Minimum projective linearizations of trees in linear time

Lluís Alemany-Puig\textsuperscript{a,}\textsuperscript{*}, Juan Luis Esteban\textsuperscript{b}, Ramon Ferrer-i-Cancho\textsuperscript{a}

\textsuperscript{a}Complexity and Quantitative Linguistics Lab, LARCA Research Group, Computer Science Department, Universitat Politècnica de Catalunya, Jordi Girona 1-3, 08034, Barcelona, Spain
\textsuperscript{b}Computer Science Department, Universitat Politècnica de Catalunya, Jordi Girona 1-3, 08034, Barcelona, Spain

Abstract

The Minimum Linear Arrangement problem (MLA) consists of finding a mapping \(\pi\) from vertices of a graph to distinct integers that minimizes \(\sum_{(u,v) \in E} |\pi(u) - \pi(v)|\). In that setting, vertices are often assumed to lie on a horizontal line and edges are drawn as semicircles above said line. For trees, various algorithms are available to solve the problem in polynomial time in \(n = |V|\). There exist variants of the MLA in which the arrangements are constrained. Iordanskii, and later Hochberg and Stallmann (HS), put forward \(O(n)\)-time algorithms that solve the problem when arrangements are constrained to be planar (also known as one-page book embeddings). We also consider linear arrangements of rooted trees that are constrained to be projective (planar embeddings where the root is not covered by any edge). Gildea and Temperley (GT) sketched an algorithm for projective arrangements which they claimed runs in \(O(n)\) but did not provide any justification of its cost. In contrast, Park and Levy claimed that GT’s algorithm runs in \(O(n \log d_{\text{max}})\) where \(d_{\text{max}}\) is the maximum degree but did not provide sufficient detail. Here we correct an error in HS’s algorithm for the planar case, show its relationship with the projective case, and derive simple algorithms for the projective and planar cases that run without a doubt in \(O(n)\) time.

Keywords: Linear arrangements, Minimum Linear Arrangement Problem, Projectivity, Planarity, One-page embeddings

1. Introduction

A linear arrangement \(\pi\) of a graph \(G = (V, E)\) is a linear ordering of its vertices (it can also be seen as a permutation), i.e., vertices lie on a horizontal line. In such arrangement, the distance \(d(u, v)\) between two vertices \(u, v\) can be defined as \(d(u, v) = |\pi(u) - \pi(v)|\) where \(\pi\) maps the \(n\) vertices to the \(n\) distinct integers in \([1, n]\). The minimum linear arrangement problem (MLA) consists of finding a \(\pi\) that minimizes the cost \(D = \sum_{(u,v) \in E} d(u,v)\) \([4, 15]\). In arbitrary graphs, the problem is NP-hard \([4]\). For trees, various algorithms are available to solve the problem in polynomial time \([6, 16, 15]\). Goldberg and Klipker \([6]\) devised an \(O(n^3)\) algorithm. Later, Shiloach \([16]\) contributed with an \(O(n^{2.2})\) algorithm. Finally, Chung \([15]\) contributed with two algorithms running in \(O(n^2)\) time and \(O(n^\lambda)\) time, respectively, where \(\lambda\) is any real number satisfying \(\lambda > \log 3/\log 2\). The latter algorithm is the best algorithm known.

There exist several variants of the MLA problem; two of them are the planar and the projective variants. In the planar variant, namely the MLA problem under the planarity constraint, the placement of the vertices of a free tree is constrained so that there are no edge crossings. These arrangements are known as planar arrangements \([12]\), and also one-page book embeddings \([1]\). Two undirected edges of a graph \(\{s, t\}, \{u, v\} \in E\) cross if \(\pi(s) < \pi(u) < \pi(t) < \pi(v)\) when, without loss of generality, \(\pi(s) < \pi(t), \pi(u) < \pi(v)\) and \(\pi(s) < \pi(u)\). To the best of our knowledge, the first \(O(n)\) algorithm was put forward by Iordanskii \([10]\). Sixteen years later, Hochberg and Stallmann (HS) put forward another \(O(n)\)-time algorithm \([9]\). However, their algorithm contains an error which is corrected in this paper.

