A Scalable Method for Readable Tree Layouts

Kathryn Gray, Mingwei Li, Reyan Ahmed, Md. Khaledur Rahman, Ariful Azad, Stephen Kobourov, Katy Börner

Abstract— Large tree structures are ubiquitous and real-world relational datasets often have information associated with nodes (e.g., labels or other attributes) and edges (e.g., weights or distances) that need to be communicated to the viewers. Yet, scalable, easy to read tree layouts are difficult to achieve. We consider tree layouts to be readable if they meet some basic requirements: node labels should not overlap, edges should not cross, edge lengths should be preserved, and the output should be compact. There are many algorithms for drawing trees, although very few take node labels or edge lengths into account, and none optimize all requirements above. With this in mind, we propose a new scalable method for readable tree layouts. The algorithm guarantees that the layout has no edge crossings and no label overlaps, and optimizing one of the remaining aspects: desired edge lengths and compactness. We evaluate the performance of the new algorithm by comparison with related earlier approaches using several real-world datasets, ranging from a few thousand nodes to hundreds of thousands of nodes. Tree layout algorithms can be used to visualize large general graphs, by extracting a hierarchy of progressively larger trees. We illustrate this functionality by presenting several map-like visualizations generated by the new tree layout algorithm.

Index Terms—Tree layouts, force-directed, readability

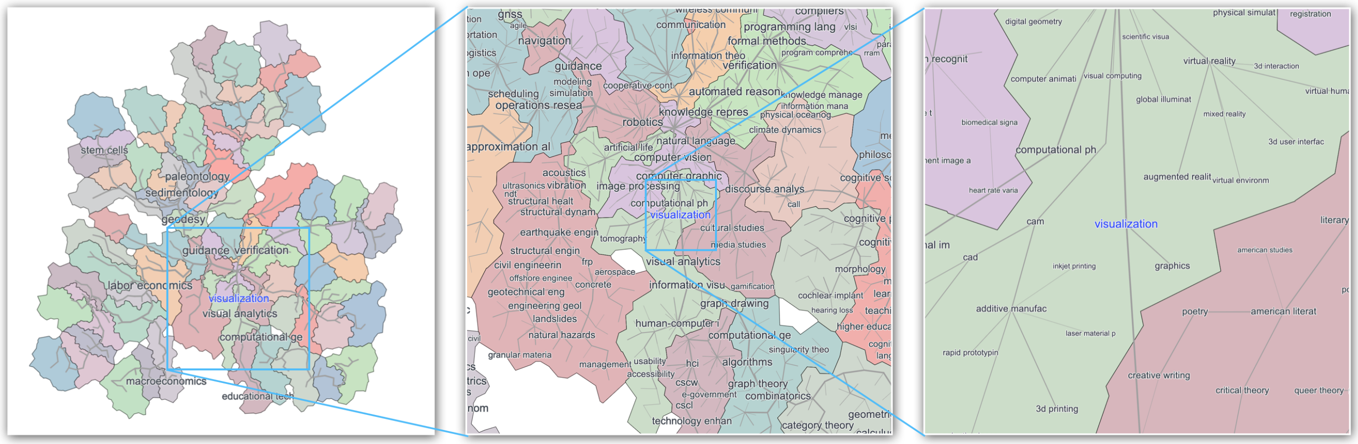


Fig. 1: A map of a real-world research topics network with over 5,000 nodes that provides semantic zooming, generated with the scalable method for readable tree layout. The visualization provides an overview of the dataset, showing the high-level structure, including important nodes and edges. Zooming into a particular area of interest provides more details. The layout obtained by the proposed algorithm is compact, there are no edge crossings, and there are no label overlaps.

1 Introduction

Many real-world datasets can be represented by a network where each node represents an object and each link represents a relationship between objects; e.g., the tree of life captures the evolutionary connections between species and a research topics network captures relationships between research areas; see Fig. 1. Abstract networks can be modeled by node-link diagrams, with points representing nodes and segments/curves representing the edges. However, real-world datasets have labels associated with nodes and attributes such as edge lengths that are not captured in node-link diagrams. For example, the tree of life has species names as node labels and evolutionary distance between the corresponding species as edge data. These networks would benefit from a visualization that shows the labels and captures the desired edge lengths. However, just adding labels to an existing layout will result in an unreadable visualization with many overlapping labels. One could scale the layout to remove such overlaps, but this would blow up the drawing area to unmanageable size. Label overlaps could also be removed with specialized overlap removal algorithms but results in layout changes: not just changing edge lengths, but also changing the topology (e.g., breaking up clusters, or introducing new edge crossings).

We consider tree layouts to be *readable* if they meet some basic requirements: node labels should not overlap, edges should not cross, edge lengths should be preserved, and the output should be compact. This gives us two hard *constraints*: (C1) No edge crossings, (C2) No label overlaps. We also consider two additional desirable properties that the algorithm *optimizes*: (O1) desired edge lengths, (O2) compactness of the drawing area. Finally, in order to efficinetly handle large networks, we also parallelize the computation.

Preserving pre-specified, desired edge lengths is important to many real-world datasets, but is not taken into account by most network and tree layout algorithms. Keeping track of the drawing area required (e.g., by comparing to the sum of areas of all labels), makes it clear that simply scaling up a given layout until labels do not overlap

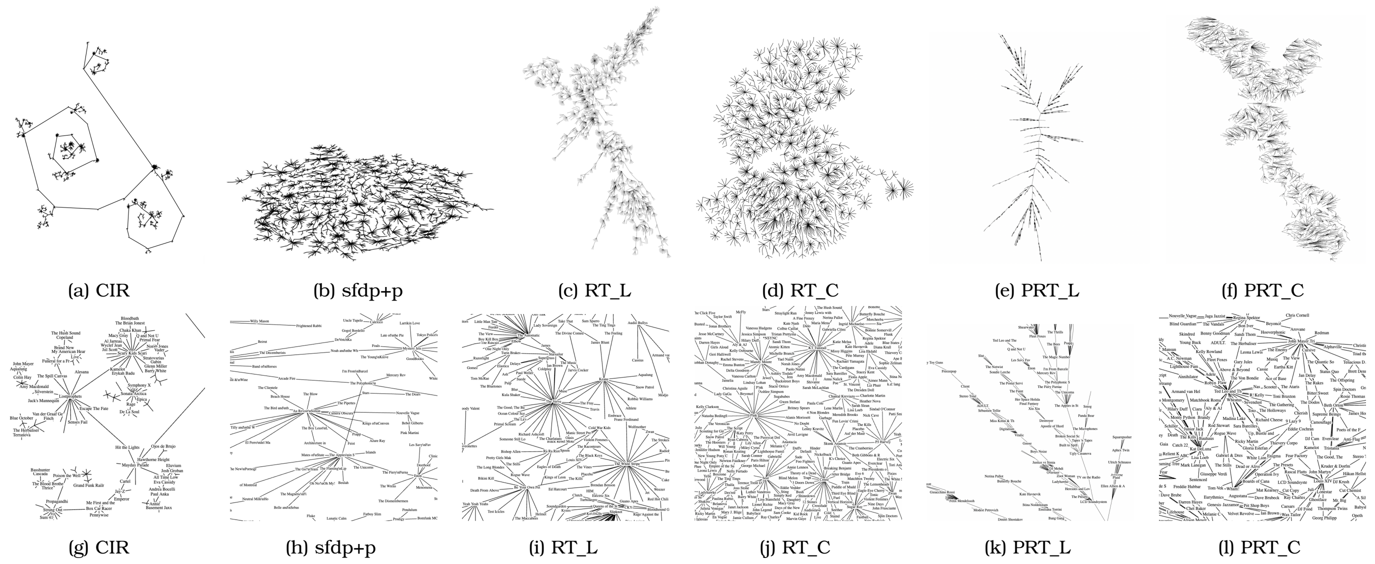


Fig. 2: Comparison of the layout of the Last.fm graph with uniform edge lengths. The CIR layout produces edges with very different lengths, while the sfdp+p has crossings, and neither of them captures the structure of the underlying tree. On the other hand, our algorithms generate layouts that show the structure of the input network. In an overview such as this one, several of the results look similar, but in these zoomed-out layouts we cannot see all the crossings, label overlaps, and area used; we provide a detailed quantitative evaluation in Table 1, which shows differences as large as orders of magnitude.

results in unusable layouts with areas that are 4-6 orders of magnitude greater than needed. Finally, the scalability of the algorithm is important when dealing with larger datasets containing hundreds of thousands of nodes.

Despite there being more than 300 algorithms for drawing trees [49], none can guarantee the two constraints (no crossings, no overlaps), while also optimizing desired edge lengths and area. With this in mind, we propose a scalable method that can guarantee both constraints while optimizing desired edge lengths and area. We evaluate four variants of the proposed method for Readable Tree layout (RT_L, RT_C, PRT_L, PRT_C) with 4 different real-world datasets of different sizes: from trees with 2,588 nodes, up to trees with 100,347 nodes; see Fig. 2. We further compare our method against state-of-the-art general network layout and tree layout algorithms, by relaxing some of the constraints. We experimented with half a dozen prior methods, but none of them are directly comparable (as discussed in detail in Sec. 2). In this paper we report the results obtained by two of these prior methods: (spfd+p) the scalable force directed placement [29] algorithm together with label overlap removal via the PRoxImity Stress Model in GraphViz [19], and (CIR) the CIRcular tree layout algorithm from yED [53].

We also show the utility of the proposed readable tree layouts method in visualizing general netowrks. There are many algorithms for extracting *important* trees from a given network: minimum spanning trees, maximum spanning trees, network backbone trees, etc. Motivated by people's familiarity with maps [9], we use a multi-level Steiner tree algorithm [1] to create a level-of-detail representation of an underlying general network, which underlies an interactive map-like representation that provides semantic zooming; see Fig. 1.

We propose a new scalable method for visualizing large, labeled trees. The method maintains the two hard constraints (C1) No edge crossings and (C2) No label overlaps, while optimizing two desirable properties (O1) desired edge lengths and (O2) compactness of the drawing area. The two constraints and two optimization goals underlie *readable* layouts, as shown in prior work:

Edge crossings are known to make network layouts less readable [45]. Since trees are planar, it is possible to create layouts without crossings, justifying the first constraint (C1). Overlapping labels make the labels harder to read. Overlaps are one of the metrics for usability of layouts [31, 50], justifying the second constraint (C2).

Preserving desired edge lengths is a standard requirement in instances where edge lengths capture important information, e.g., evolutionary time in the tree of life, and phylogenetic trees in general [3, 4, 7, 16, 30, 36]. Uniform edge lengths (a special case where all desired edge lengths are the same) are preferable in cases where all edges represent the same notion of connectivity [45, 50]. This justifies the first optimization goal (O1). Note that simmultaneously ensuring C1 and O1 is an NP-hard problem [18], which is why O1 is optimized, rather than guaranteed.

Compactness of the layout is an important feature for providing an effective overview of the underlying network [45] as a lot of white space can be detrimental to readability [42]. This justifies the second optimization goal O2.

Fig. 3 shows the workflow of the proposed readable tree (RT) layout method. The input is a node-labeled tree with pre-specified edge lengths from which a multi-level Steiner tree is computed to provide semantic zooming. The algorithm next computes a crossing-free initial layout (C1) of the tree and maintains this property in every subsequent step. In the iterative refinement step, the algorithm employs force-directed layout improvement, tailored to remove label overlaps, preserve desired edge lengths (O1), and minimize the drawing area (O2). In the final iteration step any remaining overlaps are removed, thus enforcing (C1).

