EUROGRAPHICS 2021 / R. Borgo, G. E. Marai, and T. von Landesberger
(Guest Editors)

Volume 40 (2021), Number 3

Multi-level tree-based approach for interactive graph visualization

Kathryn Gray[1] Mingwei Li[1] Reyan Ahmed[1] Stephen Kobourov[1] Katy Börner[2]

[1]University of Arizona, USA, [2]Indiana University, USA

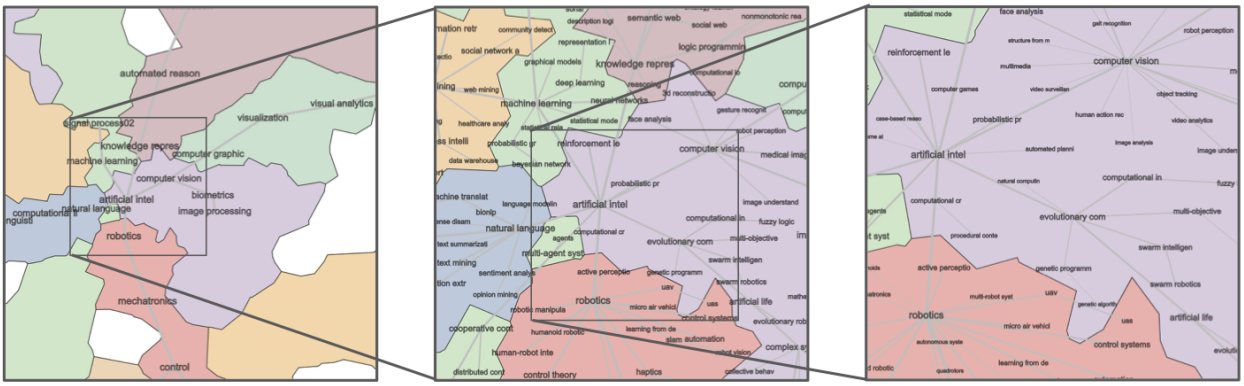


Figure 1: Map-based semantic-zoom with a multi-level tree that is representative, real, persistent, overlap-free labeled, planar, and compact.

Abstract

Human subject studies show that map-like visualizations are as good or better than standard node-link representations of graphs in terms of task performance, engagement, and memorization and recall of the underlying data. With this in mind, we propose Zoomable Multi-Level Tree (ZMLT) algorithms for multi-level, tree-based, map-like visualization of large graphs. We propose a set of seven desirable properties that such a visualization should maintain and two algorithms that fulfill them. These seven desirable properties are formalized as follows: (1) The abstract trees represent the underlying graph at a different level of details; (2) The embedded trees represent the underlying graph appropriately at different levels of details by using different edge lengths; (3) At every level of detail we show real nodes and real edges from the underlying graph; (4) If any node or edge appears in a given level, then they also appear in all deeper levels; (5) All nodes at the current level and higher levels are labeled and there are no label overlaps; (6) There are no edge crossings on any level; (7) The drawing area is proportional to the total area of the labels. The ZMLT algorithms are used along with clustering techniques to generate an interactive map. The functional prototype for this map can be found at http://uamap-dev.arl.arizona.edu:8086/. We evaluate the algorithms on two real-world datasets in terms of edge length realization and space utilization.

1. Introduction

Every day millions of people use maps to find locations such as restaurants, theaters, and shops, as well as the best routes to reach them. GoogleMaps, AppleMaps, and OpenStreetMaps are used widely and have trained many to read and use maps efficiently. In fact, most visitors to science museums can name and use maps effectively [BMBH16]. Online maps offer standard map interactions, such as zooming and panning: panning moves the map in any direction while keeping it at the same scale and zooming changes the scale of the visualization by adding or removing details for a specific area. An important feature of most map visualizations is that when zooming in or out, the information density remains roughly the same and the type of information (e.g., major cities and roads) also remains the same. However, cluster-based visualization approaches for large graphs, rely on meta-nodes (or super-nodes, cluster-nodes) and meta-edges (or edge bundling) instead of major,

real “nodes" and their key “edge" connections, and thus break the map metaphor. In order to take advantage of people’s map reading skills when exploring graphs, we propose the *Zoomable Multi-Level Tree* (*ZMLT*) approach for map-like visualization of large graphs, which realizes the seven desirable properties of a map-based semantic multi-level tree representation of a large graph:

1. **Appropriate representation**: the abstract trees represent the underlying graph appropriately at different levels of detail. This property ensures that the graphs at each level of detail are connected (each level is a tree) and that the trees represent the underlying graph well on each level. We use multi-level Steiner trees which provide appropriate representation [AAS*18].
2. **Appropriate layout**: the layout of the trees represents the underlying graph appropriately at different levels of detail. This property ensures that the global structure of the underlying graph is captured by the positions of the nodes in the layout, while also providing good local embedding.
3. **Real**: every node and every edge shown in every level of detail are “real". This is motivated by typical interactive maps, which always show real cities and real roads between them. This property prohibits meta-nodes and meta-edges, which have been shown to be difficult to read and interact with; e.g., see [WHBT14].
4. **Persistent**: if a node or edge appears at one level of detail, it will never disappear when zooming in further. This property ensures persistence of the semantic level of detail representation, as expected in multi-level map representations. This is a desirable property in interactive visualization; e.g., see [DNE08].
5. **Overlap-free**: all nodes at the current and higher levels are labeled and node label do not overlap. While abstract graph layouts consider each node as a point, the visualization should take into account that node labels must be readable. Overlap-free labeling is a necessary step in real-world visualization systems, e.g., see [KMWH20].
6. **Crossing-free**: there are no edge crossings on any level. Optionally, when visualizing graphs that are not trees, the deepest level is allowed to contain overlaps, as every edge could be shown at this level. Since we extract trees for each level, a layout without edge crossings is possible and desirable [Pur97].
7. **Compact**: the drawing area is proportional to the total area of the labels. The sum of the sizes of all node labels is the minimum area needed. However, this does not account for the space needed to show the edges. Layout compactness is a well-known desirable property; e.g., see [Ngu18]. This criterion disqualifies classic tree layout algorithms, as simply scaling the layout until all overlaps are removed results in very poor space utilization.

Formally, given a node-weighted and edge-weighted graph $G = (V,E)$, we want to visualize G as a set of progressively larger trees $T_1 = (V_1,E_1) \subset T_2 = (V_2,E_2) \subset \cdots \subset T_n = (V_n,E_n) \subseteq G$, such that $V_1 \subset V_2 \subset \cdots \subset V_n = V$ and $E_1 \subset E_2 \subset \cdots \subset E_n \subset E$. To make the trees representative of the underlying graph we rely on a multi-level variant of the Steiner tree problem [AAS*18, ASH*20], where we create the 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, etc. 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; see Figure 2. Note that extracting the best tree T_i that spans the node set V_i may require the use of nodes that are not in V_i, which are called Steiner nodes.

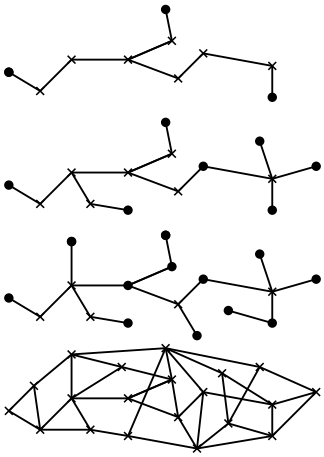


Figure 2: A multi-level Steiner tree: terminals are •, Steiner points are ×

We use these levels of trees in the visualization system. At a minimum level of zoom, only the most important cities and highways are visible, and these are provided by the tree T_1. Increasing the level of zoom brings in new cities and roads, and these are provided by the next tree in the filtration, until the maximum zoom level is reached and the whole map is shown.