In the projective variant, namely the MLA problem under the projectivity constraint, a rooted tree is arranged so that there are no edge crossings (i.e., the arrangement is planar) and the root is not covered. These arrangements are known as projective \([12, 13]\). A vertex \(w\) is covered by an edge \(\{u, v\}\) if \(\pi(u) < \pi(w) < \pi(v)\) when, without loss of generality, \(\pi(u) < \pi(v)\). Fig. 1(a) shows a projective arrangement while Fig. 1(b) shows an arrangement that is projective if we take vertex 2 as the root but not if we take vertex 1 as the root. Gildea and Temperley (GT) \([5]\) sketched an algorithm to solve this variant. The tree shown in Fig. 1 is the smallest tree for which there is a vertex that, when chosen as the
root, makes the minimum cost of the projective case be greater than that of the planar case (there are no other 6-vertex trees where that happens). While GT claimed that their sketch runs in $O(n)$ [5, p. 2], Park and Levy (PL) argued that it runs in time $O(n \log d_{\max})$, where $d_{\max}$ is the maximum degree. However, PL did not give enough detail to support their conclusion [14]. In this article, we show that this is an overestimation of the actual complexity: the problem can be actually solved in $O(n)$ time.

The remainder of the article is organized as follows. Section 2 introduces the notation and reviews HS’s algorithm. Section 3 corrects and completes HS’s algorithm [9]. The error is located in a recursive subprocedure (embed_branch) of HS’s algorithm. In Section 4, we present two detailed $O(n)$-time algorithms for the projective case that stem from HS’s algorithm. HS’s algorithm already contained a ‘subalgorithm’ for solving the projective case although the authors did not identify it as such in their article [9]. Indeed, their algorithm can be reinterpreted as consisting of two main steps: finding a centroidal vertex (as in Iordanskii’s algorithm [10]) and then solving the projective case for the input tree rooted at that vertex. Hence the first algorithm for the projective case is obtained extracting the relevant part from HS’s original algorithm, completing and simplifying it and, critically, using the correction indicated in Section 3. Our second algorithm for the projective case is a re-engineered version based on intervals that results into a more compact, clearer and readable version from HS’s original algorithm.

Now we summarize the core ideas and tools derived by HS [9]. Firstly, using Lemmas 6, 11 in [9], it is easy to see that an optimal projective arrangement of $T^r$ is obtained by arranging the immediate subtrees of $T^r$ inwards, decreasingly by size and on alternating sides, namely $T^r_1, T^r_3, \ldots, r, \ldots, T^r_4, T^r_2$ or $T^r_2, T^r_4, \ldots, r, \ldots, T^r_3, T^r_1$. Immediate subtrees of $T^r$ can be arranged in any of the two orders, whereas immediate subtrees of $T^r_u$, $u \neq r$ to have to be placed according to the side in which $u$ is placed with respect to $p(u)$: if $u$ is placed to the left of $p(u)$’s left then the optimal order is $T^r_{u,1}, T^r_{u,3}, \ldots, u, \ldots, T^r_{u,4}, T^r_{u,2}$ (Fig. 2(a)), and if $u$ is placed to the right of $p(u)$’s right the optimal order is $T^r_{u,2}, T^r_{u,4}, \ldots, u, \ldots, T^r_{u,3}, T^r_{u,1}$ (Fig. 2(b)). Notice that the root is not covered in any of these planar arrangements, as required by the projectivity constraint [12, 13].

Secondly [9, Theorem 12], an optimal planar arrangement of a free tree $T$ is obtained when $T$ is rooted at one of its centroidal vertices. Therefore, an optimal planar arrangement of a free tree $T$ is an optimal projective arrangement of $T^c$, where $c$ denotes one of the (possible two) centroidal vertices of $T$. For the calculation of a centroidal vertex, HS defined $s(u, v)$, which we call directional size of subtrees. The directional size $s(u, v)$ in a free tree $T$, for $(u, v) \in E(T)$, is the size of $T^c_{uv}$ (Fig. 3). Notice that $s(v, u) + s(u, v) = n$. They also outlined a way of calculating all of the $s(u, v)$ in $O(n)$ time [9, Section 6], but did not provide any pseudocode.