For the sake of consistency in the experimental evaluation, we use the same methodology to assign desirable edge lengths to all datasets. Specifically, we extract multi-level Steiner trees (described in Sec. 7.1) and assign de-

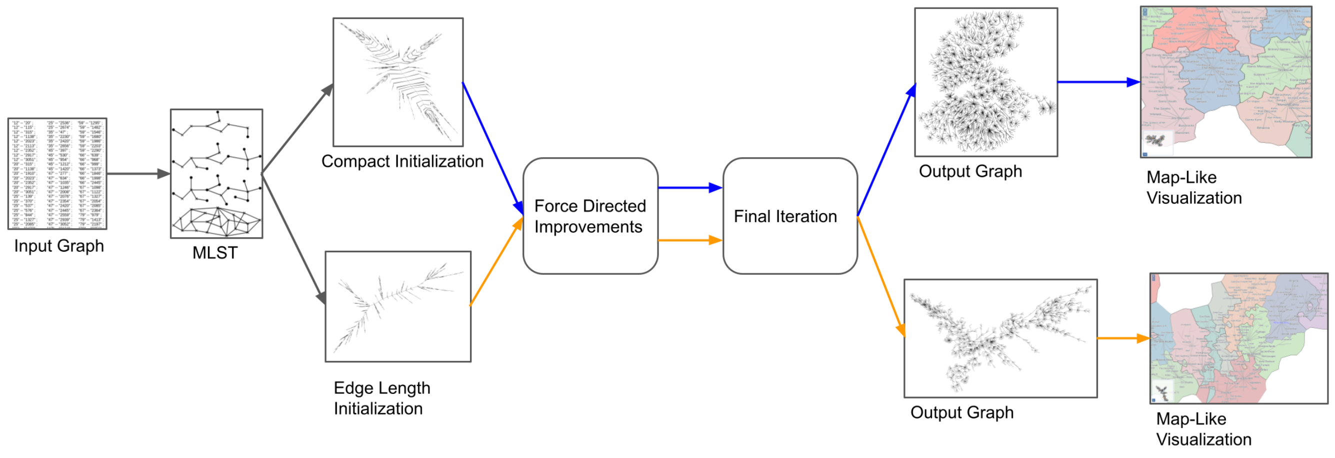


Fig. 3: Overview of the readable tree (RT) method. The input is a node-labeled tree with pre-specified edge lengths from which a multi-level Steiner tree is computed to provide semantic zooming for the interactive, map-like visualization. The algorithm next computes a crossing-free initial layout and maintains this property in every subsequent step. In the iterative refinement step, the algorithm employs force-directed layout improvement, tailored to remove label overlaps, preserve desired edge lengths, and minimize the drawing area. In the final iteration step, any remaining overlaps are removed.

sired lengths proportional to the level where the edge first appears.

Note that simultaneously improving the desirable properties is a non-trivial task, as the individual properties could require contradictory layout changes. For example, consider a high-degree node with all adjacent edges having the same desirable length. It not possible to preserve the edge lengths and obtain a compact layout simultaneously (with no label overlaps). To preserve edge lengths without overlapping labels we need larger drawing area. To obtain a compact layout with no label overlaps we need to distort edge lengths; see Fig. 4.

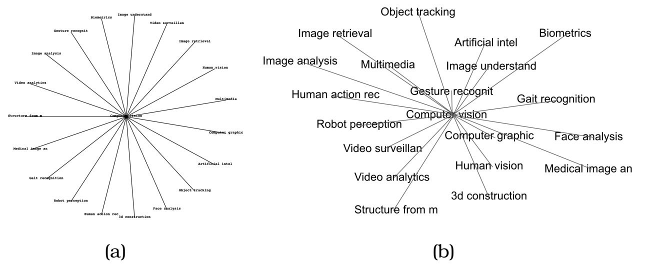


Fig. 4: While maintaining no label overlaps, preserving desired edge lengths and compactness are contradictory optimizations. In the input tree all edge lengths should be the same, and this can be preserved (a) at the expense of compactness; alternatively, a compact layout (b) distorts desired edge lengths.

With this in mind, our proposed method can emphasize desired edge length preservation or compactness. This is determined by selecting one of the two initial layouts. `Edge-Length-Initialization` preserves desired edge lengths but allows label overlaps. `Compact-Initialization` prioritizes compactness, but does not preserve desired edge lengths. In the following steps (iterative refinement and final iteration) these initial layouts are modified to ensure C1 and C2 and optimize O1 and O2.

We provide two implementations of the proposed method: one that requires parallel hardware and that does not. The d3.js [10] works well for smaller instances. The parallel variant of our method, using OpenMP [14], is 1-2 orders of magnitude faster and can handle trees with hundreds thousand nodes. Note that the interactive map-like visualization with semantic zooming relies on the readable tree layout algorithm which is run just once per dataset as a pre-processing step, and we consider runtime in minutes to be acceptable.

We quantitatively evaluate our algorithms by measuring compactness, desired edge length preservation and runtime. For comparison we use two state-of-the-art layout methods. Note that the two prior methods do not take desired edge lengths into account, and to make the comparison somewhat fair we use uniform edge lengths. Also, one of the two methods can produce edge crossings.

Since even the smallest tree we work with has more than 2500 labeled nodes, we also make the results accessible via interactive, map-like visualizations. Specifically, using the multi-level Steiner tree hierarchy extracted from the input, we provide semantic zooming functionality that allows us to see the global structure (high level) and local details (low level). The interactive visualization is accessible here: https://tiga1231.github.io/zmlt/demo/overview.html

2 Related Work

Tree and Network Layout Algorithms: Drawing trees has a rich history: Treevis.net [49] contains over three hundred different types of visualizations. Here we briefly review algorithms for 2D node-link representations, starting with arguably the best known one by Reingold and Tilford [48]. This and other early variants draw trees recursively in a bottom-up sweep. These methods produce crossings-free layouts, but do not consider node labels or edge lengths. We will now broaden our scope to general graphs. While general graphs are not necessarily planar, the layout techniques and ideas can be applied to trees. Most general network layout algorithms use a force-directed [17, 20] or stress model [12, 34] and provide a single static drawing. The force-directed model works well for small networks, but does not scale to large networks. Speedup techniques employ a multi-scale variant [21, 27] or use parallel and distributed computing architecture such as VxOrd [11], BatchLayout [47], and MULTI-GILA [2]. Libraries such as GraphViz [19] and OGDF [15] provide many general network layouts, but may not support interactions. Whereas visualization toolkits such as Gephi [5] and yEd [53] support visual network manipulation, and while they can handle large networks, the amount of information rendered statically on the screen makes the visualization difficult to use for large networks.

Overlap Removal and Topology Preservation: In theory nodes can be treated as points, but in practice nodes are labeled and these labels must be shown in the layout [28, 44]. Overlapping labels pose a major problem for most layout generation algorithms, and severely affect the usability of the resulting visualizations. A simple solution to remove overlaps is to scale the drawing until the labels no longer overlap. This approach is straightforward, although it may result in an exponential increase in the drawing area. Marriott *et al.* [38] proposed to scale the layout using different scaling factors for the x and y coordinates. This reduces the overall blowup in size but may result in poor aspect ratio. Gansner and North [25], Gansner and Hu [24], and Nachmanson *et al.* [41] describe overlap-removal techniques with better aspect ratio and modest additional area. However, these approaches can and do introduce edge-crossings, even when starting with a crossings-free input. Placing labeled nodes without overlaps has also been studied [31, 37, 40], but these approaches also cannot not guarantee crossings-free layouts.

The need for new algorithms: While there exist many algorithms for generating tree and network layouts, to the best of our knowledge, no existing algorithm considers the four aspects of the readability of labeled tree layouts: no edge crossings, no node overlaps, compact drawing area and preserved desired edge lengths. For example, one of the most frequently used network visualization systems, GraphViz [19], has an efficient layout algorithm based on the scalable force directed placement (sfdp) algorithm [29] and can remove label overlaps via the PRoxImity Stress Model (PRISM) [24]. The output, however, does not optimize the given edge lengths and cannot ensure the crossing constraint; examples in this paper contain 100-1000 crossings.

The popular visualization library d3.js provides a link-force feature to optimize desired edge lengths, but cannot ensure the crossing constraint and cannot remove label overlaps without blowing up the drawing area. Another excellent visualization toolkit, yEd [53], provides a method that can draw trees without edge crossings and optimize compactness. However, none of the methods available in yED can preserve the desired edge lengths and one cannot remove label overlap without blowing up the drawing area. Nguyen and Huang [43] describe an algorithm for compact tree layouts (note that we use a similar initialization step), however, their approach is not concerned with edge lengths and node labels.

Different dimensionality reduction techniques such as t-SNE [52] and its variants [32, 35] are hard to use to visualize tree networks since they do not guarantee crossing-free and label overlap-free drawing; in one of our experiments, we observed that t-SNE can generate more than 50 crossings in a small tree of 100 nodes. We believe our paper fills this gap in the literature by developing and making available a scalable method for readable tree layouts.

3 Readable Tree Layout Algorithm

Our algorithm can be split into five parts, multi-level Steiner tree extraction, an initial layout, a force-directed layout improvement, a final iteration ensuring no label overlaps, and a map-like visualization; see Fig. 3. Since we rely on existing techniques for the first step and the last steps, we briefly discuss them in Sec. 7. In this section, we discuss the remaining three steps in detail.

3.1 Initialization

As mentioned in Sec. 1, desired edge length preservation and compactness are contradictory optimization goals. Hence, our algorithm can produce two different types of layouts: one emphasizing the desired edge length preservation and the other emphasizing compactness. The two of intialization steps below influences the type of the obtained layout.

Edge Length Initialization:

The first initialization creates a layout that is crossing free and preserves all edge lengths, although it may have label overlaps. Our crossings-free initialization is similar to a prior work [3]. We select a root node and assign a wedge region (sector) to every child. Each child is then placed along an angle bisector of the assigned wedge, away from its parent by the desired edge length. We continue this process until the coordinates for each node have been computed; see pseudcode Alg. `Edge-Length-Initialization`. We traverse the nodes using breadth-first search (BFS) which visits parents before children. Since the angular regions are unbounded, the algorithm can preserve all the desired edge lengths exactly. When we assign wedge regions to child nodes, the angles are proportional to the size of the subtree rooted from the child; see Fig. 6. When the input is a balanced tree this algorithm computes a symmetric layout; see Fig. 5.

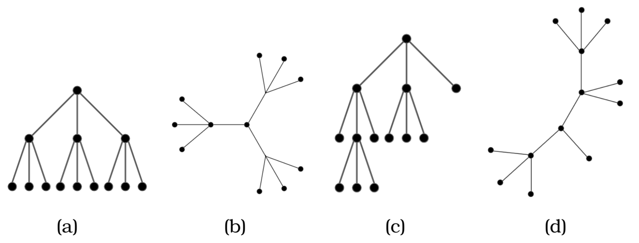


Fig. 5: Illustrating the output layout of `Edge-Length-Initialization` w.r.t. different types of inputs. If the tree is balanced (Fig: 5a) then the output is symmetric (Fig: 5b). If the tree is not balanced (Fig: 5c) then the output is not symmetric (Fig: 5d).

Compact Initialization:

The second initialization also creates a crossing-free layout that is compact, although desired edge lengths might be ditorted. Instead of assigning wedge regions to children of a node, we assign the entire fan area to the children and place children along the arc of the fan; see pseudcode Alg. `Compact-Initialization`. This results in a wider spread of nodes, but does not preserve the desired edge lengths; see Fig. 7 and compare it with Fig. 6.

3.2 Force Directed Improvements

The next step is to use a force directed algorithm to optimize our soft constraints. Here we describe all of the forces used, as well as how we prevent edge crossings at every step of the force directed improvement.

Algorithm `Edge-Length-Initialization`

```
    Input: $G = (V, E)$ // The tree network
1   $root \in V$ // root of the tree
2   $\{\dots, n_v, \dots\}$ // Size of subtree rooted from $v \in V$
3   $\{\dots, DEL_{uv}, \dots\}$ // Desired edge length from $u$ to $v \in V$
    Output: $X$ // Crossing-free initial layout for RT_L
4   Function Initial_Layout($G, root$):
5       $X_{root} \leftarrow (0, 0)$;
6       $W_{root} \leftarrow wedge(center = root,$
7           $radius = DEL_{root}, angle\_range = [0, 2\pi))$;
8       for parent node $p \in BFS(G, root)$ do
9           $\{c_1, c_2 \dots c_{|C|}\} \leftarrow children(p)$;
10          $angle\_ranges = \{\dots A_{c_i} \dots\} \leftarrow$
             $partition(angle\_range(W_p); n_{c_1}, n_{c_2}, \dots)$;
11          for $c_i \in C$ do
12              $W_{c_i} \leftarrow wedge(center = p,$
13                  $radius = DEL_{c_i}, angle\_range = A_{c_i})$;
14              $X_{c_i} \leftarrow midpoint(arc(W_{c_i}))$;
15      return $X$;
```

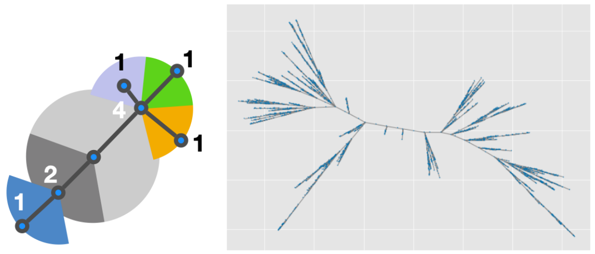


Fig. 6: **Left:** Illustration of initial radial layout of RT_L. The numbers indicate the proportion of wedge sectors determined by the size of induced subtrees of child nodes. **Right:** Initial layout of the last.FM network.