We propose two algorithms to layout the entire underlying graph independent of Steiner tree hierarchy. The algorithms focus on providing an overlap free, crossing free, compact graph with an appropriate layout. We note that the overlap free and crossing free are hard constraints, any layout given must have these attributes. The compactness and appropriate layout are soft constraints, we define measurements for these and work to minimize them.

We generate the underlying map from the tree drawing by clustering adjacent nodes. We use MapSets [KPS14] to generate the map. Combining these tools, we provide a full visualization, including the standard zooming feature.

The seven guarantees above are embodied in the ZMLT layout, which is implemented and has a functional prototype for the interactive interface in a web browser; see Figure 1.

2. Related Work

Large Graph Visualization: Most graph layout algorithms use a force-directed [Ead84, FR91] or a stress model [BP07, KCH02] and provide a single static drawing. The force-directed model works well for small graphs, but does not scale for large networks. Speedup techniques employ a multi-scale variant [HH01, GGK04] or use parallel computation as in VxOrd [DWB01, BKB05]. GraphViz [EGK*01] uses the force-directed method sfdp [Hu05] that combines the multi-scale approach with a fast n-body simulation [BH86]. Stress minimization was introduced in the more

general setting of multidimensional scaling (MDS) [KW78] and has been used to draw graphs as early as 1980 [See80]. Simple stress functions can be efficiently optimized by exploiting fast algebraic operations such as majorization. Modifications to the stress model include the strain model [Tor52], PivotMDS [BP07], COAST [GHK13], and MaxEnt [GHN13]. Although these algorithms are fast and produce good drawings of regular grid-like graphs, the results are not as good for dense, or small-world graphs [BP09].

Graph layout algorithms are provided in libraries such as GraphViz [EGK*01], OGDF [CGJ*11], MSAGL [NRL08], and VTK [SAH00], but they do not support interaction, navigation, and data manipulation. Visualization toolkits such as Prefuse [HCL05], Tulip [AAB*17], Gephi [BHJ09], and yEd [WEK04] support visual graph manipulation, and while they can handle large graphs, their rendering often does not: even for graphs with a few thousand nodes, the amount of information rendered statically on the screen makes the visualization unusable.

Multi-Level Visualization: Research on interactive multi-level interfaces for exploring large graphs includes ASK-GraphView [AVHK06], topological fisheye views [AKY05, GKN05], and Grokker [RB03]. Software applications such as Pajek [DNMB11] for social networks, and Cytoscape [SMO*03] for biological data provide limited support for multi-level network visualization. Zinsmanier *et al.* [ZBDS12] proposed a technique to deal with visual clutter by adjusting the level of detail shown, using node aggregation and edge bundling.

Most of these approaches rely on meta-nodes and meta-edges, which make interactions such as semantic zooming, searching, and navigation counter-intuitive, as shown by Wojton *et al.* [WHBT14]. This work confirms that people have difficulty reading graph layouts that use hierarchical cluster representations (via high-level meta-nodes and meta-edges) such as "super-noding" and "edge bundling." Other studies that compare standard node-link representations with map-like ones show that map-like visualizations are superior in terms of task performance, memorization and engagement [SSKB14, SSKB15]. In this context, *GraphMaps* [MN18] visualizes a graph using the map metaphor and provides different zoom levels, driven by an embedding of the entire underlying graph. GRAM [BEH*18] also visualizes a graph using the map metaphor and provides different zoom levels but neither GraphMaps nor GRAM guarantees all seven of our desirable properties. For example, GRAM does not provide consistency or planarity. GRAM provides a button to show or hide the edges of the shown graph but the first option does not show the structure of the graph and the second one leads to a hairball; see Figure 3.

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. Overlapping labels pose a major problem for most graph layout algorithms, and severely affect the usability of the resulting visualizations. The standard approach to dealing with this problem is post-processing of the layout where node positions are perturbed to remove label overlaps. This usually results in a significantly modified layout with increased stress, larger drawing area, and with the introduction of crossings even when the starting node configuration is crossings-free.

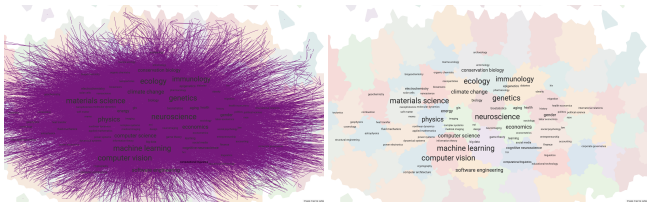


Figure 3: Screenshots from the GRAM system [BEH*18] with and without edges.

A simple solution to remove overlaps is to scale the drawing until the labels no longer overlap. This approach works on every layout and is straightforward, although it may result in an exponential increase in the drawing area. Marriott *et al.* [MSTH03] 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 [GN98], Gansner and Hu [GH09], and Nachmanson *et al.* [NNB*17] describe overlap-removal techniques with better aspect ratio and modest additional area. However, none of these approaches (except the straightforward scaling), can guarantee that no crossings are added when starting with a crossings-free input.

Placing nodes without overlaps has also been proposed [LSC08, Mot07, KMWH20], but these approaches do not guarantee crossing-free layouts of the underlying graphs.

Steiner Trees: The Steiner tree problem is one of Karp's original 21 NP-Complete problems [Kar72]. A polynomial time approximation scheme exists if the underlying graph is planar [BKMK07], but not for general graphs [CC08]. More importantly, the Steiner tree can be efficiently approximated within a factor of $\ln 4 + e < 1.39$ [BGRS10]. In the node-weighted Steiner tree problem, the cost of the tree is determined by the weights of the nodes included in the solution (rather than the weights of the edges). This version of the problem is also NP-hard [MR07] but can be approximated within a logarithmic factor [GK99, KR95]. The classic Steiner tree problem has also been generalized to a multi-level setting [BMM94, CNS04, CGNS08], where the terminals appear on different levels and must be connected by edges of appropriate levels, as described in the introduction. In particular, Ahmed *et al.* [AAS*18] showed that the standard (edge-weighted) Steiner tree problem can be efficiently approximated to within a constant factor. Similarly, Darabi *et al.* [DEK*18] described a multi-level version of the node-weighted Steiner tree, and showed that it can be approximated as well as the single-level version.

3. The Zoomable Multi-Level Drawing Algorithms

There are many graph drawing algorithms and systems, and there is also a great deal of work on visualizing trees; see *Treevis.net* [Sch11] which summarizes more than 300 techniques. However, none of these can guarantee the seven desirable layout properties that we need: appropriate representation, appropriate layout, real, persistent, overlap-free, crossing-free, and compact.

Five of the seven desirable properties are hard constraints: real, persistent, overlap-free, crossings-free. Two of the properties are soft constraints: layout appropriateness and compactness. In an appropriate multilevel map representation we should be able to distinguish global and local structures. To accomplish this we give each edge a *desired edge length (DEL)*. DELs are longer for the edges in the top levels of the hierarchy (corresponding to highways connecting big cities in a map) and shorter for edges on the bottom of the hierarchy (corresponding to local streets in a map). In our experiments, the hierarchies contain eight levels of Steiner trees and we set the desired edge length l_{uv} of an edge (u,v) to the value in the index of the maximum level of u and v in this list: $\{200,250,300,350,400,450,500,550\}$.

A key observation about crossing-free drawing is that maintaining the crossing free property is often easier than removing existing crossings. With this in mind, both algorithms initialize the layout with crossing-free drawings and iteratively improve them to optimize the remaining desirable properties.

We describe two different approaches to obtain tree layouts that fit in the Zoomable Multi-Level Tree (ZMLT) framework. Both of them preserve the hard constraints: real, persistent, overlap-free, and crossings-free, but differ in how they handle the soft constraints, edge length preservation and compactness. The first approach is *Desirable Edge Length Guided (DELG)*, and prioritizes realizing the given edge lengths. The second approach is *Compactness Guided (CG)* and optimizes space utilization.