---

\[ a) \quad b) \]

---

Figure 1: Two different linear arrangements of the same free tree $T$. a) A minimum projective arrangement of $T$ rooted at 1 with cost $D = 7$; the circled dot denotes the root. b) A minimum planar arrangement of $T$ with cost $D = 6$ under the planarity constraint; the squared dot denotes its (only) centroidal vertex.
Figure 2: a,b) Optimal arrangements of $T^v_u$ according to the relative position of $v$ with respect to $v$'s parent. c) Depiction of the directed edges $(p(u), u), (u, v) \in E$ in an optimal projective arrangement, divided into the anchor (the part of the edge $(u, v)$ to the left of the vertical line), and the coanchor (the part of the edge $(u, v)$ to the right). The length of the anchor of edge $(p(u), u)$ is the sum $n_j$ for even $j \in [2, k]$.

Figure 3: a) A free tree with $s(u, v) = 7$, $s(v, u) = 3$ and $s(v, w) = s(u, v) = 5$. b) The free tree in a) rooted at $v$; $|V(T^v_u)| = s(v, u)$. Borrowed from [9, Fig. 7].

here we provide it in Algorithm 2.1. Using the $s(u, v)$ for all edges in $T$, we can construct a sorted adjacency list of the tree which we denote as $L$, with the pseudocode given in Algorithm 2.2, and with it we calculate one of the centroidal vertices. Algorithm 2.3 reports the pseudocode for the calculation of the centroidal vertex. All algorithms have $O(n)$-time and $O(n)$-space complexity.

We also need to consider the rooting of the list $L$ with respect to a given vertex $w$, denoted as $L^w$. This operation is called $\text{Root}_\text{-}\text{List} (L, w)$ in the pseudocode.

Algorithm 2.1: Calculation of directional sizes for free trees. Cost $O(n)$ time, $O(n)$ space.

\begin{algorithm}
\begin{algorithmic}
\Function{compute_s_ft}{$T$}\Comment{Algorithm 2.1}
\Statex \textbf{In:} $T$ free tree.
\Statex \textbf{Out:} $S = \{(u, v, s(u, v)), (v, u, s(v, u)) \mid (u, v) \in E\}$. \Comment{$T$ is a free tree, $s(u, v)$ denotes size of edge $(u, v)$.}
\State $S' \leftarrow \emptyset$
\State $u_\ast \leftarrow$ choose an arbitrary vertex
\For{$v \in \Gamma_u \ast$} \Comment{$\Gamma_u \ast$ is the list of adjacency edges for $v$. See Algorithm 2.2.}
\State $(\_ , S')' \leftarrow \text{comp_s_ft_rec}(T, (u_\ast, v))$
\State $S \leftarrow S \cup S'$
\EndFor
\State \Return $S$
\EndFunction
\end{algorithmic}
\end{algorithm}

Algorithm 2.2: Calculation of the sorted adjacency list for free trees. Cost $O(n)$ time, $O(n)$ space.

\begin{algorithm}
\begin{algorithmic}
\Function{sorted_adjacency_list_ft}{$T$}\Comment{Algorithm 2.1}
\Statex \textbf{In:} $T$ free tree.
\Statex \textbf{Out:} $L$, the decreasingly-sorted adjacency list of $T$.
\State \Comment{// Algorithm 2.1}
\State $S \leftarrow \text{compute_s_ft}(T)$ \Comment{Algorithm 2.1}
\State Sort the triples $(u, v, s)$ in $S$ decreasingly by $s$ using counting sort \[3\]
\State $L \leftarrow \emptyset^n$
\For{$(u, v, s) \in S$} \Comment{// Append at end in $O(1)$}
\State $L[u] \leftarrow L[u] \cup (v, s)$
\EndFor
\State \Return $L$
\EndFunction
\end{algorithmic}
\end{algorithm}
Algorithm 2.3: Calculation of a centroidal vertex of a free tree. Cost $O(n)$ time, $O(n)$ space.