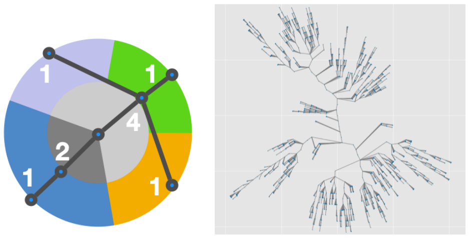


Fig. 7: **Left:** Illustration of initial radial layout of RT_C. The number indicate the proportion of wedge sectors, determined by the size of induced subtrees of child nodes. **Right:** Initial layout of the last.FM network by RT_C.

Algorithm `Compact-Initialization`

```
    Input: $G = (V, E)$ // The tree network
1   $root \in V$ // root of the tree
2   $\{\dots, n_v, \dots\}$ // Size of subtree rooted from $v \in V$
3   $\{\dots, d_v, \dots\}$ // Number of hops from the root to $v \in V$
    Output: $X$ // Crossing-free initial layout for RT_C
4   Function Initial_Layout($G, root$):
5       $X_{root} \leftarrow (0, 0)$;
6       $W_{root} \leftarrow wedge(center = root,$
7           $radius = 1, angle\_range = [0, 2\pi))$;
8       for parent node $p \in BFS(G, root)$ do
9           $\{c_1, c_2 \dots c_{|C|}\} \leftarrow reorder(children(p),$
10              $mode =$ 'centralize heavy subtrees');
11          $angle\_ranges = \{\dots A_{c_i} \dots\} \leftarrow$
             $partition(angle\_range(W_p); n_{c_1}, n_{c_2}, \dots)$;
12          for $c_i \in C$ do
13              $W_{c_i} \leftarrow wedge(center = root,$
14                  $radius = d_{c_i}, angle\_range = A_{c_i})$;
15              $X_{c_i} \leftarrow midpoint(arc(W_{c_i}))$;
16      return $X$;
```

3.2.1 Maintaining the Crossings-Free Constraint

The force-directed algorithm improves the layout by applying different forces while ensuring that there are no edge crossings introduced in any iteration of the algorithm. The algorithm starts with a layout computed in the previous initialization step. In each iteration of the algorithm, it computes different forces for each node as discussed in the next sections. Then for each node v, it computes the movement T_v applied by the forces. The algorithm combines the forces linearly, with scaling factors discussed in Sec. 5. If T_v introduces any edge crossings, then the algorithm does not apply the movement. Otherwise, it computes the new coordinate of v according to T_v. The algorithm continues this step until a maximum number of iterations is reached; see pseudocode Alg. `Force-Directed-Improvement`.

3.2.2 Label Overlap Force

We use an elliptical force by modifying the traditional collision force to help remove the label overlaps. Since labels are typically wider than they are tall, a circular collision region potentially wastes space above and below the labels. We build an elliptical force out of a circular collision force by stretching the y-coordinate by a constant factor b (e.g., by default we use $b = 3$) before a circular collision force is applied, and restoring the coordinates after the force is applied. The velocity computed by the collision force is processed in a similar manner, with a reciprocal scaling factor. Formally, let X_v denote the coordinate of node v. We specify a different collision radius depending on label size, denoted by r_v, for every node v. Note that the collision radius depends on both the font size of a label and the number of characters in the label. A circular collision force first calculates a movement T'_v, and then the elliptical movement T_v is computed by stretching the x-coordinate and compressing the y-coordinate, e.g., $T_v.x = T'_v.x \times b, T_v.y = T'_v/y$. Then we update the coordinate X_v by adding T_v.

3.2.3 Edge Length Force

The edge length force is designed to maintain the desired edge lengths. For every edge we apply either a repulsive force $f_r^e = K/d \cdot I_{d<l_e}(d)$ (when the edge is compressed) or an attractive force $f_a^e = Kd \cdot I_{d>l_e}(d)$ (when the edge is stretched), determined by the indicator function. The force is proportional/reciprocal to distance d.

3.2.4 Distribution Force

We define a global node distribution force: a repulsive force between every pair of nodes. We set the repulsive force between two nodes inversely proportional to the squared distance in the current layout; similar to an electrical charge between nodes: $|f_d(u, v)| = s(u, v)/||X_u - X_v||^2$, where $s(u, v)$ denotes the strength of the force between nodes and depends on the longest desired edge length adjacent to u and v. We set $s(u, v) =$

Algorithm `Force-Directed-Improvement`

```
Input: $G = (V, E)$ // The tree network
1 $X$ // Crossing-free initial layout
2 $niter$ // Number of iterations
Output: $X$ // Improved crossing-free layout
3 Function Force_Directed_Improvement($G, X$):
4     for $i = 1, 2, \ldots, niter$ do
5         for each node $u \in V$ do
              // Label overlap force
6             for each node $v \in collision\_region(u)$ do
7                 $T_v \leftarrow \Delta T_v + f_c(X_v, X_u)$ ;
              // Edge length force
8             for each neighbor $v$ of $u$ do
9                 if $length(X_u, X_v) > l_{uv}$ then
10                    $T_v \leftarrow T_v + f_a(X_u, X_v)$;
11                else
12                    $T_v \leftarrow T_v - f_r(X_u, X_v)$;
              // Distribution force
13            for each node $v$ of $u$ do
14                $T_v \leftarrow T_v - f_d(X_u, X_v)$ ;
              // Node-edge force
15            for each neighbor $v$ of $u$ do
16                $T_v \leftarrow T_v - f_{node-edge}(X_u, X_v)$ ;
              // Maintaining no edge crossings
17            if $T_v$ does not introduce edge-crossing then
18                $X_v \leftarrow X_v + T_v$;
19    return $X$;
```

$max_{w \in V, (u,w) \in E} \{l_{uw}\} * max_{w \in V, (v,w) \in E} \{l_{vw}\}$.

3.2.5 Node-Edge Force

Finally, we define a force between nodes and edges. This improves readability by reducing the number of instances where labels are placed over edges. This force is inversely proportional to the distance between the node and edge, acting orthogonally from the edge (evaluating to zero if the node does not project onto the edge segment or is too far from the edge): $|f_{node-edge}(v, e)| = c/d(v, e)$, where c is a constant across all pairs and $d(v, e)$ denotes the Euclidean distance between node v and edge e.

3.3 Final Iteration

The final iteration is needed to ensure any remaining overlaps are removed. It is possible that this step will have no work to do, but this check is necessary in order to enforce our hard constraint (C2).

In this step, we go over all pairs of overlapping nodes and move them until the overlap is repaired. To do this we check whether we can move one of the overlapping nodes so that the distance between two nodes increases without introducing any crossing and label overlap. Specifically, for each node v of the pair of nodes, we consider a square bounding box that has a small area. We denote the set of nodes in that bounding box by V'. We then sample some random points from that bounding box. For each of these sample points, we check whether we have an overlap-free and crossing-free drawing. If we find such a point, then we move v to that point and consider the next label overlap; see pseudocode Alg. `Final-Iteration`.

Note that in some cases this step is not needed at all, as all overlaps are removed during the force-directed layout improvement step. In other cases (e.g., larger input instances with denser subtrees) a handful of final iteration steps are needed to remove remaining overlaps.

Algorithm `Final-Iteration`

```
Input: $steps$ // Sample size
1 $size$ // Width of sample area
Output: $X$ // A crossing-free layout with reduced label overlaps
2 Function Final_Overlap_Removal($steps, size$):
3     for $u, v \in V \times V$ s.t. $u$ and $v$ overlaps do
4         for $k = 1, 2, \cdots, steps$ do
5             $r = random(0, 1)$;
6             $\Delta X_u = (r * size) - (size/2)$;
7             if $\Delta X_u$ does not introduce crossing and new overlap then
8                 $X_u \leftarrow X_u + \Delta X_u$
9             if $u$ and $v$ overlaps then
10                $\Delta X_v = (r * size) - (size/2)$;
11                if $\Delta X_u$ does not introduce crossing and new overlap then
12                    $X_v \leftarrow X_v + \Delta X_v$
13    return $X$;
```

4 Parallel Readable Tree Drawing

Here we describe the parallel version of the algorithm (PRT), which again maintains the hard constraints, but now also adds scalability in order to handle larger trees. The previous algorithm, RT, generates good quality layouts that satisfy all the hard constraints and optimize the soft constraints well. Additionally, RT does not require any specialized equipment and can be run on any computer. However, it is sequential in nature and does not work well for networks with more than about 5000 nodes. The parallel tree version takes advantage of opportunities to speed up some of the necessary computations. As before, there are two variants: Edge Length Initialized Parallel Readable Tree (PRT_L) algorithm and Compactness Initialized Parallel Readable Tree (PRT_C) algorithm that emphasize the preservation of desired edge lengths and compactness.

Note that force calculations in the PRT algorithm have an inherent dependence on neighbors and non-neighbors (i.e., collision/edge forces for a node depend on the coordinates of other nodes). If such forces are computed in different threads, they cannot be seamlessly integrated. Instead of running the entire algorithm in parallel, we use the mini-batch approach, similar to BatchLayout [47]. The mini-batch technique is commonly used for parallel Stochastic Gradient Descent (SGD) training, where one gradient update has a dependency on other gradient updates [26].

The PRT algorithm follows the same workflow shown in Fig. 3: a crossings-free initial layout, followed by customized force-directed improvement, and a final iteration the enforces the overlaps constraint.

Algorithm `PRT-Center-Node`

```
Input: $G = (V, E)$ // The tree network
Output: $n_c$ // center node id $n_c$
1 Function PRT_Center_Node($G$):
2     $C_u \leftarrow 0.0, \forall u \in V$
3     for each node $u \in V$ in parallel do
4         $\mathcal{D} \leftarrow \sum_{v \in V} dist(u, v)$
5         $C_u \leftarrow \frac{|V|}{\mathcal{D}}$
6     $n_c \leftarrow \arg\max_u C_u$
7     return $n_c$
```

Initialization: We create an initial layout of the tree with no edge crossings using parallelized versions of Alg. `Edge-Length-Initialization` and Alg. `Compact-Initialization`. The most time-consuming part here is determining the center node that serves as *root*. The time complexity of this step is $O(n^2)$, where n is the number of vertices in G. Thus, we fully parallelize Alg. `PRT-Center-Node`, where normalized closeness centrality [6] is computed for each vertex. Then, we take the node with the maximum score as root. Since G is a tree, we can apply BFS to compute the distance between vertices u and v as shown in line 3 of Alg. `PRT-Center-Node`.

We place the center node at the origin of the Cartesian coordinate system and iteratively traverse the tree in a BFS fashion, placing nodes so that they do not introduce edge-crossings. The runtime of this part of the algorithm is $O(n)$. Thus, finding the center node is the slowest step of Initialization, and that has been effectively parallelized so that it has minimal effect on the overall runtime.

Algorithm `PRT-Force-Directed-Improvement`

```
   Input: $G = (V, E)$ // The tree network
1  $X$ // Crossing-free initial layout
2  $batch$ // No. of vertex batches form V
3  $samples$ // Sample size
4  $niter$ // Number of iterations
   Output: $X$ // Improved crossing-free layout
5  Function $PRT\_Force\_Directed\_Improvement(G, X)$:
6      for $i = 1, 2, \ldots, niter$ do
7          $T \leftarrow \{0\}^{|V|\times 2}$
8          Partition $V$ into $B = \lceil \frac{|V|}{batch} \rceil$ batches
9          for each batch $B \in V$ do
10             for each node $u \in B$ in parallel do
                   // Label overlap force
11                 for each node $v \in collision\_region(u)$ do
12                     $T_v \leftarrow \Delta T_v + f_c(X_v, X_u)$
                   // Edge length force
13                 for each neighbor $v$ of $u$ do
14                     if $length(X_u, X_v) > l_{uv}$ then
15                         $T_u \leftarrow T_u + f_a(X_u, X_v)$
16                     else
17                         $T_u \leftarrow T_u - f_r(X_u, X_v)$
                   // Distribution force
18                 for a random node $w$ upto samples times do
19                     $T_u \leftarrow T_u - f_d(X_u, X_w)$
                   // Node-edge force
20                 for each neighbor $v$ of $u$ do
21                     $T_u \leftarrow T_u - f_{node-edge}(X_u, X_v)$
                   // Maintaining no edge crossings
22                 if $T_u$ does not introduce edge-crossing then
23                     $X_u \leftarrow X_u + T_u$
24     return $X$;
```

Parallel Force-directed Improvement: We describe the parallel force-directed improvement of layout in Alg. `PRT-Force-Directed-Improvement`. In each iteration of the algorithm, we select a batch B from the set of nodes V and compute attractive and repulsive forces in parallel similar to the BatchLayout method [47]. We apply label overlap forces, edge length forces, and node-edge forces to each node and edge of B in a similar way described in Sec. 3.2. To compute repulsive distribution forces with respect to the non-neighboring nodes, we select $sample$ nodes at random, to speed up the process by approximate repulsive force computation [46]. For each random node w, we compute repulsive force $f_d(X_u, X_w)$ and update the temporary coordinates T_u. Note that this force computation for nodes within the same batch is independent and thus we can run it in parallel. Before updating the coordinates of a batch, we check whether it introduces edge-crossings (line 22). Even though an increased number of batches exposes more parallelism, the quality of the layout may be negatively impacted, as observed in stochastic gradient descent (SGD) [46]. We found that a batch size of 128 or 256 gives a good balance between speed and quality [47]. Since the batch size is small compared to the size of the tree, we perform sequential updates in line 23. However, we perform the edge-crossing check in parallel.