3.1. Desirable-Edge-Length-guided (DELG) Algorithm

We begin by computing a tree layout without crossings that preserves the desired edge lengths. This layout is then iteratively refined to remove overlaps while ensuring good area utilization. The overall process has three steps as shown in Figure 4:

1. **DELG crossings-free initialization:** Initialize with a crossing-free layout that exactly preserves the desired edge lengths.
2. **Simultaneous DEL and Overlap-removal improvement:** Run a force-directed algorithm to remove label overlap without introducing crossings while preserving desired edge lengths.
3. **Final overlap removal:** Post process the layout to remove the remaining label overlaps and improve compactness.

Figure 4: The pipeline of the DELG algorithm.

3.1.1. DELG Crossings-free Initialization

We select a root node and assign an angular region (sector) to every child of this root node so that the assigned region is proportional to the size of the subtree of the child. We continue this process recursively; see algorithm 6 in the appendix. Since the angular regions are unbounded, the algorithm can realize all the desired edge lengths.

Algorithm 1: Angular Region-wise Initialization(r,θ)

Result: A layout which is crossing free and preserves desired edge length

Add the current node r;
Let n be the number of nodes in the tree;
for *each child v of r* **do**
  Let n_v be the nodes in the subtree;
  Angular region of $v, \theta_v = \theta \times \frac{n_v}{n}$;
  Initial layout(v, θ_v);
end

Algorithm 6 takes a root r as an input parameter. We use a center node of the tree as root, where the set of center nodes of a tree is either the middle node or middle two nodes in every longest path along the tree. Note that finding the longest path is NP-hard in general, but the problem can be solved in polynomial time for trees [CLRS01]. We place r at the origin of the 2D coordinate system. We denote the number of children of r by n_r. We denote the i-th child of r by v_i and their corresponding edge by $e_i = (r, v_i)$. We denote the desired length of e_i by l_i. We place v_1 at (l_1, θ_1) where θ_1 has a value between $0 - 2\pi$ (in the polar coordinate system). Placing each remaining child v_i at position $(l_i, \theta_1 + \frac{2\pi}{n_r} \times (i-1))$, distributes the tree components well and allow us to realize the desired edge lengths. When the input is a balanced tree this algorithm computes a symmetric layout; see Figure 5.

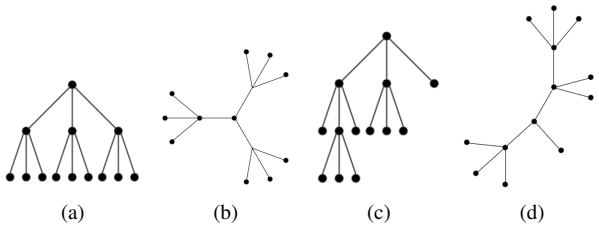


Figure 5: Illustrating the output layout of algorithm 6 w.r.t. different types of inputs. If the tree is balanced (see 5a) then the output is symmetric (see 5b). If the tree is not balanced (see 5c) then the output is not symmetric (see 5d).

Algorithm 6 divides the angular region of the root node and distributes the regions to child nodes. The size of the angular regions is proportional to the size of the child subtrees. Then the process is continued recursively.

3.1.2. Simultaneous DEL and Overlap-removal Improvement

This initial layout is crossing free and preserves desired edge lengths. However, it may contain label overlaps. For example, in Figure 6 we show a layout of a large real-world graph which contains many dense regions that will cause label overlaps. To remove these overlaps and obtain a more natural drawing we take the initial layout as an input to a force-directed algorithm. In this algorithm, we add two forces, a collision force to every node to remove label overlaps, and a force to every edge to maintain the desired edge lengths. In every iteration of the algorithm, we ensure that no

edge crossings are introduced by checking every move to determine whether that move will introduce a crossing. If it would, we do not update the coordinate of the node; see algorithm 2.

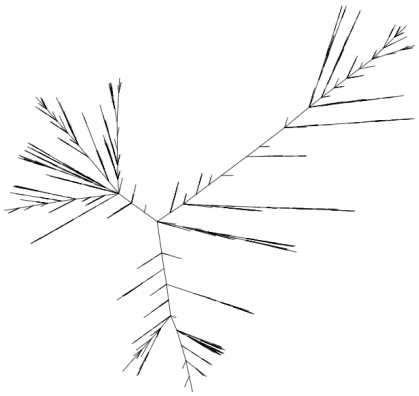


Figure 6: An initial layout using algorithm 6 that has good desired edge length preservation.

Algorithm 2: Simultaneous DEL and overlap-removal improvement algorithm($D, steps$)

```
Result: A layout with improved space utilization
Initialize the coordinates using D;
for i = 1, 2, ··· , steps do
    for each node v do
        Apply collision force;
    end
    for each edge e do
        Apply edge force;
    end
    for each node v do
        Update the coordinates of v if no crossing is
          introduced;
    end
end
```

We now discuss the two different types of forces used in this step in detail, specifically the collision force and the edge force. The goal of the collision force is to remove label overlaps, while the edge force works to maintain the desired edge lengths. To help define the collision force, each node is assigned a collision circle centered around the node. When a node u enters the collision circle of another node v then a repulsive force acts between u and v. We can present this repulsive force using the following equation:

$$f_r = \frac{K}{d} I(d),\ I(d) = \begin{cases} 1 & \text{if } d \leq r \\ 0 & \text{otherwise} \end{cases}$$

Here K is a constant, the default value is equal to the area of the initial drawing. The force f_r is activated when the distance d from u to v is less than or equal to the collision circle radius r. The value of f_r is reciprocal to d. We apply this collision force to every node.

In the initial embedding, the edge lengths are equal to the desired lengths. If we just apply collision force then the edge lengths will change arbitrarily. To keep the edge lengths close to the desired edge lengths, we will apply edge forces along with collision forces. We apply an edge force to each edge e and provide the equation of the edge force below:

$$f_r^e = \frac{K}{d} I_1(d),\ I_1(d) = \begin{cases} 1 & \text{if } d < l_e \\ 0 & \text{otherwise} \end{cases}$$

$$f_a^e = KdI_2(d),\ I_2(d) = \begin{cases} 1 & \text{if } d > l_e \\ 0 & \text{otherwise} \end{cases}$$

Here the force f_r^e is a repulsive force and f_a^e is an attractive force. Both forces are applied to each edge e. If the end vertices u and v of e have a distance smaller than the desired edge length l_e, then the repulsive force f_r^e gets activated, the function $I_1(d)$ is designed for this purpose. The force is reciprocal to the edge distance d. On the other hand, f_a^e gets activated when the edge distance d is greater than l_e, the function $I_2(d)$ is designed for this purpose. The force f_a^e is proportional to the distance d.

3.1.3. Final Overlap Removal

The previous step does not necessarily remove all label overlaps. In this step we go over all pairs of overlapping nodes and move them until the overlap is repaired. This process involves checking whether we can move one of the overlapping nodes so that the distance between two nodes increases without introducing any crossing and label overlap.

3.2. Compactness-guided (CG) Algorithm

In this section, we describe a second algorithm that achieves the desired properties but prioritizes a compact output.

This algorithm begins with either the same crossing-free layout given by algorithm 6 or an initial layout based on sfdp. After this initial layout, a modified force directed algorithm is used which will optimize the edge length and remove the label overlap while also ensuring that we do not introduce any edge crossings to the graph.

This algorithm contains three basic steps. These are laid out below and in 7.

- **Initialize** a crossing-free layout.
- **Fine Tuning:** iteratively optimize desired edge length, remove label overlap, and maintain other layout aesthetics.
- **Crossing check:** after every iteration, check whether the update of a node introduces new crossings or not. If so, reduce the step size to a certain fraction of the current one, until all crossings are eliminated.

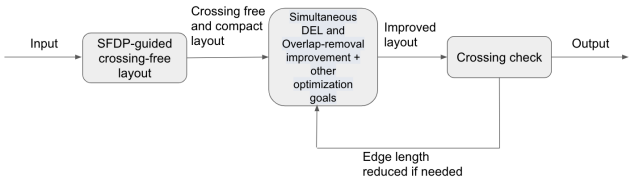


Figure 7: The pipeline of the CG algorithm.