1 Function FIND_CENTROIDAL_VERTEX($T$) is
2   In: $T$ free tree.
3   Out: A centroidal vertex of $T$.
4     // Algorithm 2.2
5     $L$ ← SORTED_ADJACENCY_LIST_FT($T$)
6     return FIND_CENTROIDAL_VERTEX($T$, $L$)

Function FIND_CENTROIDAL_VERTEX($T$, $L$) is
7     In: $T$ free tree, $L$ sorted adjacency list of $T$.
8     Out: A centroidal vertex of $T$.
9     $u$ ← choose an arbitrary vertex
10    while true do
11       // $O(1)$ time since $L[u]$ is sorted
12       ($v$, $s$) ← largest entry in $L[u]$
13       if $s > n/2$ then $u ← v$
14       else return $u$

3. Minimum planar linear arrangements

In [9], HS present an easy-to-understand algorithm to calculate a minimum planar linear arrangement for any free tree in linear time. The idea behind the algorithm was presented in Section 2. The implementation has two procedures, embed and embed_branch, that perform a series of actions in the following order:

- Procedure embed gets one centroidal vertex, $c$, uses it as a root and orders its immediate subtrees by size.

- Procedure embed puts immediate subtrees with an even index in one side of the arrangement and immediate subtrees with odd index in the other side (the bigger the subtree, the farther away from the centroidal vertex), calling procedure embed_branch for every subtree.

- Procedure embed_branch calculates recursively a displacement of all nodes with respect to the placement of the centroidal vertex (of the whole tree) in the linear arrangement.

- Procedure embed calculates the centroidal vertex’s position (the sum of sizes of trees on the left of the centroidal vertex) and applies the displacement to the rest of the nodes.

From Algorithm 3.1, we can see that vector relPos must contain the displacement of all nodes from the position of the centroidal vertex in the linear arrangement. Note that these are only the last lines of embed. The problem lays in procedure embed_branch, which does not calculate correctly the displacement vector relPos. In Algorithm 3.2, we give a correct version of procedure embed_branch, where changes with respect to HS’s version are marked in red. Lines 6 to 10 are needed to calculate the correct displacement. For a vertex $u \neq c$, variable under_anchor is the number of nodes of $T_u$ between $u$ and $p(u)$. Adding under_anchor to parameter base (line 10), we obtain the correct displacement. There is also a slight modification in the recursive calls (lines 14 and 17) which is the addition of all the parameters needed.

We should note that embed needs to calculate a sorted adjacency list $L$ to calculate a centroidal vertex $c$ for $T_c$ (Algorithm 2.3). However, in order to calculate the arrangement, we need $L$ to be rooted at $c$, then we use $L^c$ (see Section 2, explanation of ROOT_LIST).

In Section 4, we give an even simpler algorithm that can be seen as a different interpretation of HS’s algorithm as it uses the same idea for ordering the subtrees but instead of calculating displacements for nodes it only uses the interval of positions where a subtree must be arranged.

Prior to HS’s work, Iordanskii [10] presented an algorithm to solve the task of minimizing $D$ under the planarity constraint. He devised a different approach to solve the same problem: given a free tree, the algorithm roots the tree at its centroid, and then separates the tree into chains of vertices, which have to be arranged in such a way that a planar arrangement is produced. The proper selection of the chains, coupled with the proper labeling of their vertices, produces a minimum planar arrangement. An outline of the algorithm that is applied on $T^c$ is as follows [11]:

1. Select an arbitrary vertex $v_0$ in the current decomposition subtree (initial tree).

2. Go from vertex $v_0$ along the branches with the greatest number of vertices to some hanging vertex $v_1$.

3. Starting from vertex $v_1$, construct a chain along the branches with the largest number of vertices to some other hanging vertex $v_j$.