Parallel Final Iteration: This step checks for any remaining label overlaps and repairs them in parallel. The underlying edge crossings check method is similar to the parallel force computation in Alg. `PRT-Force-Directed-Improvement`. Other than the parallel edge crossings check, the algorithm is similar to Alg. `Final-Iteration`. For each overlapping pair of nodes, we sample some random points from a square bounding box that has a small area. For each of these sample points, we check whether we have an overlap-free and crossing-free drawing. When we find such a point, we move the node there.

5 Algorithm Parameters

Like many force-directed algorithms, our RT and PRT methods depend on several parameters. We set some default parameter values based on prior work. For example, we select effective approximation algorithms to compute multi-level Steiner trees and the number of levels in the trees proportional to size of the underlying data based on prior work [1]. We set the batch size of Alg. `PRT-Force-Directed-Improvement` equal to 128-256 as in BatchLayout [47]. Repulsive force parameters are based on those in Force2Vec [46].

We performed a small-scale parameter search for most of the remaining parameters. To determine good values for these parameters we extracted samples with 2,000 nodes from each of our datasets and analyzed the effect of parameter modification. We do this by setting up default values from pilot experiments and explore modifying one parameter at a time, while fixing the remaining ones.

5.1 Different Forces

In Alg. `Force-Directed-Improvement` (lines 6-16) and Alg. `PRT-Force-Directed-Improvement` (lines 11-21), we combine several different forces acting on each node of the input network. Specifically, there are four types of forces: the label overlap force F_c to remove label overlaps, the edge length force F_l to achieve the desired edge lengths, the distribution force F_d to distribute the nodes in the drawing area uniformly, and the node-edge force F_{ne} to keep adjacent nodes closer. Each of these forces has a strength, or scaling factor, that indicates its impact on the overall force. We denote the strengths of the label overlap force, edge length force, distribution force, and node-edge force by S_c, S_l, S_d and S_{ne}, respectively. The range of the strength values is $[0-1]$. Then the total force $F(u)$ applied on a node u is calculated by the following equation:

$$F(u) = S_c \cdot F_c(u) + S_l \cdot F_l(u) + S_d \cdot F_d(u) + S_{ne} \cdot F_{ne}(u)$$

Edge Length Force: The edge length force, F_l, plays a very important role in all RT and PRT variants as it corresponds to one of the two optimization goals (O1). With this in mind, we use the highest strength for this force: $S_l = 1$.

Label Overlap Force: The initialization based on edge length preservation consistently results in more overlaps that the compactness-based one. Hence to determine an appropriate strength for the label overlap force, we use Alg. `Edge-Length-Initialization` in the parameter search. Fig. 8a shows the percentage of label overlaps of all networks, with respect to the initial overlaps, after 50 iterations of Alg. `Force-Directed-Improvement`. As we increase the strength of this force from 0 to 1, overlaps decrease, while edge length preservation decreases, as illustrated in Fig. 8b. To balance this, we set the strength of label overlap force to 0.16, leaving around 10% overlaps (which are removed in the final step), while providing good edge length preservation.

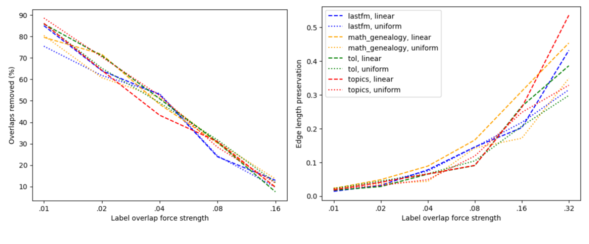


Fig. 8: Illustrating the impact of the label overlap force strength. (a) shows the percentage of label overlaps with respect to the label overlap force. (b) shows the edge length preservation with respect to the label overlap force.

We also explore the aspect ratio of the ellipse in the label overlap force. The aspect ratio, or the parameter b defined in Sec. 3.2.2), denotes a wide elliptical collision force when $b > 1$ and a tall one when $0 < b < 1$. As shown in Fig. 9, the edge lengths can be preserved with a wide range of ellipse aspect ratios with the best compaction achieved when the aspect ratio is set to 5.

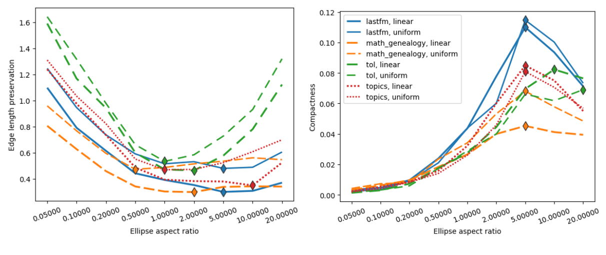


Fig. 9: Impact of label overlap force directions (ellipse aspect ratio) on the two quality metrics: Edge length (a) and compactness (b). Diamonds mark the best parameter setup for the specific network.

Distribution and Node-Edge Forces: The distribution force and the node-edge force play a larger role with the compactness initialization, and we experimentally determine suitable strengths with Alg. `Compact-Initialization`. As shown in Fig. 10, smaller force strength leads to better edge length preservation and better compactness for all datasets, and so we set $S_d = 0.003$

We also explore the strength of node-edge force defined in Sec. 3.2.5. As shown in Fig. 11, both edge lengths preservation and compactness are better with a weak node-edge force. Setting node-edge force strength $S_ne = 0.1$ seems to provide a reasonable balance between the two metrics.

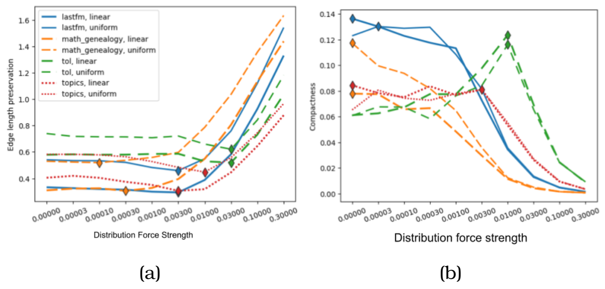


Fig. 10: Impact of distribution force on the two quality metrics: Edge length (a) and compactness (b). Diamonds mark the best parameter setup for the specific network.

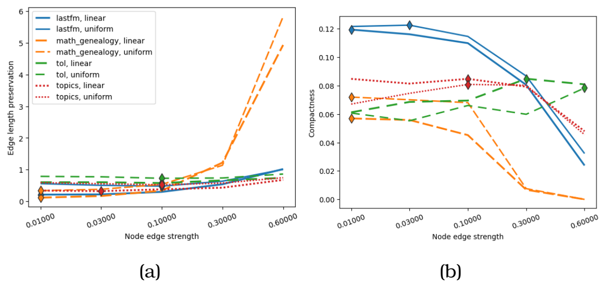


Fig. 11: Impact of node-edge force on the two quality metrics: Edge length (a) and compactness (b). Diamonds mark the best parameter setup for the specific network.

5.2 Number of Iterations

The force-directed improvement step of the algorithm and the final iteration perform iterative refinements; the number of iterations of each of these steps impacts the quality of the layouts and the runtime of the overall algorithm.

Number of Force-directed Iterations: The force-directed algorithm converges relatively quickly when initialized using Alg. `Compact-Initialization`. On the other hand, Alg. `Edge-Length-Initialization` needs more force-directed iterations to remove all label overlaps. Hence we analyze the impact of the number of iterations when initializing with Alg. `Edge-Length-Initialization`. As illustrated in Fig. 12a, early iterations remove many overlaps, with diminishing returns after 40-50 iterations. On the other hand, the running time increases with the number of iterations as shown in Fig. 12b. Hence, we set the number of iterations equal to 50.

Final Iteration: Since the force-directed improvement step terminates after 50 iterations, it may not have removed all overlaps. The final iteration step enforces the no-overlaps constraint (C2) by removing any remaining overlaps. Its performance depends on two parameters: the number of samples and the width of the square sample area. As illustrated in Fig. 13a, the running time increases as the number of samples increases. We can reduce the number of samples by tuning the width of the sample area. We denote the width of the sample area by the percentage of the minimal square box that contains the drawing. As illustrated in Fig. 13b, as the width of the sample area increases, the number of needed samples is

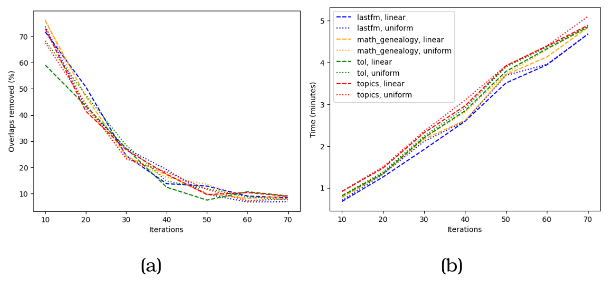


Fig. 12: Illustrating the impact of the number of iterations on overlaps and time. (a) shows the percentage of overlaps by iterations and (b) shows the time by the number of iterations. We choose a value to balance the two considerations.

reduced. Hence, we set the sample width to $0.01 - 0.02\%$ of the total area and the number of samples to 20.

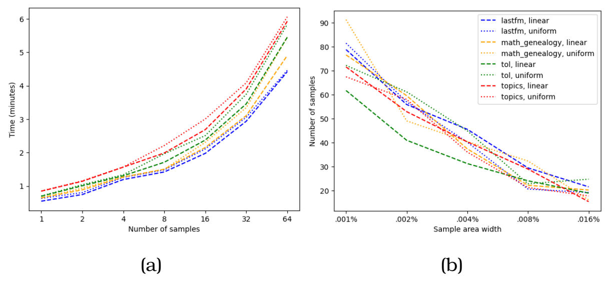


Fig. 13: Illustrating the impact of the number of samples and sample area width of the final overlap removal algorithm. (a) shows the running time by the number of samples. (b) shows the width of the sample area for the number of samples needed. We choose a value that allows for fewer samples to reduce the runtime.

6 Evaluation

In this section, we evaluate different algorithms using four real-world datasets. We denote the RT algorithm by RT_L and RT_C when initialized by Alg. `Edge-Length-Initialization` and Alg. `Compact-Initialization`, respectively. Similarly, we denote the PRT algorithm by PRT_L and PRT_C when initialized by Alg. `Edge-Length-Initialization` and Alg. `Compact-Initialization`, respectively.

Prior Methods: As there are no prior algorithms that guarantee the two constraints (no crossings, no overlaps), while optimizing desired edge lengths and compactness, it can be somewhat unfair to compare against prior approaches. Nevertheless, with some careful modifications (and clarifications) we can use existing tree/network layout algorithms in a comparison. We chose two such algorithms as described below.

GraphViz [19] can efficiently lay out a given tree or network with sfdp [29], label the nodes and then remove overlaps via the PRoxImity Stress Model (PRISM) [24]. Note that the output does not optimize given edge lengths and does not guarantee that trees are drawn in a crossings-free manner. We denote this by sfdp+p.

The yED [53] system provides several methods that can draw trees without edge crossings and optimize compactness. Note that yED does not optimize edge lengths and the only way to remove label overlap is to scale the drawing area. We use the *Circular Layout* (CIR) in *yED* as it produces the most compact layouts.