3.2.1. Crossing-free Initialization

We start with a layout algorithm, SFDP [EGK*01], which preserves a good global structure of the graph but may have a few crossings that we need to remove to create a good initial layout. First, we pick a node as the root of the tree. In our experiment, this was chosen as the node with the most children. Then we find a breadth-first search ordering of the remaining nodes. Next, we start with the root and add one node at a time while maintaining the crossing-free: we initialize a new node around its parent node which has already been initialized, following their relative position in the SFDP layout. If the new node introduces any crossings, we shrink the edge length (e.g. by 20%) and randomly place it around its parent by adding a random perturbation. We repeat this process until it does not introduce any crossings (see algorithm 3).

Algorithm 3: Layout Initialization

Result: SFDP-guided crossing-free layout
layout = {root};
$k = 0.8$;
for *each node* $v \in V \setminus root$ **do**
  $dx = sfdp(v).x - sfdp(\pi(v)).x$;
  $dy = sfdp(v).y - sfdp(\pi(v)).y$;
  $d = \sqrt{dx^2 + dy^2}$;
  $v.x = \pi(v).x + dx$;
  $v.y = \pi(v).y + dy$;
  i = 1;
  while *hasCrossing(layout* $\cup$ *v)* **do**
    $r = random\_uniform(0, 1 - k^i)$;
    $\theta = random\_uniform(0, 2\pi)$;
    $v.x = \pi(v).x + k^i \cdot dx + r \cdot d \cdot cos\theta$;
    $v.y = \pi(v).y + k^i \cdot dy + r \cdot d \cdot sin\theta$;
    i++;
  end
  $layout = layout \cup \{v\}$;
end
return layout;

3.2.2. Iterative Improvement

Once we have a good initialization, we move to improving the layout. This step consists of a force directed algorithm with forces for edge length preservation, overlap removal, and some other aesthetics. These other aesthetics are not explicit in our seven desired properties, but do help the overall layout and global structure of the layout. We will now go over the forces for each of these three considerations.

Desired Edge Length To get the edge lengths closer to the desired length, we use a force set up in the same way as the one for edge lengths in Section 3.1.2.

Label Overlap Removal To remove the label overalp, We use a collision force that is similar to the method described in Section 3.1.2, but instead of using a circular region for collision detection, we use elliptic regions. Normally, we can only specify a circular area for collision detection, however, since labels are typically wider than they are tall, a circular collision force could give too much space on the top of our labels and result in poor space utilization. We can build an elliptical force out of a circular collision force by stretching the y-coordinate by a factor b (e.g. we set $b = 3$) before a circular collision force is applied, and restoring the coordinate back after applying the circular force. The velocity computed by the collision force should be processed in a similar manner, with the scaling factor set reciprocally. Formally, let each node $v \in V$ store the x, y coordinates of the layout and $v.x, v.y$, the x, y directions of the force to be updated at the end of the iteration. We specify a different collision radius, denoted by $v.r$, for every node v. In our experiments, we set it to be half of the label width. Note that the collision radius depends on both font size of a label and number of characters of the text. A circular collision force will calculate and update the movement speed $v.vx$ and $v.vy$ according to the bounding area of nodes defined by their respective radii, under the stretched layout. Algorithm 4 describes how we remove label overlaps through an elliptic force.

Algorithm 4: Overlap Removal(V)

Result: An elliptical force that removes label overlaps
b = 3;
for *each node* $v \in V$ **do**
  v.y *= b;
  v.vy *= 1/b;
end
for *each node* $v \in V$ **do**
  Apply circular collision force with radius v.r, which will update v.vx and v.vy if v collides with other nodes.
end
for *each node* $v \in V$ **do**
  v.y *= 1/b;
  v.vy *= b;
end

Other Aesthetics Note that both desired edge length preservation and label overlap removal only affect layout locally, neither encourages a good global structure of the layout. To preserve the global structure of a graph layout, we added forces to minimize stress and improve readability.

First, we add a force to minimize stress. We only add this force if two nodes are away by more than 1 hop, since we do not want to interfere with the forces already defined to maintain the desired edge lengths. For every pair of nodes $v_i, v_j \in V$, the total stress is

defined as

$$stress = \Sigma_{v_i,v_j \in V}\ w_{ij}(||v_i - v_j||_2 - l_{ij})^2$$

where $||v_i - v_j||_2$ denotes the Euclidean distance between nodes i and j, l_{ij} denotes the desired distance, and w_{ij} is a weighting coefficient. We set l_{ij} to be a power of the shortest-path distance $d_G(i,j)$ between the two nodes v_i and v_j in the graph. The exponent depends on the number of hops $h(i,j)$ between two nodes. We set

$$l_{ij} = d_G(i,j)^{L_1 + 1/h(i,j)}$$

Where L_1 is a positive constant that defines an asymptotic bound to the target distance. In our examples, we set $L_1 = 0.95$. Generally speaking, this design of desired distance between every pair of nodes encourages a straight path between nearby nodes and a curved path between far-reaching nodes. We set the weight to be

$$w_{ij} = 1/d_G(i,j)^{L_2}$$

In our experiments we set $L_2 = 1.3$, as this gives a good balance between the global structure and compactness of the graph layout. We minimize the stress by stochastic gradient descent - moving nodes towards the negative gradient of the stress defined above. Given that other forces go through all the n nodes once per iteration and we want to keep the number of updates in stress minimization comparable to other forces, we randomly choose n pairs of nodes to update in each iteration.

Next, we apply a repulsive force between every pair of nodes to further improve the readability of the drawing. We set the repulsive force between two nodes as inversely proportional to the squared distance in the current layout, this resembles an electrical charge between nodes.

$$|f_{charge}(v_i,v_j)| = s(i,j)/||X_i - X_j||^2$$

where X_i and X_j are the locations of nodes i and j and $s(i,j)$ denotes the strength of the charge between the nodes and depends on the longest desired edge length that are in the set of the children of i and j. We set

$$s(i,j) = max_{k \in V, e_{ik} \in E}\ \{l_{ik}\} * max_{k \in V, e_{jk} \in E}\ \{l_{jk}\}$$

Finally, we add a force between nodes and edges. This force helps the readability as well, to help make sure there are less labels directly over edges. This force is inversely proportional to the distance between the node and edge, pointing orthogonally from the edge, and is capped to zero if the node does not project onto the edge segment or is too far from the edge.

$$|f_{node-edge}(v_i,e_j)| = c/d(v_i,e_j)$$

Where c is a constant across all pairs and $d(v_i,e_j)$ denotes the distance between v_i and e_j.

3.2.3. Crossing Check

As mentioned before, at every iteration we check whether the layout update will introduce any new crossings. If so, for each node which introduce new crossings, we try reducing theg step size to a fraction k (e.g. $k = 0.8$) of the original. If the update at this new step size does not introduce a crossing, we make the update, but if it would still introduce a crossing, we repeat and reduce the step size further. This continues until all crossings are eliminated or a maximum number of iterations (e.g. $niter = 12$) is reached. To keep a relatively balanced amount of updates for each node, we shuffle the ordering of nodes in every iteration. See algorithm 5 for more detail.

Algorithm 5: Layout Crossing Check

```
Result: A Crossing-free layout
shuffle(V);
niter = 12;
k = 0.8;
for each node v ∈ V do
    v.x0 = v.x;
    v.y0 = v.y;
    v.x = v.x0 + v.vx;
    v.y = v.y0 + v.vy;
    i = 0;
    while hasCrossing(V,E) and i < niter do
        v.x = v.x0 + k^i · v.vx;
        v.y = v.y0 + k^i · v.vy;
        i++;
    end
    if i == niter then
        v.x = v.x0;
        v.y = v.y0;
    end
end
```

We have used the implementation in the visualization library d3 [BOH11] for overlap removal by collision force, desired edge length improvement forces and charges between nodes. We have used NetworkX [HSSC08] for computing the center of a tree and breadth first search.