4. Assign the highest and lowest numbers to the vertices $v_i$ and $v_j$ from the range allocated for the current decomposition subtree (1 and $n$ for the initial tree).
Algorithm 3.2: EMBED_BRANCH corrected

1 Function EMBED_BRANCH(L, v, base, dir, relPos) is
2   In: (Rooted) sorted adjacency list L for Tc as described in Section 2; v the root of the subtree to be arranged; base the displacement for the starting position of the subtree arrangement; dir whether or not v is to the left or to the right of its parent.
3   Out: relPos contains the displacement from the centroidal vertex of all nodes of the subtree.
4   // the children of v decreasingly sorted by size
5     \( C_v \leftarrow L_v[v] \)
6     \( \text{before} \leftarrow \text{after} \leftarrow 0 \)
7     \( \text{under} \_\text{anchor} \leftarrow 0 \)
8   for \( i = 2 \) to \( |C_v| \) step 2 do
9       \( v_i, n_i \leftarrow C_v[i] \)
10      \( \text{under} \_\text{anchor} \leftarrow \text{under} \_\text{anchor} + n_i \)
11     base \leftarrow base + \text{dir} \ast (\text{under} \_\text{anchor} + 1)
12   for \( i = |C_v| \) downto 1 do
13     \( v_i, n_i \leftarrow C_v[i] \)
14     if \( i \) is even then
15       EMBED_BRANCH(L, v_i, base - \text{dir} \ast \text{before} - \text{dir}, relPos)
16       \( \text{before} \leftarrow \text{before} + n_i \)
17     else
18       EMBED_BRANCH(L, v_i, base + \text{dir} \ast \text{after} + \text{dir}, relPos)
19       \( \text{after} \leftarrow \text{after} + n_i \)
20   relPos[v] \leftarrow base

5. Enumerate monotonically the chain connecting the vertices \( v_j \) and \( v_j \), leaving the corresponding ranges of numbers for each selected decomposition subtree.

6. The procedure recursively repeats until all vertices are numbered.

The algorithm requires \( O(n) \) comparison operations and \( O(n \log n) \) additional memory.

Iordanskii’s approach differs markedly from HS’s algorithm, e.g., using chains instead of anchors, and here we have focused on deriving a couple of algorithms for the projective case that stems from HS’s algorithm for the planar case.

4. Minimum projective linear arrangements

The two algorithms for the projective that are presented in this section have \( O(n) \)-time and \( O(n) \)-space complexity, hence our upper bound for the projective case is tighter than that given by PL [14]. The first algorithm is derived from HS’s for the planar case (Algorithm 4.3). This algorithm is obtained after extracting the relevant part from HS’s original algorithm, adapting it and simplifying procedure embed. The simplifications have to do with reducing the computations that Algorithm 2.1 does, which are not necessary in the projective variant (Algorithms 4.1 and 4.2). Algorithm 4.1 is the simplified version of 2.1 that calculates only the sizes of the subtrees \( T_v \) of \( T^c \) for every vertex \( u \) of \( T^c \); Algorithm 4.2 constructs the rooted sorted adjacency list of a rooted tree \( T^c \) with less calculations than Algorithm 2.2. There is no equivalent to Algorithm 2.3 for rooted trees because we do not need to look for any centroidal vertex. Finally, one has to use the correction of the subprocedure embed_branch Algorithm 3.2. Algorithm 4.3 inherits the \( O(n) \)-time and \( O(n) \)-space complexity from HS’s algorithm.

The second algorithm for the projective case is based on a different approach based on intervals (Algorithms 4.4 and 4.6). Although the pseudocode given can be regarded as a formal interpretation of GT’s sketch [5] its correctness stems largely from the theorems and lemmas given by HS [9] (summarized in Section 2). In Algorithm 4.4 we give the main procedure that includes the call to the embedding recursive procedure, given in Algorithm 4.6, which could be seen as a combination of HS’s methods embed_branch and embed excluding the calculation of one of the centroidal vertices [9].

Algorithm 4.6 calculates the arrangement of the input tree \( T^c \) using intervals of integers \([a, b]\), where \( 1 \leq a \leq b \leq n \), that indicate the first and the last position of the vertices of a subtree in the linear arrangement; an approach based on intervals (but using chains) was considered earlier by Iordanskii [11].
Algorithm 4.1: Calculation of size of subtrees for rooted trees. Cost $O(n)$ time, $O(n)$ space.