To provide a fair comparison, we consider two settings for the evaluation: one in which we have different desired edge lengths and the other one considers uniform edge lengths (to make it possible to compare with sfdp+p and CIR).

Datasets: We extract seven networks from four datasets.

Last.FM Network [23], extracted from the last.fm Internet radio station with 2588 nodes and 28221 edges. The nodes are popular musical artists with weights corresponding to the number of listeners. Edges are placed between similar artists, based on listening habits.

Google Topics Network [13] has as nodes research topics from Google Scholar's profiles, with weights corresponding to the number of people working on them. Edges are placed between pairs of topics that co-occur in profiles. We work with two versions of this network: one with $34,741$ nodes and $646,565$ edges, and the other a smaller subset with 5001 nodes.

Tree of Life[1], extracted from the tree of life web project. This dataset contains a node for every species, with an edge between two nodes representing the phylogenetic connection between the two. We work with two versions of this network: one with $35,960$ nodes, and the other with $2,934$ nodes.

Math Genealogy Network[2], every node represents a mathematician with edges capturing (advisor, advisee) relationship. We work with two versions of this network: one with $257,501$ nodes, and the other with $3,016$ nodes.

Dataset Processing: We compute the multi-level Steiner tree as described in Sec. 7. The details about the terminal selection method, number of levels, and desired edge lengths are provided in that section. We assign edge lengths, increasing linearly as we go from the lowest level to top. Note that we can equivalently consider such a multi-level tree as a single-level tree with different edge lengths. Specifically, a multi-level tree has a hierarchical structure: all the nodes and edges of a particular level are also present in the lower levels. Hence, for each edge, we consider the highest level where the edge is present. We assign the desired edge length of this edge to the edge length of that level.

With this in mind, we compare the performance of the four variants of our algorithm using a single-level tree, where all edges are present and have different desired edge lengths.

Since the sfdp+p and CIR methods cannot handle different edge lengths, when comparing them to our algorithm, we consider the setting where the trees are given as above, but the edge lengths are uniform.

Quantitative Evaluation: We measure the optimization goals: desired edge length preservation and compactness, as well as the runtime.

Desired Edge Length (DEL): evaluates the normalized desired edge lengths in each layer. Given the desired edge lengths $\{l_{ij} : (i,j) \in E\}$, defined in Sec. 7, and coordinates of the nodes X in the computed layout, we evaluate DEL with the following formula:

$$\mathrm{DEL} = \sqrt{\frac{1}{|E|}\sum_{(i,j)\in E}\left(\frac{||X_i - X_j|| - l_{ij}}{l_{ij}}\right)^2} \tag{1}$$

[1]http://tolweb.org/tree/
[2]https://genealogy.math.ndsu.nodak.edu/

Network			Desired Edge Length ↓						Compactness ↑						Runtime (sec) ↓								Crossings ↓	
Name	$\|V\|$	Edge Length	CIR	sfdp+p	RT_L	RT_C	PRT_L	PRT_C	CIR	sfdp+p	RT_L	RT_C	PRT_L	PRT_C	CIR	sfdp+p	RT_L	RT_C	PRT_L	PRT_L*	PRT_C	PRT_C*	sfdp+p	others
Last.FM	2,588	uniform	2.06	0.56	**0.18**	0.45	0.19	0.42	3e-7	0.04	0.01	**0.13**	0.02	0.11	**5**	8	926	309	54	15	26	7	147	**0**
		linear	-	-	**0.13**	0.45	0.15	0.43	-	-	0.01	**0.07**	0.02	0.05	-	-	1635	233	55	15	31	**9**	-	**0**
Topics	5,001	uniform	1.93	0.72	**0.27**	0.38	0.28	0.36	7e-6	0.03	0.01	**0.05**	0.02	0.04	44	**22**	3997	1000	175	32	142	23	203	**0**
		linear	-	-	**0.14**	0.30	0.16	0.28	-	-	0.01	**0.05**	0.01	0.04	-	-	8456	1096	178	32	147	**25**	-	**0**
Tree of Life	2,934	uniform	1.24	0.53	0.43	0.49	**0.42**	0.47	6e-6	**0.09**	0.01	**0.09**	0.02	0.08	**7**	**7**	3157	333	62	36	43	21	134	**0**
		linear	-	-	**0.45**	0.47	0.46	0.47	-	-	0.01	**0.10**	0.01	0.07	-	-	1178	698	61	55	49	**26**	-	**0**
Math Genealogy	3,016	uniform	3.15	0.78	**0.21**	0.40	0.23	0.34	3e-6	**0.13**	0.02	0.02	0.02	0.02	**5**	6	2067	774	57	14	46	11	1508	**0**
		linear	-	-	**0.29**	0.34	0.31	0.36	-	-	0.02	**0.03**	0.02	**0.03**	-	-	1340	563	64	14	51	**12**	-	**0**
Topics (large)	34,758	uniform	3.88	**0.76**	-	-	0.84	0.91	3e-6	**0.005**	-	-	0.001	0.004	1259	**258**	-	-	9346	2925	8375	2658	7713	**0**
Tree of Life (large)	35,960	uniform	2.17	**0.83**	-	-	1.29	1.36	3e-6	**0.006**	-	-	0.001	**0.006**	792	**455**	-	-	8780	2184	7194	1856	12088	**0**
Genealogy (large)	100,347	uniform	-	0.88	-	-	**0.83**	0.87	-	**0.007**	-	-	0.002	0.006	-	**2041**	-	-	-	8689	-	7675	27478	**0**

Table 1: Quantitative algorithmic comparison using desired edge length preservation, compactness and runtime for different algorithms. The ↓ next to a criterion indicates lower scores are better and ↑ indices that higher scores are better. PRT_L* and PRT_C* refer to the experiments run on the Skylark server, while PRT_L and PRT_C refer to the experiments run on the laptop.

This measures the root mean square of the relative error, producing a positive number, with 0 corresponding to perfect preservation.

Compactness Measure (CM): measures the ratio between the total areas of labels (the minimum possible area needed to draw all labels without overlaps) and the area of the actual drawing (measured by the area of the smallest bounding rectangle). CM scores are in the range $[0, 1]$, where 1 corresponds to perfect area utilization, this measure is the fourth found in the paper [39].

$$\mathtt{CM} = \frac{\sum_{v \in V} \text{label\_area}(v)}{(X_{max,0} - X_{min,0})(X_{max,1} - X_{min,1})} \tag{2}$$

Results: We evaluate the performance of all algorithms on seven trees extracted from the four datasets above. Our two RT variants (algorithms RT_L and RT_C) are applied only to the 4 small trees. The two PRT variants, sfdp+p, and CIR are applied to all 7 trees. Note that for each of the 4 small trees we consider two different desired edge lengths: the *linear edge setting* used to drive the interactive, zoomable visualization, and the *uniform edge setting* which makes is possibe to compare our algorithms to the prior ones (sfdp+p and CIR). We show all Last.fm trees layouts (with the uniform edge length setting) in Fig. 2. We provide additional layouts in the supplementary materials. These figures highlight some significant differences which stand out visually, which we discuss in Sec. 8.

We provide all quantitative data in Table 1. From these results, we can see that in the *linear edge length setting* RT_L does best in all instances. As expected, PRT_L provides similar scores. When looking at the compactness measure, RT_C performs best and, again, the parallel variant PRT_C performs nearly as well. Blank entries in the Desired Edge Length or Compactness scores indicate that the experiment was not performed for this setting (e.g., the linear setting for sfp+p and CIR) or the computation did not terminate by the maximum time limit of 8 hours (e.g., CIR with the large Math Genealogy tree).

Overall, the RT variants provide slightly better results than the PRT variants, both in edge length preservation and in compactness. We believe this due to RT utilizing the fine-tuned force-directed algorithm in d3.js, while the PRT force-directed algorithm is our own, and not as well tuned.

The expected, the PRT variants are usually more than an order of magnitude faster than the RT variants. This is due to parallelizing the underlying computations and implementation in C++, which is faster than d3.js. Note that we report four parallel runtimes: PRT_L, PRT_C (running on a laptop), and PRT_L*, PRT_C* (running on a server). Blank entries in the runtime columns indicate that the algorithm did not terminate by the maximum time limit of 8 hours.

Recall that the *uniform edge length setting* for the small networks makes it possible to compare the prior algorithms (sfdp+p and CIR) against PR and PRT. Here either RT_L or PRT_L performs best in desired edge length preservation and the scores of the two algorithms are similar. On the other hand, RT_C and PRT_C performs well in compactness, although sfdp+p outperforms our algorithms in some instances (at the expense of edge crossings). The CIR method is the fastest in most of the instances (at the expense of both compactness and edge lengths), and the running times of sfdp+p and PRT_C* are comparable.

For the 3 large networks, we only consider the uniform edge setting. The desired edge length scores of sfdp+p and PRT_L are comparable, with sfdp+p outperforming PRT in two of the three cases. Similarly, sfdp+p and PRT_C are the best in compactness. sfdp+p is the clear winner in speed, at the expense of thousands of edge crossings; see last two columns of Table 1.

7 Multi-Level Interactive Visualization

The focus of this paper is the algorithmic framework for creating readable tree layouts. However, on a desktop, laptop, or phone screen, even *viewing* a large tree with thousands of labeled nodes, requires more than a good layout.

With this in mind, we process all trees and their layouts further, to provide an interactive visualization environment. The idea is to create a hierarchy of trees, starting with the input and extracting progressively smaller trees that capture more and more abstract views, recalling interactive geographic maps. Important nodes (like large cities) and edges (like highways) are present in all representation levels. Less important nodes (like smaller towns) and edges (like smaller roads) appear when zooming in. We create such a hierarchy using multi-level Steiner trees and combining the results with the readable tree layout and an interactive map-like environment. Note that this approach is applicable to arbitrary networks and not just trees.

7.1 Multi-Level Steiner Trees

A Steiner tree minimizes the total weight of the subtree spanning a given subset of nodes (called the terminals). The multi-level Steiner tree problem is a generalization of the Steiner tree problem where the objective is to minimize the sum of the edge weights at all levels. As both problems are NP-hard, we use approximation algorithms that have been shown to work well in practice [1].

Formally, given a node-weighted and edge-weighted network $G = (V, E)$, we want to visualize G with the aid of a hierarchy of progressively larger trees $T_1 = (V_1, E_1) \subset T_2 = (V_2, E_2) \subset \cdots \subset T_n = (V_n, E_n) \subseteq G$, such that $V_1 \subset V_2 \subset \cdots \subset V_n = V$ and $E_1 \subset E_2 \subset \cdots \subset E_n \subset E$.

We use multi-level Steiner trees in order to make the hierarchy representative of the underlying network, based on a node filtration $V_1 \subset V_2 \subset \cdots \subset V_n = V$ with the most important nodes (highest weight) in V_1, the next most important nodes added to form V_2, and so on. A solution to the multi-level Steiner tree problem then creates the set of progressively larger trees $T_1 = (V_1, E_1) \subset T_2 = (V_2, E_2) \subset \cdots \subset T_n = (V_n, E_n) \subseteq G$ using the most important (highest weight) edges.

7.2 Dataset Processing

Many real-life networks come with well-defined notions of the importance of nodes and edges. Node importance can determined by the number of listeners for a specific band in the Last.FM network, or by the number of researchers of a specific topic in the Google Topics network. In the absence of such node information, importance can be computed based on structural properties of the network, such as degree centrality, eigenvector centrality, etc [8]. Similarly, edge importance can be given (e.g., number of listeners of a pair of bands in the Last.FM network, number of researchers listing a pair of topics in the Google Topics network), or can be computed based on structural properties (e.g., betweenness centrality, PageRank).

Node Weights: The Last.FM network and the topics network have node weights (number of listeners and number of researchers, respectively) and we use the heavy nodes in the higher levels and lighter nodes in the lower levels. For the tree of life and math genealogy networks, we set the node weight equal to the node degree (degree centrality).

We select the terminals of the multi-level Steiner tree instance according to the node weights: higher-level terminal sets contain heavy nodes, since the larger the weight is the more important the node is. The size of the terminal sets grows linearly. If we have n nodes and h levels then the top terminal set contains the most important n/h nodes. The next terminal set contains all the nodes of the top terminal set, together with the next n/h important nodes. We continue the process until the bottom level, when all nodes are present.

Edge Weights: Edge weights in the Last.FM network are given by the number of listeners of the corresponding pair of bands. Similarly, edge weights in the Google Topics network are the number of researchers listing the corresponding pair of topics. Edge lengths in the Tree of Life network are given by phylogenetic distance between the two endpoints. The math genealogy network is unweighted, and we use uniform edge lengths.