4. Interactive Interface and Prototype

In this section we describe a functional browser interface and a prototype deployment of ZMLT with two real-world datasets.

4.1. Layer Definition

We use a multi-level node-weighted Steiner tree (MVST) [DEK*18], defined and discussed in the Introduction to determine which level a node will appear. For our purposes we are interested in the maximum weight Steiner tree, as nodes and edges with high weights are important and should be present on the higher levels.

The number of levels in the heirarchy is a user defined parameter. We use 8 levels, with the following percentage of nodes appearing at each level: $(5, 15, 30, 40, 60, 70, 85, 100)$.

While the Steiner tree problem usually asks for the minimum cost tree, this is easy to fix by simply choosing the reciprocal of the weights (assuming all weights are positive and greater than 1). This approach can also be extended to unweighted graphs, where node weights can be assigned based on structural graph importance, e.g. node degrees or betweenness centrality.

4.2. Web Browser Interface

The output of the algorithms (CG and DELG) contain the positions of all the nodes in all the levels of the graph. We modify the node-link representation into a map-like one using *GMap* [GHK10]. Finally, we extract GeoJSON files and use OpenLayers [Tea20] to display the data in the browser. We briefly describe the process.

Node Clustering: By default, we use MapSets [KPS14] to cluster the graph and show the clusters as contiguous regions on the map, although other GMap [GHK10] options are also available.

Generating Map Layers: GeoJSON [BDD*08] is a standard format for representing simple geographical features, along with their non-spatial attributes. We split all geometric elements of the map into *nodelayer*, *edgelayer*, and *clusterlayer*. Each of the map layers is composed of many attributes. For example, nodelayer contains node-ID, coordinates, font-name, font-size, height, width, label and weights for all nodes.

JavaScript Application: For rendering and visualizing the map layers we use OpenLayers [Tea20], a JavaScript library for displaying map data in the browser. This visualization requires additional work to show the map elements in a multi-layer fashion. Node attributes, edge attributes and zoom level are used to determine the appropriate levels in the multi-level visualization.

Interactions: As an interactive visualization tool, we enable panning and zooming, as well as clicking on nodes and edges to see details such as level and weight.

4.3. Prototype ZMLT Deployment

In this section we evaluate the DELG and CG algorithms on two real-world graphs: (i) the Google Topics Graph [BEH*18] capturing relations between research topics extracted from Google Scholar with 5947 nodes and 26695 edges; and (ii) the Last.fm Graph [GHK10] showing relations between musical artists with 2588 nodes and 28221 edges.

The Google Topics graph is obtained from Google Scholar academic research profiles. The nodes of the graph are research topics, with weights corresponding to the number of people reporting to work on them. Edges are placed between pairs of topics that co-occur in the profiles.

In the Google Topics graph, there are many high degree nodes, for example, the maximum degree is 153. It is hard to remove label overlaps when one vertex has a large number of neighbors. Hence, we removed a few edges and made the maximum degree equal to 30. In the original graph, the number of nodes was 5943, after reducing the degree the number of nodes becomes 5483. In the degree reduction method, we compare the edge weight with the weight of the end vertices, and we compute a score. If that score is smaller than a cut-off then we delete the edge. Specifically, we use the following formula:

$$\frac{weight(e) - \frac{weight(u)\times weight(v)}{W}}{\sqrt{\frac{weight(u)\times weight(v)}{W}}} < cutoff$$

Here, $e = (u,v)$ is an edge and $W = \sum_{v\in V} weight(v)$. We keep the value of cut-off equal to 50 for our dataset.

The Last.fm graph, is based on data from the last.fm website, an Internet radio and music community website with a recommender system based on user listening habits. The nodes are popular musical artists with weights corresponding to the number of listeners. Edges are placed between pairs of similar artists, based on listening habits.

Both of these graphs can be interactively explored in our prototype ZMLT implementation: http://uamap-dev.arl.arizona.edu:8086/.

4.4. Evaluation

We compare our two ZMLT algorithms to an implementation relying on an off-the-shelf tool using both the Google Topics graph and the Last.fm graphs as defined above.

Direct Approach: The goals of ensuring a crossing-free output and labeling all nodes without overlaps can be achieved using the off-the-shelf *Circular Layout* offered by *yED*. This algorithm produces layouts that emphasize group and tree structures within a network by drawing trees in a radial tree layout fashion [WEK04]. We produced this layout by drawing the last layers of the hierarchies, namely, T_8 using Circular Layout with BCC Compact and the "consider node labels" feature. Then we extracted the subtrees of the hierarchy. In the following we call this procedure the *Direct Approach*.

Quality Metrics: We use the following two metrics to compare our outputs and those from the direct approach:

- *Desired edge length (DEL)* evaluates the normalized desired edge lengths in each layer. Since higher levels include the most important nodes and edges they correspond to larger nodes and longer edges. Given the desired edge lengths $\{l_{ij} : (i,j) \in E\}$, defined in section 3, and coordinates of the nodes X in the computed layout, we evaluate DEL with the following formula:

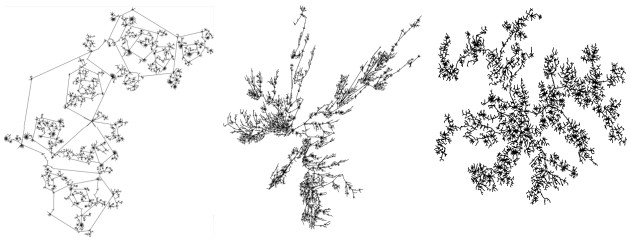
$$\texttt{DL} = \sqrt{\frac{1}{|E|}\sum_{(i,j)\in E}\left(\frac{||X_i - X_j|| - l_{ij}}{l_{ij}}\right)^2} \quad (1)$$

  This measures the root mean square of the relative error as in [ALD*20] and produces a positive number, with zero corresponding to perfect realization.
- *Compactness* measures the ratio between the sum of the 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).

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

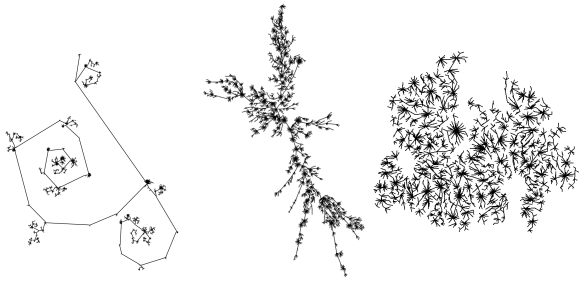
  CM scores are in the range $[0,1]$, where 1 corresponds to perfect area utilization.

Results: We show the layouts created by the three algorithms for the Google Topics graph in Figure 8 and for the Last.fm graph in Figure 9. These figures highlight some significant differences



(a) Direct Approach (b) DELG (c) CG

Figure 8: Comparison of the tree layout structure of the Google Topics graph drawn with the Direct Approach, our Desirable-Edge-Length-guided (DELG) Algorithm and our Compactness-guided (CG) Algorithm.



(a) T_8 (b) T_8 (c) T_8

Figure 9: Comparison of the tree layout structure of the Last.fm graph drawn with the Direct Approach, our Desirable-Edge-Length-guided (DELG) Algorithm and our Compactness-guided (CG) Algorithm.

which stand out visually. For example, in order to achieve compact layouts, the direct approach places nodes in a spiral fashion, which is not a good representation of the underlying graphs.

The metric analysis in Table 1 and Table 2 shows that the two ZMLT algorithms better utilize the drawing area and better realize the desired edge lengths. The tables report the values for the two measures computed on each of the 8 trees in the graph hierarchy (representing the graphs at different levels of detail). Low values in desired edge length preservation and high values in compactness correspond to better results. According these measurements, the ZMLT algorithms are consistently better than the direct approach.