1 Function `compute_s_rt(T)` is
2   In: $T$ rooted tree.
3   Out: $S = \{(u, v, s(u, v)) \mid (u, v) \in E\}$.
4   $S \leftarrow \emptyset$
5   for $v \in \Gamma_r$ do
6      $(s', S') \leftarrow \text{comp_s_rt_rec}(T, (r, v))$
7      $S \leftarrow S \cup S'$
8   return $S$

9 Function `comp_s_rt_rec(T, (u, v))` is
10   In: $T$ rooted tree, $(u, v)$ directing edge.
11   Out: $s$ the size of $T_v$ in vertices,
12      $S = \{(u, v, s(u, v)) \mid (u, v) \in E(T_v)\}$.
13   $s \leftarrow 1$
14   // Iterate on the out-neighbours of $v$
15   for $w \in \Gamma_v$ do
16      $(s', S') \leftarrow \text{comp_s_rt_rec}(T, (v, w))$
17      $s \leftarrow s + s'$
18      $S \leftarrow S \cup S'$
19      // Append at end in $O(1)$
20   return $(s, S)$

Algorithm 4.2: Calculation of the sorted adjacency list for rooted trees. Cost $O(n)$ time, $O(n)$ space.

1 Function `sorted_adjacency_list_rt(T)` is
2   In: $T$ rooted tree.
3   Out: $L$ the decreasingly-sorted adjacency list of $T_r$.
4   // Algorithm 4.1
5   $S \leftarrow \text{compute_s_rt}(T)$
6   Sort the triples $(u, v, s)$ in $S$ decreasingly by $s$ using counting sort [3]
7   $L \leftarrow \{0\}^n$
8   for $(u, v, s) \in S$ do
9      // Append at end in $O(1)$
10     $L[u] \leftarrow L[u] \cup (v, s)$
11   return $L$

Algorithm 4.3: Adaptation of HS’s main procedure for the projective case.

1 Function `HS_PROJECTIVE(T)` is
2   In: $T$ rooted tree at $r$.
3   Out: An optimal projective arrangement $\pi$.
4   // Steps 1 and 3 of HS’s algorithm
5   // Algorithm 4.2
6   $L \leftarrow \text{sorted_adjacency_list_rt}(T)$
7   $\text{relPos} \leftarrow \{0\}^n$
8   $\text{leftSum} \leftarrow \text{rightSum} \leftarrow 0$
9   for $i = k \text{ downto } 1$ do
10      if $i$ is even then
11         // Algorithm 3.2
12         $\text{embed_branch}(L, v_i, \text{rightSum}, 1, \text{relPos})$
13         $\text{rightSum} \leftarrow \text{rightSum} + n_i$
14      else
15         // Algorithm 3.2
16         $\text{embed_branch}(L, v_i, -\text{leftSum}, -1, \text{relPos})$
17         $\text{leftSum} \leftarrow \text{leftSum} + n_i$
18   $\pi \leftarrow \{0\}^n$ // empty arrangement
19   $\pi(r) \leftarrow \text{leftSum} + 1$
20   $\text{relPos}[r] \leftarrow 0$
21   for each vertex $v$ do
22      $\pi(v) \leftarrow \pi(r) + \text{relPos}[v]$
23   return $\pi$
the whole stack of the recursion) in the worst case (for any building the adjacency list only once. The spatial complexity is $O(u \cdot n)$ iterations (for a vertex $	au$ of the rooted tree). Fig. 1(a) shows an optimal projective arrangement of $T^r$; Fig. 1(b) shows an arrangement that needs not be an optimal projective arrangement for $T^r$ and optimal projective for $T^c$ [9]. Clearly, an optimal planar arrangement for $T$ is an optimal projective arrangement for $T^c$ [9]. 

Figure 1(a) shows an optimal projective arrangement of the rooted tree $T^1$, which is not an optimal planar arrangement of $T$; Fig. 1(b) shows an arrangement that is both optimal planar for $T$ and optimal projective for $T^2$.