Desired Edge Length Settings: The edge weighs described above can be used desired edge lengths by computing the reciprocal of the original edge weights (converting similarities into dissimilarities). We have used this setting in experimenting with our algorithms but this setting is not what we report in the paper as it does not lend itself to semantic zooming and prior methods cannot handle desired edge lengths.

The simplest desired edge length setting used in the paper is the *uniform edge length setting*. We need such a setting to be able to compare the performance of our algorithms against those of prior methods (sfdp+p and CIR) that do not take edge lengths into account.

We can use the given edge weights and combine them with the multi-level Steiner tree solution to create edge lengths that work well with semantic zooming. This is the *linear edge setting* used in some of the experiments discussed in the paper. While in the uniform setting edge lengths are set to 200 in the linear setting the edge lengths depend on the level on which the edge first appears in the multi-level Steiner tree (which in turn depends on the underlying edge weights). The desired edge length on the lowest level is $l_{min} = 200$ and this value is increased by l_{add} each time we got to a higher level. In our examples, l_{add} varies depending on the number of levels. We use different number of levels for different networks, with more levels for larger networks. For the smallest dataset, the Last.FM network, we have only 8 levels while for the largest dataset, the math genealogy tree, there are more than 100 levels.

7.3 Map-like Visualization

From the tree layout, we generate a map using the GMap system [22]. The map generation method depends on a clustering step, and by default we use the MapSets [33] clustering technique. Our system uses OpenLayers [51] with zooming and panning provided via buttons, mouse scrolling, or through the mini-map.When viewing level i, all nodes at this or higher levels are labeled and there are no label-overlaps. Edge widths are determined based on their levels: higher level edges are thicker, lower level edges are thinner. A search bar allows for direct queries with auto-complete suggestions and clicking on a search result recenters the map on the selected node. By default we show labels with at most 16 characters (truncating longer ones) but the full label is shown on a mouse-over event. Node attributes and edge attributes are provided when clicking on the node/edge. An example of this visualization can be seen in Fig. 1.

8 Discussion

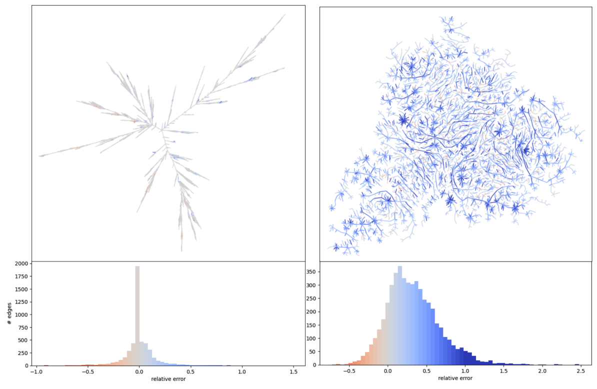
Comparing our algorithms shows that they do well in their respective optimization goals. Table 1 confirms that the RT_L and PRT_L algorithms outperform the other two in desired edge length preservation, RT_C and PRT_C algorithms outperform the other two in compactness, and PRT_C* is the fastest of the variants.

Comparing the new algorithms to the two prior ones (only possible when using uniform edge lengths) also seem encouraging, even though sfdp+p introduces crossings in all layouts and CIR has compactness scores that are orders of magnitude worse:

- RT_L and PRT_L outperform sfdp+p w.r.t. edge lengths on all 4 small networks
- RT_C and PRT_C outperform sfdp+p w.r.t. compactness on 2/4 small networks, and they are almost equal in another dataset
- RT_L and PRT_L outperform CIR w.r.t. edge lengths on all 4 small networks
- RT_C and PRT_C outperform CIR w.r.t. compactness on all 4 small networks
- PRT_L* and PRT_C* are slower than both sfdp+p and CIR on the large networks, but not by much.

8.1 Qualitative Analysis

Here we take a closer look at the results returned by the new algorithms and the two prior ones. Fig. 2 shows the results of last.FM networks. As the name implies, circular layout (CIR) wraps branches of the tree into spiraling circles to form a compact layout. Edges close to the center of the tree are stretched in order to provide large areas for the subtrees, resulting in poor edge length preservation. Since leaves are drawn in small regions, the overall layout must be scaled a lot in order to show the labels without overlaps, yielding poor compactness. The sfdp+p results are consistently good in compactness, at the expense of many crossings. Our new methods are better at capturing the global structure.



(a) RT_L (b) RT_C

Fig. 14: Analysis of failure modes in desired edge length preservation. **Top left:** In the layout of the topic network computed by RT_L, most edges are drawn with their desired edge lengths. **Bottom left:** A histogram of relative errors in desired edge length preservation, colored in the same way as the layout. **Right**: The same analysis on the RT_C algorithm, showing RT_C stretches (blue) more edges to improve compactness.

Next, we look more closely at the layouts obtained from RT_L and RT_C. First, we focus on desired edge length preservation. Fig. 14 colors individual edges in the layout by their relative error. Recall that we use the relative error to measure the desired length preservation in Eqn. 1. For each edge $(u, v) \in E$, it measures the discrepancy between the actual edge length $||X_u - X_v||$ in the drawing and the given desired edge length l_{uv}: relative_error(u,v) = $(||X_u - X_v|| - l_{uv})/l_{uv}$.

With RT_L, we observed from the left layout and histogram in Fig. 14 that most edges are drawn with their desired edge lengths due to its edge-length guided initialization. We can see a few stretched or compressed edges (colored in blue and red in Fig. 14), but since most of the other edges are drawn near perfectly in terms of edge length, the drawing has a low variation in relative error. The errors are larger in RT_C, which seems to be due to the way label overlaps are handled: in dense regions, e.g. around high degree nodes, the edges are more likely to be stretched, whereas on the periphery the edge lengths are better preserved.

Next we look at the compactness of the two layouts. Consider the difference between the two algorithms in their rendering of the region around the 'Artificial Intelligence' node in the maps, as shown in Fig. 15. From the layout overview in Fig. 15, we can already see that RT_L uses space less efficiently, it has more empty, white space. The 'Artificial Intelligence' node, highlighted in blue in Fig. 15, is close to multiple heavy subtrees such as those from the nodes 'natural language processing', 'machine learning', and 'computer vision'. RT_C distributes the heavy trees evenly, due to its initial layout. RT_L, on the other hand, places most heavy branches on the left side, resulting in a less efficient use of drawing area.

8.2 Scalability

We experimented further with the PRT algorithm to evaluate how it behaves with larger number of cores and with larger number of nodes, shown in Fig. 16(a) and Fig. 16(b), respectively. In Fig. 16(a), we show strong scaling results for the Tree of Life and Math Genealogy datasets. For both

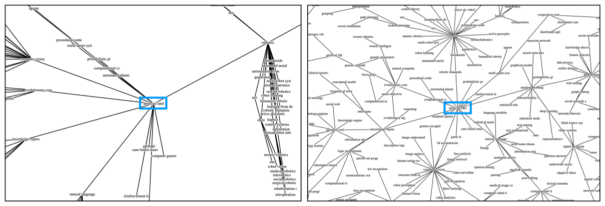


Fig. 15: Analysis of failure modes in layout compactness. **Left:** The topic network layout returned by RT_L has more unused space and fails to distribute heavy subtrees around the 'Artificial Intelligence' node (pointed and circled in blue) evenly. **Right:** RT_C generates a more balanced layout and is able to distribute subtrees more evenly.

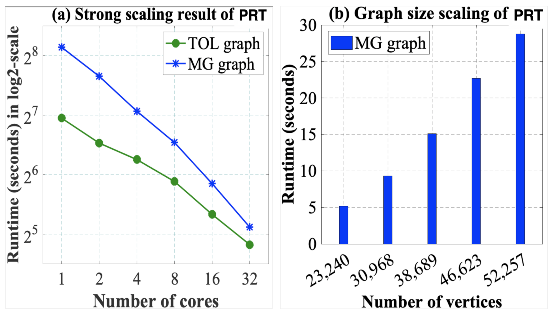


Fig. 16: (a) Strong scaling results of Tree of Life (TOL) dataset (35,960 nodes) and Math Genealogy (MG) dataset (52,257 nodes). (b) Network scaling results for different size of MG trees using 48 cores. Only per-iteration runtime is reported.

datasets, we observe that the runtime decreases almost linearly as the number of cores increases. In Fig. 16(b), we report the per-iteration runtime on the Math Genealogy tree when increasing the size of the trees. We observe that the runtime increases almost linearly with the size of the tree. This provides support for the scalability of the PRT. It is notable that while PRT does not outperform RT, PRT's numbers are within a small constant factor of the best values.

Experimental Environment: We conducted most the experiments on a laptop, except one set of experiments which used a Skylake server machine (indicated by PRT_L* and PRT_C*). The laptop is configured with MacOS, 2.3 GHz Dual-Core Intel Core i5, 8GB RAM, and 4 logical cores. The server is configured with Linux OS, Intel(R) Xeon(R) Platinum 8160 CPU @ 2.10GHz, 256GB RAM, 2 sockets, and 24 cores per socket.

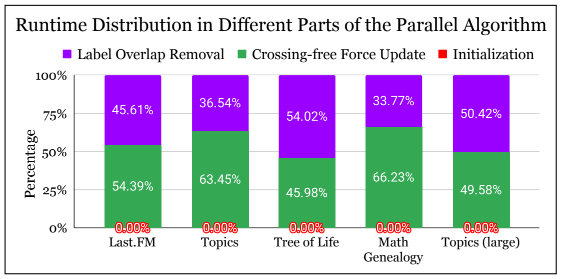


Fig. 17: Runtime distribution of different steps of the parallel PRT Algorithm for several input trees.

Runtime Analysis of the PRT Algorithm

There are three main steps in the PRT algorithm: (i) Initialization, (ii) Parallel Force-directed Improvement of Layout, and (ii) Parallel Label Overlap Removal. In Fig. 17, we report the percentage of total runtime spent in the different steps for several input trees. In all cases, we observe that the initialization step takes little time (less than a millisecond). This is expected since we run this step only once and the most time-consuming part of this step (finding center node by Alg. `PRT-Center-Node`) is fully parallelized. On the other hand, we iteratively run the crossing-free force update (parallel force-directed improvement) and label overlap removal steps multiple times. The force update and label overlap removal steps consume almost 50% of the total runtime for the large Topics tree, respectively.

9 Limitations

While we attempted to compare our readable tree layout framework with prior algorithms, we only compared against two. Similarly, we used only four real-world datasets and seven trees extracted from them for our evaluation. Further experiments with different types of trees, and with synthetically generated trees that test the limits of the prior and proposed methods (e.g., with respect to balance, degree distribution, diameter, etc.) are needed.

The utility of crossings-free, compact layouts that capture pre-specified edge lengths and show all node labels without overlaps needs to be evaluated. While intuitively these seem like desirable features (non-overlapping labels make for readable layouts, non-crossing layouts help grasp the underlying structure, compact layouts require less panning and zooming), a human subjects study can further validate these goals.

We use a simple overlap removal technique to improve the compactness of the algorithm. Overlap removal is a well-studied problem and there are several advanced algorithms [24, 38, 41]. Modifying such algorithms to ensure they do not change the topology of the current layout (e.g., by introducing edge crossings) remains an interesting open problem.

Since we optimize compactness, our algorithms might create zigzag-like paths, making it difficult to quickly estimate paths lengths and to compare different paths. Balancing the need for compactness with such distortions requires further examination.

Force directed algorithms often have a large parameter space. We add on to this with parameters from two initial layouts and a final overlap step. This gives us a very large parameter space. While we have done some small-scale parameter space search, more careful and detailed analysis will likely yield better results.

10 Conclusions

Both the quantitative evaluation and the visual analysis provide evidence of the utility of the proposed Readable Tree (RT) framework. The parallel version (PRT) makes the framework applicable to large instances, without much degradation in quality (edge length preservation and compactness).

Comparing RT and PRT with sfdp+p and CIR also seems encouraging. On all 4 small networks RT_L outperforms sfdp+p and CIR in edge length preservation; RT_C outperforms CIR and sfdp+p in compactness in most cases; PRT is slower than both sfdp+p and CIR, but not by much, especially given that sfdp+p introduces crossings in all layouts and CIR has compactness scores that are orders of magnitude worse.