Error Analysis: From the measures in Table 1 and Table 2 as well as the overview of tree layouts, we can see that the two ZMLT algorithms favor different objectives: the DELG algorithm performs better in desired edge length preservation while the CG algorithm performs better in compactness. Here we take a closer look at the results returned by these two algorithms and try to analyze their respective failure modes.

First, we focus on desired edge length preservation. Figure 10 colors individual edges in the layout by their relative error. Recall that we use the relative error to measure the desired length preservation in Equation 1. For each edge $e_{ij} \in E$, it measures the discrepancy between the actual edge length $||X_i - X_j||$ in the drawing and the given desired edge length l_{ij}:

$$\text{relative\_error} = \frac{||X_i - X_j|| - l_{ij}}{l_{ij}}$$

With the DELG algorithm, we observed from the left layout and histogram in Figure 10 that most edges are drawn with their desired edge lengths, thanks to its edge-length guided initialization. We can see several extensively stretched edges (colored in blue in Figure 10), 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. On the other hand, with the CG algorithm the error is well distributed, but when comparing to DELG the edge lengths tend to deviate more from the desired lengths. We suspect this is due to how the CG algorithm attempts to remove label overlaps: we can see that in dense regions the edges are more likely to be stretched, whereas on the periphery the edges tend to be compressed.

Next we look at the compactness of the two ZMLT layouts. Consider the difference between the two algorithms in their rendering

	Desired Edge Length		
	DIR	DELG	CG
T_1	0.56	**0.03**	0.38
T_2	1.26	**0.12**	0.40
T_3	1.64	**0.13**	0.39
T_4	1.84	**0.21**	0.41
T_5	2.55	**0.28**	0.42
T_6	1.98	**0.41**	0.43
T_7	2.36	**0.44**	0.45
T_8	1.93	**0.40**	0.51

	Compactness		
	DIR	DELG	CG
T_1	1e-5	**0.019**	0.010
T_2	1e-5	0.008	**0.009**
T_3	1e-5	0.011	**0.012**
T_4	9e-6	0.012	**0.021**
T_5	7e-6	**0.014**	0.011
T_6	1e-5	0.014	**0.015**
T_7	6e-6	0.015	**0.020**
T_8	7e-6	0.011	**0.026**

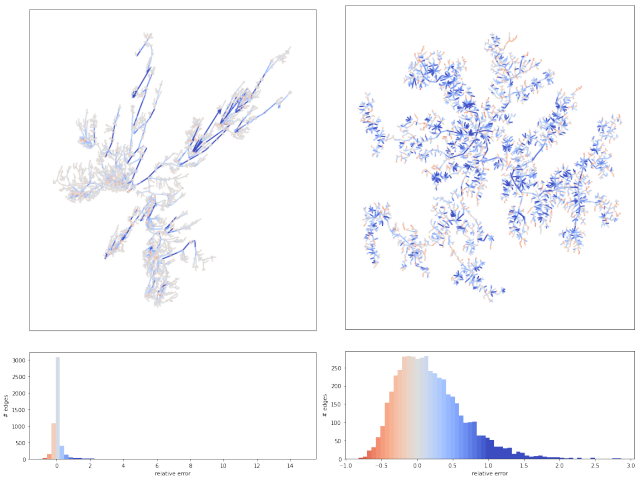
Table 1: Metric-based Topics graph layout evaluation of the Direct Approach (DIR), Desirable-Edge-Length-guided (DELG) Algorithm and Compactness-guided (CG) Algorithm. **Left:** Desired Edge Length, evaluated through Equation 1. Values ranges from 0 (best) to positive infinity (worst). **Right:** Compactness evaluated through Equation 2 values range from 0 (worst) to 1 (best).

	Desired Edge Length		
	DIR	DELG	CG
T_1	1.262	**0.04**	0.43
T_2	1.484	**0.05**	0.50
T_3	1.603	**0.06**	0.51
T_4	1.727	**0.09**	0.51
T_5	1.847	**0.10**	0.53
T_6	1.907	**0.12**	0.54
T_7	1.976	**0.12**	0.56
T_8	2.061	**0.13**	0.60

	Compactness		
	DIR	DELG	CG
T_1	1.5e-6	0.017	**0.018**
T_2	1e-6	0.015	**0.025**
T_3	1e-6	0.015	**0.022**
T_4	6e-7	0.014	**0.026**
T_5	5e-7	0.012	**0.032**
T_6	4e-7	0.011	**0.034**
T_7	4e-7	0.010	**0.038**
T_8	3e-7	0.010	**0.041**

Table 2: Last.fm graph layout evaluation of the Direct Approach (DIR), Desirable-Edge-Length-guided (DELG) Algorithm and Compactness-guided (CG) Algorithm. **Left:** Desired Edge Length, evaluated through Equation 1 where the value ranges from 0 (best) to positive infinity (worst). **Right:** Compactness evaluated through Equation 2 where the value ranges from 0 (worst) to 1 (best).

of the region around the 'Artificial Intelligence' node in the maps, as shown in Figure 11. From the layout overview in Figure 11, we can already see that the DELG algorithm uses space less efficiently, as it leaves more empty space, for example on the lower right of the whole layout. The 'Artificial Intelligence' node, circled in blue in Figure 11, is close to multiple heavy subtrees such as those from the nodes 'natural language processing', 'machine learning' and 'computer vision'. On the CG algorithm the heavy trees are distributed evenly, due to its good initial layout. The DELG algorithm, on the other hand, placed all heavy branches in the top left quadrant, resulting in a less efficient use of drawing area, especially in the top right region or bottom left region around the 'Artificial Intelligence' node.



(a) DELG (b) CG

Figure 10: Analysis of failure modes in desired edge length preservation. **Top left:** In the layout of Topic graph computed by the DELG algorithm, most edges are drawn with their desired edge lengths, while some of the edges are drawn too long (colored in blue) and almost no edges are drawn too short (in red). **Bottom left:** A histogram of relative errors in desired edge length preservation, colored in the same way as the layout. **Top right and Bottom right**: The same visual analysis on the Topic Graph layout returned by the CG algorithm, where highway edges often stretched (blue) and edges closer to leaf nodes are often compressed (red).

5. Discussion and Limitations

There are many limitations to the proposed approach for multi-level representation of large graphs and the evaluation we provide in this paper. We focus on a small set of limitations and briefly discuss possible directions for improvement.

From trees to general graphs: We use trees to obtain connected graphs for each level that represent the underlying graph well. To leverage the map-representation metaphor better, we need subgraphs that are closer to real-world road networks. Generalizing the approach to multi-level planar subgraphs or multi-level graph spanners [ABS*20] (sparse subgraphs that approximately preserve shortest path distances) will likely lead to better results.

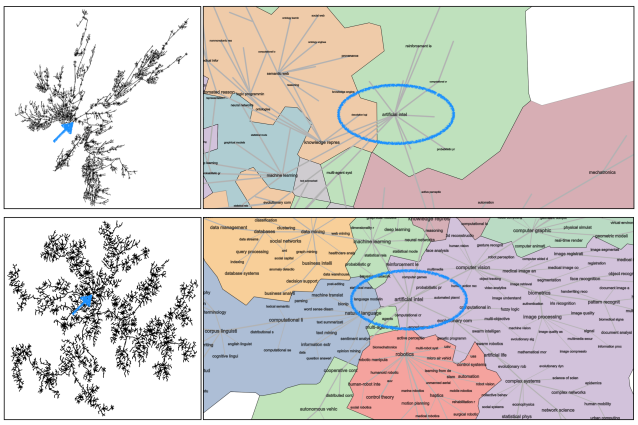


Figure 11: Analysis of failure modes in layout compactness. **Top:** The topic graph layout returned by the DELG algorithm has more unused space and fails to distribute heavy subtrees around the 'Artificial Intelligence' node (pointed and circled in blue) evenly. **Bottom:** The CG algorithm generates a more balanced layout and is able to distribute subtrees more evenly.