Algorithm 4.4’s time and space complexities are $O(n)$: the ‘for’ loop (line 7) contains constant-time operations, a single recursive call and, since each loop consists of $d_u = |V(u)|$ iterations (for a vertex $u$), the total running time is $O(\sum_{u \in V} d_u) = O(n)$ because every vertex is visited only once. The spatial complexity is $O(n)$: sorting and building the adjacency list $L^u$ requires $O(n)$ space (for any $u$) and Algorithm 4.6 requires extra $O(n)$ space (for the whole stack of the recursion) in the worst case (for

---

**Algorithm 4.4:** Linear-time calculation of an optimal projective arrangement.

| Line | Function AARRANGE\_OPTIMAL\_PROJECTIVE($T^r$) is |
|------|-------------------------------------------------|
| 1    | **In:** $T^r$ rooted tree at $r$.              |
| 2    | **Out:** An optimal projective arrangement $\pi$.|
| 3    | // **Algorithm 4.2**                           |
| 4    | $L^r \leftarrow \text{SORTED\_ADJACENCY\_LIST\_RT}(T^r)$ |
| 5    | $\pi \leftarrow \{0\}^n$ // empty arrangement |
| 6    | // The starting side ‘right’ is arbitrary.     |
| 7    | **Algorithm 4.6.**                             |
| 8    | **Arrangement($L^r, r, \text{right}, 1, n, \pi$) |
| 9    | return $\pi$                                   |

---

**Algorithm 4.5:** Linear-time calculation of an optimal planar arrangement.

| Line | Function AARRANGE\_OPTIMAL\_PLANAR($T$) is |
|------|------------------------------------------|
| 1    | **In:** $T$ free tree.                  |
| 2    | **Out:** An optimal planar arrangement $\pi$. |
| 3    | // **Algorithm 2.2**                    |
| 4    | $L \leftarrow \text{SORTED\_ADJACENCY\_LIST\_FT}(T)$ |
| 5    | // **Algorithm 2.3**                    |
| 6    | $c \leftarrow \text{FIND\_CENTROIDAL\_VERTEX}(T, L)$ |
| 7    | // list $L$ rooted at $c$ (Section 2)    |
| 8    | $L^c \leftarrow \text{ROOT\_LIST}(L, c)$ |
| 9    | $\pi \leftarrow \{0\}^n$ // empty arrangement |
| 10   | // The starting side ‘right’ is arbitrary. |
| 11   | **Algorithm 4.6.**                      |
| 12   | **Arrangement($L^c, c, \text{right}, 1, n, \pi$) |
| 13   | return $\pi$                            |
Algorithm 4.6: Optimal arrangement of a tree according to its sorted adjacency list.

1 Function ARRANGE(L′, u, τ, a, b, π) is
2     In: (Rooted) sorted adjacency list L′ as described in Section 2; u the root of the subtree to be arranged; τ position of u with respect to its parent p(u); [a, b] interval of positions of the arrangement where to embed T′ u; π the partially-constructed arrangement.
3     Out: π updated with the optimal projective arrangement for T′ u in [a, b].
4     C′ u ← L′[u] // the children of u decreasingly sorted by size
5     if τ is right then side ← right
6     else side ← left
7     for i from 1 to |C′ u| do
8         // the i-th child of u and its size n′ v = |V(T′ v) |
9         v, n′ v ← C′ u[i]
10        if side is left then
11           τ next ← left
12           a next ← a
13           b next ← a + n′ v − 1
14        else
15           τ next ← right
16           a next ← b − n′ v + 1
17           b next ← b
18        ARRANGE(L′, v, τ next, a next, b next, π)
19        if side is left then a ← a + n′ v
20        else b ← b − n′ v
21     side ← opposite side
22     π(u) ← a

5. Conclusions and future work

To the best of our knowledge, our work is the first to highlight a relationship between the MLA problem under planarity and the same problem under projectivity. We have shown that HS’s algorithm for planarity [9] contains a subalgorithm to solve the projective case. We suspect that Iordanskii’s algorithm for planarity [10] may also contain a subalgorithm for the projective case. We have corrected a few aspects of HS’s algorithm (Algorithm 3.2).

We provided two detailed algorithms for the projective case that run without a doubt in O(n) time. One that stems directly from HS’s original algorithm for the planar case (Algorithms 4.3 and 3.2), and another interval-based algorithm (Algorithms 4.4 and 4.6) that builds on HS’s work but is less straightforward. The latter algorithm leads immediately to a new way to solve the planar case in O(n) time (Algorithms 4.5 and 4.6) thanks to the correspondence between the planar case and the projective case that we have uncovered in this article.