Even though that layout of trees is a well-known and arguably solved problem, the *readable tree layout problem* shows that there is more work to be done in this domain. We propose an algorithmic framework for creating readable tree layouts that guarantee crossings-free layouts with non-overlapping node labels. Our RT algorithm works well on smaller networks and can be executed on any computer, and our PRT algorithm speeds up the computation making it applicable to larger networks, but relies on more advanced hardware. The utility of such algorithms goes beyond drawings of trees, to providing interactive exploration of large neoworks, as illustrated by several examples and a video on the project website[3]. All source code, datasets, and analysis can be found at https://github.com/abureyanahmed/multi_level_tree.

Acknowledgments

This work was supported in part by NSF grants CCF-1740858, CCF-1712119, and DMS-1839274.

References

[1] R. Ahmed, P. Angelini, F. Sahneh, A. Efrat, D. Glickenstein, M. Gronemann, N. Heinsohn, S. Kobourov, R. Spence, J. Watkins, and A. Wolff, "Multi-level Steiner trees," *ACM Journal of Experimental Algorithmics*, vol. 24, no. 1, pp. 2.5:1–2.5:22, 2019.

[2] A. Arleo, W. Didimo, G. Liotta, and F. Montecchiani, "A distributed multilevel force-directed algorithm," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 4, pp. 754–765, 2018.

[3] C. Bachmaier, U. Brandes, and B. Schlieper, "Drawing phylogenetic trees," in *ISAAC'05: Proceedings of the International Symposium on Algorithms and Computations*, ser. Lecture Notes in Computer Science, X. Deng and D. Du, Eds. Springer, 2005, pp. 1110–1121.

[4] C. J. Ballen and H. W. Greene, "Walking and talking the tree of life: Why and how to teach about biodiversity," *PLoS biology*, vol. 15, no. 3, p. e2001630, 2017.

[5] M. Bastian, S. Heymann, and M. Jacomy, "Gephi: An open source software for exploring and manipulating networks," in *International AAAI Conference on Web and Social Media*, 2009.

[6] A. Bavelas, "Communication patterns in task-oriented groups," *The journal of the acoustical society of America*, vol. 22, no. 6, pp. 725–730, 1950.

[7] R. Blanch, R. Dautriche, and G. Bisson, "Dendrogramix: A hybrid tree-matrix visualization technique to support interactive exploration of dendrograms," in *2015 IEEE Pacific Visualization Symposium (PacificVis)*. IEEE, 2015, pp. 31–38.

[8] P. Bonacich, "Some unique properties of eigenvector centrality," *Social networks*, vol. 29, no. 4, pp. 555–564, 2007.

[9] K. Börner, A. Maltese, R. N. Balliet, and J. Heimlich, "Investigating aspects of data visualization literacy using 20 information visualizations and 273 science museum visitors," *Information Visualization*, vol. 15, no. 3, pp. 198–213, 2016.

[10] M. Bostock, V. Ogievetsky, and J. Heer, "D^3 data-driven documents," *IEEE transactions on visualization and computer graphics*, vol. 17, no. 12, pp. 2301–2309, 2011.

[11] K. W. Boyack, R. Klavans, and K. Börner, "Mapping the backbone of science," *Scientometrics*, vol. 64, no. 3, pp. 351–374, 2005.

[12] U. Brandes and C. Pich, "Eigensolver methods for progressive multidimensional scaling of large data," in *Graph Drawing*, Springer. Springer, 2007, pp. 42–53.

[13] R. Burd, K. Espy, I. Hossain, S. Kobourov, N. Merchant, and H. Purchase, "GRAM: Global research activity map," in *Intl. Conference on Advanced Visual Interfaces (AVI)*. ACM, 2018, pp. 31:1–31:9.

[14] R. Chandra, L. Dagum, D. Kohr, R. Menon, D. Maydan, and J. McDonald, *Parallel Programming in OpenMP*. Morgan kaufmann, 2001.

[3] https://tiga1231.github.io/zmlt/demo/overview.html

[15] M. Chimani, C. Gutwenger, M. Jünger, G. W. Klau, K. Klein, and P. Mutzel, "The open graph drawing framework (OGDF)," *Handbook of Graph Drawing and Visualization*, pp. 543–569, 2011.

[16] J.-H. Choi, H.-Y. Jung, H.-S. Kim, and H.-G. Cho, "PhyloDraw: a Phylogenetic Tree Drawing System ," *Bioinformatics*, vol. 16, no. 11, pp. 1056–1058, 11 2000. [Online]. Available: https://doi.org/10.1093/bioinformatics/16.11.1056

[17] P. Eades, "A heuristic for graph drawing," *Congressus Numerantium*, vol. 42, pp. 149–160, 1984.

[18] P. Eades and N. C. Wormald, "Fixed edge-length graph drawing is np-hard," *Discrete Applied Mathematics*, vol. 28, no. 2, pp. 111–134, 1990. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0166218X9090110X

[19] J. Ellson, E. Gansner, L. Koutsofios, S. North, G. Woodhull, S. Description, and L. Technologies, "Graphviz — open source graph drawing tools," in *Lecture Notes in Computer Science*. Springer, 2001, pp. 483–484.

[20] T. M. J. Fruchterman and E. M. Reingold, "Graph drawing by force-directed placement," *Software: Practice and Experience*, vol. 21, no. 11, pp. 1129–1164, 1991.

[21] P. Gajer, M. Goodrich, and S. Kobourov, "A fast multi-dimensional algorithm for drawing large graphs," *Computational Geometry: Theory and Applications*, vol. 29, no. 1, pp. 3–18, 2004.

[22] E. R. Gansner, Y. Hu, and S. Kobourov, "GMap: Visualizing graphs and clusters as maps," in *2010 IEEE Pacific Visualization Symposium (PacificVis)*, 2010, pp. 201–208.

[23] E. Gansner, Y. Hu, S. Kobourov, and C. Volinsky, "Putting recommendations on the map: Visualizing clusters and relations," in *Proceedings of the Third ACM Conference on Recommender Systems*, ser. RecSys '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 345–348. [Online]. Available: https://doi.org/10.1145/1639714.1639784

[24] E. R. Gansner and Y. Hu, "Efficient node overlap removal using a proximity stress model," in *Graph Drawing*, I. G. Tollis and M. Patrignani, Eds. Berlin, Heidelberg: Springer, 2009, pp. 206–217.

[25] E. R. Gansner and S. C. North, "Improved force-directed layouts," in *Proceedings of the 6th International Symposium on Graph Drawing*, ser. Graph Drawing '98. Springer, 1998, pp. 364–373. [Online]. Available: http://dl.acm.org/citation.cfm?id=647550.729069

[26] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: Training imagenet in 1 hour," *arXiv preprint arXiv:1706.02677*, 2017.

[27] R. Hadany and D. Harel, "A multi-scale algorithm for drawing graphs nicely," *Discrete Applied Mathematics*, vol. 113, no. 1, pp. 3–21, 2001.

[28] S. Hadlak, H. Schumann, and H.-J. Schulz, "A survey of multi-faceted graph visualization." in *EuroVis (STARs)*, 2015, pp. 1–20.

[29] Y. Hu, "Efficient, high-quality force-directed graph drawing," *Mathematica Journal*, vol. 10, no. 1, pp. 37–71, 2005.

[30] L. A. Hug, B. J. Baker, K. Anantharaman, C. T. Brown, A. J. Probst, C. J. Castelle, C. N. Butterfield, A. W. Hernsdorf, Y. Amano, K. Ise *et al.*, "A new view of the tree of life," *Nature microbiology*, vol. 1, no. 5, pp. 1–6, 2016.

[31] C. Kittivorawang, D. Moritz, K. Wongsuphasawat, and J. Heer, "Fast and flexible overlap detection for chart labeling with occupancy bitmap," in *IEEE VIS Short Papers*, 2020. [Online]. Available: http://idl.cs.washington.edu/papers/fast-labels

[32] D. Kobak and P. Berens, "The art of using t-sne for single-cell transcriptomics," *Nature communications*, vol. 10, no. 1, pp. 1–14, 2019.

[33] S. Kobourov, S. Pupyrev, and P. Simonetto, "Visualizing Graphs as Maps with Contiguous Regions," in *EuroVis - Short Papers*, N. Elmqvist, M. Hlawitschka, and J. Kennedy, Eds. The Eurographics Association, 2014.

[34] Y. Koren, L. Carmel, and D. Harel, "Ace: A fast multiscale eigenvectors computation for drawing huge graphs," in *Information Visualization, 2002. INFOVIS 2002. IEEE Symposium on*. IEEE, 2002, pp. 137–144.

[35] Y. Y. Leow, T. Laurent, and X. Bresson, "Graphtsne: A visualization technique for graph-structured data," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[36] I. Letunic and P. Bork, "Interactive Tree Of Life (iTOL) v5: an online tool for phylogenetic tree display and annotation," *Nucleic Acids Research*, vol. 49, no. W1, pp. W293–W296, 04 2021. [Online]. Available: https://doi.org/10.1093/nar/gkab301

[37] M. Luboschik, H. Schumann, and H. Cords, "Particle-based labeling: Fast point-feature labeling without obscuring other visual features," *IEEE transactions on visualization and computer graphics*, vol. 14, no. 6, pp. 1237–1244, 2008.

[38] K. Marriott, P. Stuckey, V. Tam, and W. He, "Removing node overlapping in graph layout using constrained optimization," *Constraints*, vol. 8, no. 2, pp. 143–171, 2003.

[39] M. McGuffin and J. Robert, "Quantifying the space-efficiency of 2d graphical representations of trees," *Information Visualization*, vol. 9, pp. 115–140, 2010.

[40] K. Mote, "Fast point-feature label placement for dynamic visualizations," *Information Visualization*, vol. 6, no. 4, pp. 249–260, 2007.

[41] L. Nachmanson, A. Nocaj, S. Bereg, L. Zhang, and A. Holroyd, "Node overlap removal by growing a tree," pp. 33–43, 2016.

[42] Q. H. Nguyen, "INKA: an ink-based model of graph visualization," *CoRR*, vol. abs/1801.07008, 2018. [Online]. Available: http://arxiv.org/abs/1801.07008

[43] Q. V. Nguyen and M. L. Huang, "A space-optimized tree visualization," in *IEEE Symposium on Information Visualization, 2002. INFOVIS 2002.*, 2002, pp. 85–92.

[44] C. Nobre, M. Meyer, M. Streit, and A. Lex, "The state of the art in visualizing multivariate networks," in *Computer Graphics Forum*, vol. 38, no. 3, 2019, pp. 807–832.

[45] H. C. Purchase, D. Carrington, and J.-A. Allder, "Empirical evaluation of aesthetics-based graph layout," *Empirical Software Engineering*, vol. 7, no. 3, pp. 233–255, 2002. [Online]. Available: https://doi.org/10.1023/A:1016344215610

[46] K. Rahman, M. Sujon, and A. Azad, "Force2Vec: Parallel force-directed graph embedding," in *Intl. Conference on Data Mining (ICDM)*. IEEE, 2020, pp. 442–451.

[47] M. K. Rahman, M. H. Sujon, and A. Azad, "BatchLayout: A batch-parallel force-directed graph layout algorithm in shared memory," in *2020 IEEE Pacific Visualization Symposium (PacificVis)*. IEEE, 2020, pp. 16–25.

[48] E. M. Reingold and J. S. Tilford, "Tidier drawings of trees," *IEEE Transactions on Software Engineering*, vol. SE-7, no. 2, pp. 223–228, 1981.

[49] H. Schulz, "Treevis.net: A tree visualization reference," *IEEE Computer Graphics and Applications*, vol. 31, no. 6, pp. 11–15, Nov 2011.

[50] D. Sun and K. Wong, "On evaluating the layout of UML class diagrams for program comprehension," in *13th International Workshop on Program Comprehension (IWPC'05)*, 2005, pp. 317–326.

[51] T. O. D. Team, *OpenLayers*, 2020 (accessed December 22, 2020). [Online]. Available: https://openlayers.org/

[52] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne." *Journal of machine learning research*, vol. 9, no. 11, 2008.

[53] R. Wiese, M. Eiglsperger, and M. Kaufmann, "yfiles visualization and automatic layout of graphs," in *Graph Drawing Software*. Springer, 2004, pp. 173–191.

11 Supplementary Material

In this section, we provide additional example layouts from the different datasets, and that show a zoomed-in view of the images. We also discuss topology preservation using an example.