UI-centric graph drawing: The proposed approaches rely on several user-defined specifications, such as the number of levels and size of each level. Automatically determining the appropriate number of levels for a given graph is non-trivial and UI-centric: the number of non-overlapping labels one can show on a given zoom scale are layout-dependent. For now, the set of labeled nodes on each level in the graph hierarchy is predetermined by the Steiner tree algorithm without consideration of the labels. In other words, we are optimizing labels in a pre-designed graph hierarchy, while the label overlaps may also be resolvable by *redesigning* the graph hierarchy.

Interactive Labels A group of lengthy labels in the graph may sometimes be hard to deal with non-overlapping labels is a hard constraint. To simplify the problem, we make two simplifying decisions: (1) we use the same font size for all labels, regardless of what level they are on; and (2) we truncate lengthy labels in the graphs, such as "multidisciplinary design optimization", to no more than 16 characters. Using different font sizes and interactively adjusting them according to the zoom level is natural direction for future work.

Scaling to larger graphs: Our initial implementations of the two ZMLT algorithms are not efficient. While these algorithms only need to be run once to precompute the levels of details and set up the interactive map visualization, we have only tested them on graphs with about 5000 nodes.

Leveraging the map metaphor: In the visualization we do not leverage all the possibilities of a map-based approach. For example, highways and roads still look like collections of graph edges rather than smooth curves. To leverage the map metaphor better, one could represent edges and paths in a way that more closely resembles a road network or a railway network.

Evaluation: We only used two real-world graphs, two metrics, and

one alternative approach to evaluate our ZMLT algorithms. Better quantitative analysis and some qualitative analysis is needed.

6. Conclusions

Taking inspiration from maps, our approach applies people's natural understanding of maps to a graph layout. Specifically, we presented multi-level graph visualization algorithms that use semantic multi-level trees of a given graph, together with the map metaphor that guarantee several desirable properties (real, persistent, overlap-free, crossing-free, and appropriate representation) and optimize several others (compact and appropriate layout). The seven properties provide an initial set of guidelines for such representations, we demonstrate a prototype implementation of the two ZMLT methods and provide source code for the entire system.

Acknowledgments

We thank Felice de Luca and Iqbal Hossain for help with earlier versions of ZMLT [DLHG*19]. We also thank Faryad Darabi Sahneh for his contribution to refining the topics graph by reducing the maximum degree.

References

[AAB*17] AUBER D., ARCHAMBAULT D., BOURQUI R., DELEST M., DUBOIS J., LAMBERT A., MARY P., MATHIAUT M., MELANÇON G., PINAUD B., RENOUST B., VALLET J.: TULIP 5. In *Encyclopedia of Social Network Analysis and Mining*, Alhajj R., Rokne J., (Eds.). Springer, Aug. 2017, pp. 1–28. 3

[AAS*18] AHMED R., ANGELINI P., SAHNEH F. D., EFRAT A., GLICKENSTEIN D., GRONEMANN M., HEINSOHN N., KOBOUROV S. G., SPENCE R., WATKINS J., WOLFF A.: Multi-Level Steiner Trees. In *17th International Symposium on Experimental Algorithms (SEA 2018)* (Dagstuhl, Germany, 2018), D'Angelo G., (Ed.), vol. 103 of *Leibniz International Proceedings in Informatics (LIPIcs)*, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp. 15:1–15:14. URL: http://drops.dagstuhl.de/opus/volltexte/2018/8950, doi:10.4230/LIPIcs.SEA.2018.15. 2, 3

[ABS*20] AHMED R., BODWIN G., SAHNEH F. D., HAMM K., JEBELLI M. J. L., KOBOUROV S., SPENCE R.: Graph spanners: A tutorial review. *Computer Science Review 37* (2020), 100253. 10

[AKY05] ABELLO J., KOBOUROV S., YUSUFOV R.: Visualizing large graphs with compound-fisheye views and treemaps. In *GD* (2005), pp. 431–441. 3

[ALD*20] AHMED R., LUCA F. D., DEVKOTA S., KOBOUROV S., LI M.: Graph drawing via gradient descent, $(gd)^2$. *arXiv preprint arXiv:2008.05584* (2020). 8

[ASH*20] AHMED R., SAHNEH F. D., HAMM K., KOBOUROV S., SPENCE R.: Kruskal-Based Approximation Algorithm for the Multi-Level Steiner Tree Problem. In *28th Annual European Symposium on Algorithms (ESA 2020)* (Dagstuhl, Germany, 2020), Grandoni F., Herman G., Sanders P., (Eds.), vol. 173 of *Leibniz International Proceedings in Informatics (LIPIcs)*, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, pp. 4:1–4:21. URL: https://drops.dagstuhl.de/opus/volltexte/2020/12870, doi:10.4230/LIPIcs.ESA.2020.4. 2

[AVHK06] ABELLO J., VAN HAM F., KRISHNAN N.: Ask-GraphView: A large scale graph visualization system. *Visualization and Computer Graphics, IEEE Transactions on 12*, 5 (2006), 669–676. 3

[BDD*08] BUTLER H., DALY M., DOYLE A., GILLIES S., SCHAUB T., SCHMIDT C.: The geojson format specification. *Rapport technique 67* (2008). 8

[BEH*18] BURD R., ESPY K. A., HOSSAIN M. I., KOBOUROV S., MERCHANT N., PURCHASE H.: Gram: Global research activity map. In *Proceedings of the 2018 International Conference on Advanced Visual Interfaces* (New York, NY, USA, 2018), AVI '18, ACM, pp. 31:1–31:9. doi:10.1145/3206505.3206531. 3, 8

[BGRS10] BYRKA J., GRANDONI F., ROTHVOSS T., SANITÀ L.: An improved lp-based approximation for steiner tree. In *Proceedings of the forty-second ACM symposium on Theory of computing* (2010), ACM, pp. 583–592. 3

[BH86] BARNES J., HUT P.: A hierarchical O(N log N) force calculation algorithm. *Nature 324* (Dec. 1986), 446–449. 2

[BHJ09] BASTIAN M., HEYMANN S., JACOMY M.: Gephi: An open source software for exploring and manipulating networks, 2009. 3

[BKB05] BOYACK K. W., KLAVANS R., BÖRNER K.: Mapping the backbone of science. *Scientometrics 64*, 3 (2005), 351–374. 2

[BKMK07] BORRADAILE G., KENYON-MATHIEU C., KLEIN P.: A polynomial-time approximation scheme for steiner tree in planar graphs. In *SODA* (2007), vol. 7, pp. 1285–1294. 3

[BMBH16] BÖRNER K., MALTESE A., BALLIET R. N., HEIMLICH J.: Investigating aspects of data visualization literacy using 20 information visualizations and 273 science museum visitors. *Information Visualization 15*, 3 (2016), 198–213. doi:10.1177/1473871615594652. 1

[BMM94] BALAKRISHNAN A., MAGNANTI T. L., MIRCHANDANI P.: Modeling and Heuristic Worst-Case Performance Analysis of the Two-Level Network Design Problem. *Management Science 40*, 7 (July 1994), 846–867. URL: https://ideas.repec.org/a/inm/ormnsc/v40y1994i7p846-867.html, doi:10.1287/mnsc.40.7.846. 3

[BOH11] BOSTOCK M., OGIEVETSKY V., HEER J.: D^3 data-driven documents. *IEEE transactions on visualization and computer graphics 17*, 12 (2011), 2301–2309. 7

[BP07] BRANDES U., PICH C.: Eigensolver methods for progressive multidimensional scaling of large data. In *Graph Drawing* (2007), Springer, pp. 42–53. 2, 3

[BP09] BRANDES U., PICH C.: An experimental study on distance-based graph drawing. In *Graph Drawing* (2009), Springer, pp. 218–229. 3

[CC08] CHLEBÍK M., CHLEBÍKOVÁ J.: The steiner tree problem on graphs: Inapproximability results. *Theoretical Computer Science 406*, 3 (2008), 207–214. 3