GT [5] sketched an algorithm for the projective case and claimed it to run in linear time. PL [14] added some details but not sufficiently, concluding that it runs in $O(n \log d_{\text{max}})$ time, which, as we have seen, overestimates the actual complexity. During the reviewing process of this paper, it has come to our knowledge a Master Thesis [2] where an error in GT’s algorithm is pointed out. This error does not affect our implementation.

It could be the case that a unified approach for planarity and projectivity could also be adopted for the maximum linear arrangement problem [8]. To the best of our knowledge, a polynomial-time algorithm for the unrestricted case is not forthcoming. An intriguing question is if the maximum linear arrangement problem on trees can be solved in linear time for the projective and planar variants as in the corresponding minimization problem.

Acknowledgements

We are grateful to M. Iordanskii for helpful discussions. We thank C. Gómez-Rodríguez for making us aware of reference [2]. LAP is supported by Secretaria d’Universitats i Recerca de la Generalitat de Catalunya and the Social European Fund. RFC and LAP are supported by the grant TIN2017-89244-R from MINECO (Ministerio de Economía, Industria y Competitividad). RFC is also supported by the recognition 2017SGR-856 (MACDA) from AGAUR (Generalitat de Catalunya).
JLE is funded by the grant PID2019-109137GB-C22 from MINECO.

References

[1] Frank Bernhart and Paul C. Kainen. The book thickness of a graph. *Journal of Combinatorial Theory, Series B*, 27(3):320–331, 1979.

[2] Rishi Bommasani. Generalized optimal linear orders. Master’s thesis, Cornell University, 2020.

[3] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, USA, 2nd edition, 2001.

[4] Michael R. Garey and D. S. Johnson. Some simplified NP-complete graph problems. *Theoretical Computer Science*, pages 237–267, 1976.

[5] Daniel Gildea and David Temperley. Optimizing grammars for minimum dependency length. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 184–191, Prague, Czech Republic, 06 2007. Association for Computational Linguistics.

[6] M. K. Goldberg and I. A. Klipker. Minimal placing of trees on a line. Technical report, Physico-Technical Institute of Low Temperatures. Academy of Sciences of Ukranian SSR, USSR, 1976. in Russian.

[7] Frank Harary. *Graph Theory*. Addison-Wesley, Reading, MA, 1969.

[8] Refael Hassin and Shlomi Rubinstein. Approximation algorithms for maximum linear arrangement. In *Scandinavian Workshop on Algorithm Theory - Algorithm Theory - SWAT 2000*, volume 1851, pages 231–236, 2000.

[9] Robert A. Hochberg and Matthias F. Stallmann. Optimal one-page tree embeddings in linear time. *Information Processing Letters*, 87(2):59–66, 2003.

[10] M. A. Iordanskii. Minimal numberings of the vertices of trees — Approximate approach. In Lothar Budach, Rais Gatič Bukharajev, and Oleg Borisovič Lupanov, editors, *Fundamentals of Computation Theory*, pages 214–217, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg.

[11] M.A. Iordanskii. *Introduction to graph theory*. Minin Pedagogical State University, Nizhny Novgorod, 2014. in Russian.

[12] Marco Kuhlmann and Joakim Nivre. Mildly non-projective dependency structures. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, COLING-ACL ’06, pages 507–514, 07 2006.

[13] Igor Melčuk. *Dependency Syntax: Theory and Practice*. State University of New York Press, Albany, NY, USA, 1988.

[14] Y. A. Park and Roger Levy. Minimal-length linearizations for mildly context-sensitive dependency trees. In *Proceedings of the 10th Annual Meeting of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT) conference*, pages 335–343, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[15] Fan R. K. Chung. On optimal linear arrangements of trees. *Computers & Mathematics with Applications*, 10(1):43–60, 1984.

[16] Yossi Shiloach. A minimum linear arrangement algorithm for undirected trees. *SIAM Journal on Computing*, 8(1):15–32, 1979.