11.1 Evaluation

We now provide some images that show the outputs from different algorithms on different graphs. In Fig. 18, we give layouts of uniform Topics Graph computed by two existing algorithms (CIR and sfdp+p) and ours (RT_L and RT_C). In Fig. 20, we compare the layout of the Last.FM graph generated using RT_L and RL_C. Note that these overviews do not show details such as label overlaps, crossings, or compactness, but they do provide a feel for how the graphs are laid out. For instance, it is easy to see that CIR does not preserve edge lengths while SFDP+P appears to do well. However, when zooming in we see many crossings and label overlaps in SFDP+P. The RT_L is able to preserve desired edge lengths more which helps to capture the overall topology of the layout. However, if we zoom in then we can see that the drawing is not compact: there are some free spaces among the labels. On the other hand, RT_C focuses more on compactness and by zooming in the layout shows that relatively more labels are drawn compactly. Similarly, RT_L preserve the desired edge lengths while RT_C optimizes compactness for topics and tree of life graphs, see Fig. 21 and Fig. 22 respectively.

11.2 A Note on Topology Preservation

We discuss topology preservation in Sect 2, but here we illustrate the concept with an example. In Fig. 19 we can see that in order to realize the given edge lengths and draw the given small network we must introduce label overlaps (left). The label overlaps can be removed (right) but the topology of the layout is modified (the order of the nodes around each node is different) and this may introduce unnecessary crossings.

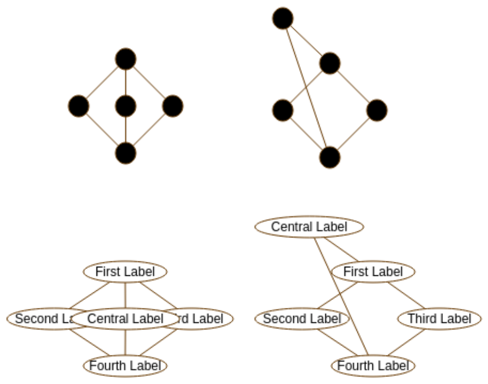
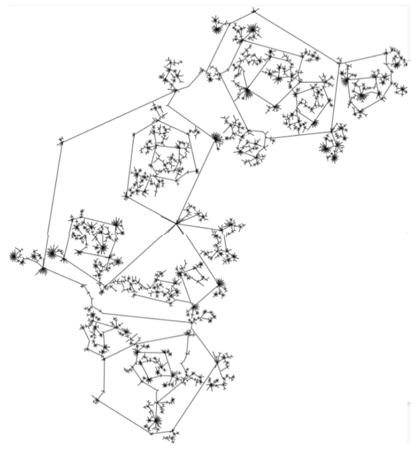
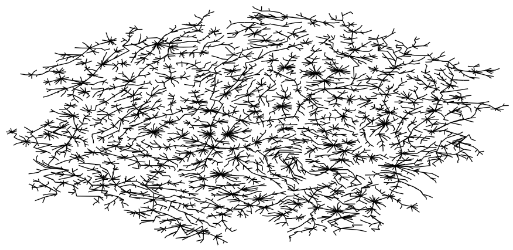


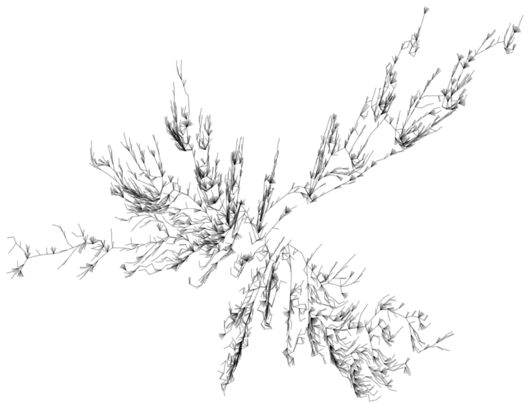
Fig. 19: An example of topology changing based on node labels



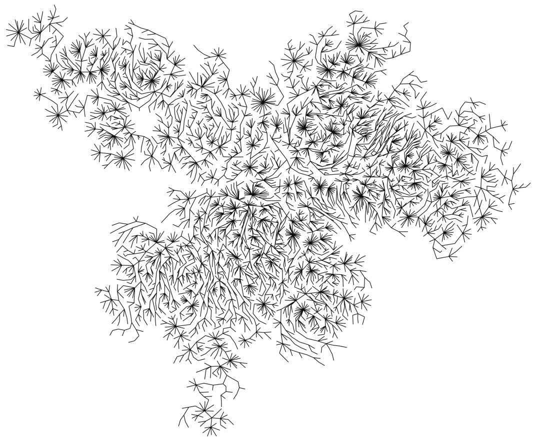
(a) CIR



(b) sfdp+p

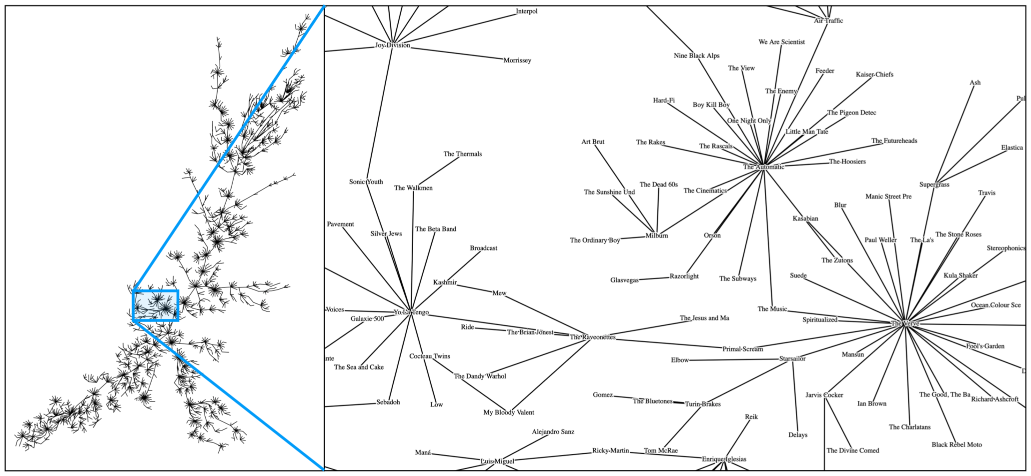


(c) RT_L



(d) RT_C

Fig. 18: Comparison of the tree layout structure of the uniform Google Topics graph drawn with CIR, sfdp+p, RT_L, and RT_C


(a) RT_L

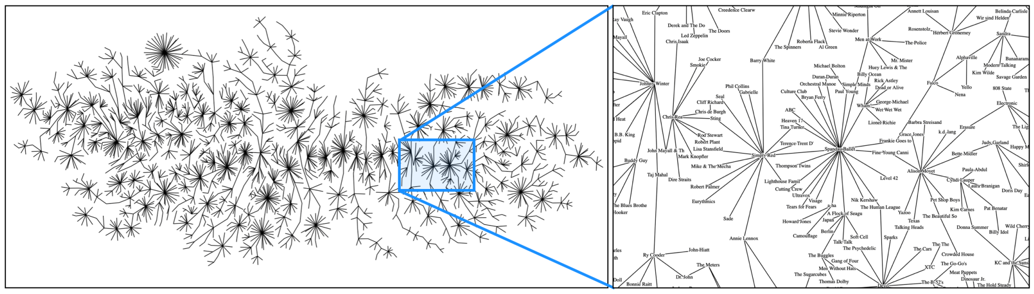
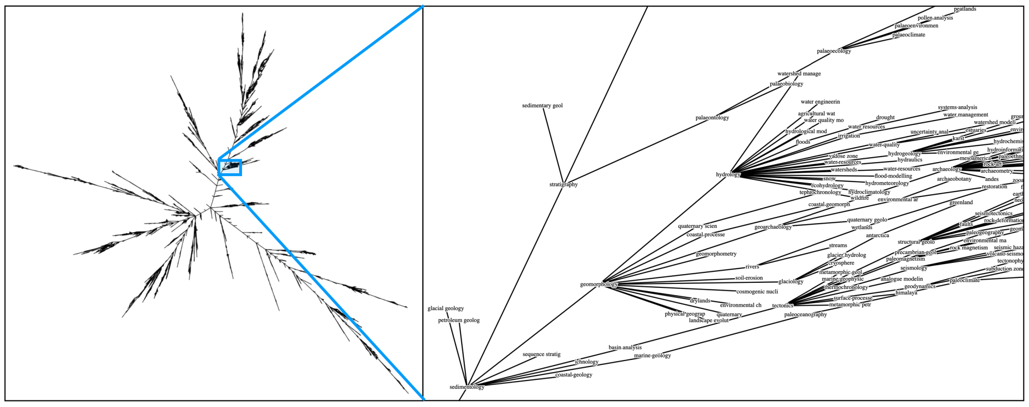
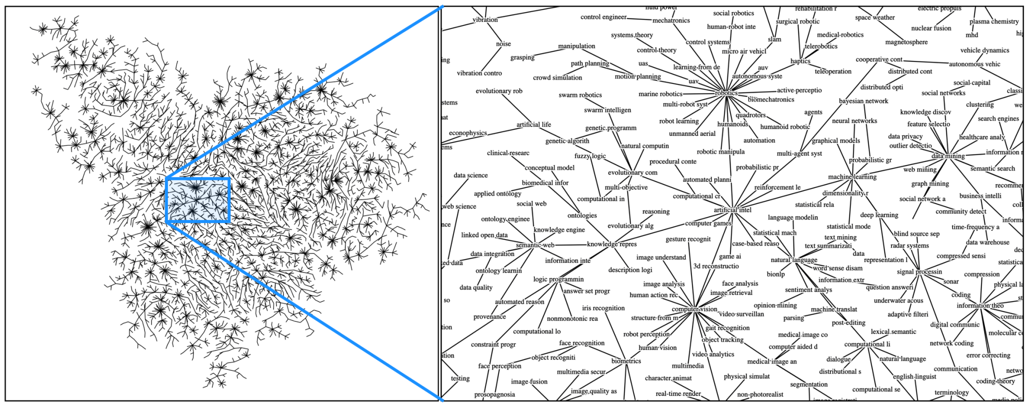

(b) RT_C

Fig. 20: Comparing Last.FM linear layouts drawn with our algorithms.

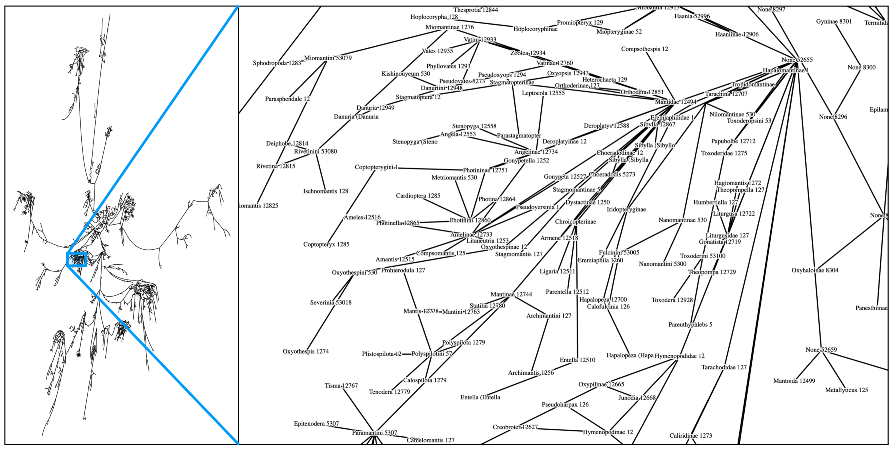


(a) RT_L



(b) RT_C

Fig. 21: Comparing Topics linear layouts drawn with our algorithms.


(a) RT_L

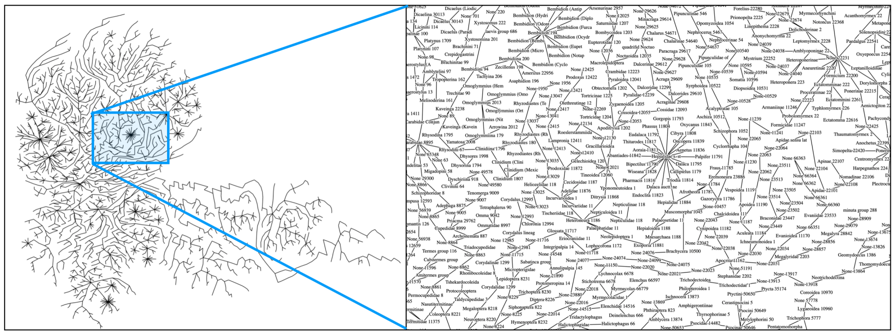

(b) RT_C

Fig. 22: Comparing Tree of Life linear layouts drawn with our algorithms.