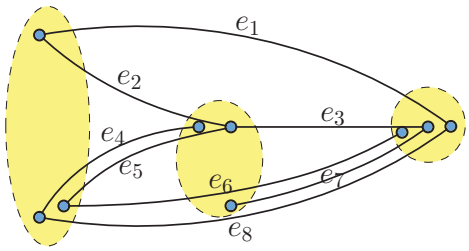

Figure 7: A locked 6-monotone linear trees with straight backbone of 8 edges. Vertices surrounded by dashed circles are tighter than drawn.

Theorem 9 *The 6-monotone linear tree of 8 edges in Figure 7 is strongly locked.*

6 Conclusions

In this paper, we first show that a 3-monotone linear tree can always be straightened in $O(n)$ moves and $O(n)$ time, and a 4-monotone linear tree can always be straightened in $O(n)$ moves and $O(n \log n)$ time. Then we show that 5-monotone linear trees can lock by providing a locked 5-monotone linear tree of 14 edges. Thus we determine that the minimum monotonicity of locked linear trees is 5. Furthermore, we provide a locked 6-monotone linear tree of 8 edges with straight backbone. Lastly, we leave open the question whether a 5-monotone linear tree with straight backbone can always be straightened.

References

[1] T. G. Abbott, E. D. Demaine, and B. Gassend. A generalized carpenter's rule theorem for self-touching linkages. arXiv:0901.1322, 2009.

[2] Z. Abel, E. D. Demaine, M. L. Demaine, S. Eisenstat, J. Lynch, T. B. Schardl, and I. Shapiro-Ellowitz. Folding equilateral plane graphs. *International Journal of Computational Geometry & Applications*, 23:75–92, 2013.

[3] B. Ballinger, D. Charlton, E. D. Demaine, M. L. Demaine, J. Iacono, C.-H. Liu, and S.-H. Poon. Minimal locked trees. In *Algorithms and Data Structures*, pages 61–73, 2009.

[4] T. Biedl, E. D. Demaine, S. Lazard, S. Robbins, and M. Soss. Convexifying monotone polygons. In *Algorithms and Computation*, pages 415–424, 1999.

[5] R. Connelly, E. D. Demaine, and G. Rote. Infinitesimally locked self-touching linkages with applications to locked trees. In *Physical Knots: Knotting, Linking, and Folding of Geometric Objects in R^3*, pages 287–311. AMS, 2002.

[6] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 3rd edition, 2008.

[7] Y. Kusakari. On reconfiguring radial trees. *IEICE Transactions*, 89-A:1207–1214, 2006.

[8] Y. Kusakari, M. Sato, and T. Nishizeki. Planar reconfiguration of monotone trees. *IEICE Transactions*, E85-A:938–943, 2002.

[9] S.-H. Poon. On unfolding 3D lattice polygons and 2D orthogonal trees. In *Computing and Combinatorics*, pages 374–384, 2008.