[CGJ*11] CHIMANI M., GUTWENGER C., JÜNGER M., KLAU G. W., KLEIN K., MUTZEL P.: The open graph drawing framework (OGDF). *Handbook of Graph Drawing and Visualization* (2011), 543–569. 3

[CGNS08] CHUZHOY J., GUPTA A., NAOR J. S., SINHA A.: On the approximability of some network design problems. *ACM Trans. Algorithms 4*, 2 (May 2008), 23:1–23:17. URL: http://doi.acm.org/10.1145/1361192.1361200, doi:10.1145/1361192.1361200. 3

[CLRS01] CORMEN T. H., LEISERSON C. E., RIVEST R. L., STEIN C.: *Introduction to algorithms*, vol. 5. MIT press Cambridge, 2001. 4

[CNS04] CHARIKAR M., NAOR J., SCHIEBER B.: Resource optimization in qos multicast routing of real-time multimedia. *IEEE/ACM Transactions on Networking 12*, 2 (April 2004), 340–348. doi:10.1109/TNET.2004.826288. 3

[DEK*18] DARABI SAHNEH F., EFRAT A., KOBOUROV S., KRIEGER S., SPENCE R.: Approximation algorithms for the vertex-weighted grade-of-service Steiner tree problem. *arXiv e-prints* (Nov 2018), arXiv:1811.11700. arXiv:1811.11700. 3, 7

[DLHG*19] DE LUCA F., HOSSAIN I., GRAY K., KOBOUROV S.,

BÖRNER K.: Multi-level tree based approach for interactive graph visualization with semantic zoom. *arXiv preprint arXiv:1906.05996* (2019). 11

[DNE08] DO NASCIMENTO H. A., EADES P.: User hints for map labeling. *Journal of Visual Languages & Computing 19*, 1 (2008), 39–74. 2

[DNMB11] DE NOOY W., MRVAR A., BATAGELJ V.: *Exploratory social network analysis with Pajek*, vol. 27. Cambridge University Press, 2011. 3

[DWB01] DAVIDSON G. S., WYLIE B. N., BOYACK K. W.: Cluster stability and the use of noise in interpretation of clustering. In *IEEE Symposium on Information Visualization* (2001), pp. 23–30. 2

[Ead84] EADES P.: A heuristic for graph drawing. *Congressus Numerantium 42* (1984), 149–160. 2

[EGK*01] ELLSON J., GANSNER E., KOUTSOFIOS L., NORTH S., WOODHULL G., DESCRIPTION S., TECHNOLOGIES L.: Graphviz — open source graph drawing tools. In *Lecture Notes in Computer Science* (2001), Springer-Verlag, pp. 483–484. 2, 3, 6

[FR91] FRUCHTERMAN T. M. J., REINGOLD E. M.: Graph drawing by force-directed placement. *Software: Practice and Experience 21*, 11 (1991), 1129–1164. doi:10.1002/spe.4380211102. 2

[GGK04] GAJER P., GOODRICH M., KOBOUROV S.: A fast multi-dimensional algorithm for drawing large graphs. *Computational Geometry: Theory and Applications 29*, 1 (2004), 3–18. 2

[GH09] GANSNER E. R., HU Y.: Efficient node overlap removal using a proximity stress model. In *Graph Drawing* (Berlin, Heidelberg, 2009), Tollis I. G., Patrignani M., (Eds.), Springer Berlin Heidelberg, pp. 206–217. 3

[GHK10] GANSNER E. R., HU Y., KOBOUROV S.: Gmap: Visualizing graphs and clusters as maps. In *2010 IEEE Pacific Visualization Symposium (PacificVis)* (March 2010), pp. 201–208. doi:10.1109/PACIFICVIS.2010.5429590. 8

[GHK13] GANSNER E. R., HU Y., KRISHNAN S.: Coast: A convex optimization approach to stress-based embedding. In *Graph Drawing* (2013), Springer, pp. 268–279. 3

[GHN13] GANSNER E. R., HU Y., NORTH S.: A maxent-stress model for graph layout. *Visualization and Computer Graphics, IEEE Transactions on 19*, 6 (2013), 927–940. 3

[GK99] GUHA S., KHULLER S.: Improved methods for approximating node weighted steiner trees and connected dominating sets. *Information and computation 150*, 1 (1999), 57–74. 3

[GKN05] GANSNER E., KOREN Y., NORTH S.: Topological fisheye views for visualizing large graphs. *TVCG 11*, 4 (July 2005), 457–468. 3

[GN98] GANSNER E. R., NORTH S. C.: Improved force-directed layouts. In *Proceedings of the 6th International Symposium on Graph Drawing* (Berlin, Heidelberg, 1998), GD '98, Springer-Verlag, pp. 364–373. URL: http://dl.acm.org/citation.cfm?id=647550.729069. 3

[HCL05] HEER J., CARD S. K., LANDAY J. A.: Prefuse: a toolkit for interactive information visualization. In *Proc. SIGCHI conference on Human factors in computing systems* (2005), ACM, pp. 421–430. 3

[HH01] HADANY R., HAREL D.: A multi-scale algorithm for drawing graphs nicely. *Discrete Applied Mathematics 113*, 1 (2001), 3–21. 2

[HSSC08] HAGBERG A., SWART P., S CHULT D.: *Exploring network structure, dynamics, and function using NetworkX*. Tech. rep., Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008. 7

[Hu05] HU Y.: Efficient, high-quality force-directed graph drawing. *Mathematica Journal 10*, 1 (2005), 37–71. 2

[Kar72] KARP R. M.: Reducibility among combinatorial problems. In *Complexity of computer computations*. Springer, 1972, pp. 85–103. 3

[KCH02] KOREN Y., CARMEL L., HAREL D.: Ace: A fast multiscale eigenvectors computation for drawing huge graphs. In *Information Visualization, 2002. INFOVIS 2002. IEEE Symposium on* (2002), IEEE, pp. 137–144. 2

[KMWH20] KITTIVORAWANG C., MORITZ D., WONGSUPHASAWAT K., HEER J.: Fast and flexible overlap detection for chart labeling with occupancy bitmap. In *IEEE VIS Short Papers* (2020). URL: http://idl.cs.washington.edu/papers/fast-labels. 2, 3

[KPS14] KOBOUROV S., PUPYREV S., SIMONETTO P.: Visualizing Graphs as Maps with Contiguous Regions. In *EuroVis - Short Papers* (2014), Elmqvist N., Hlawitschka M., Kennedy J., (Eds.), The Eurographics Association. doi:10.2312/eurovisshort.20141153. 2, 8

[KR95] KLEIN P., RAVI R.: A nearly best-possible approximation algorithm for node-weighted steiner trees. *Journal of Algorithms 19*, 1 (1995), 104–115. 3

[KW78] KRUSKAL J. B., WISH M.: *Multidimensional Scaling*. Sage Press, 1978. 3

[LSC08] LUBOSCHIK M., SCHUMANN H., CORDS H.: Particle-based labeling: Fast point-feature labeling without obscuring other visual features. *IEEE transactions on visualization and computer graphics 14*, 6 (2008), 1237–1244. 3

[MN18] MONDAL. D., NACHMANSON. L.: A new approach to graphmaps, a system browsing large graphs as interactive maps. In *Proceedings of the 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 3: IVAPP,* (2018), INSTICC, SciTePress, pp. 108–119. doi:10.5220/0006618101080119. 3

[Mot07] MOTE K.: Fast point-feature label placement for dynamic visualizations. *Information Visualization 6*, 4 (2007), 249–260. 3

[MR07] MOSS A., RABANI Y.: Approximation algorithms for constrained node weighted steiner tree problems. *SIAM Journal on Computing 37*, 2 (2007), 460–481. 3

[MSTH03] MARRIOTT K., STUCKEY P., TAM V., HE W.: Removing node overlapping in graph layout using constrained optimization. *Constraints 8*, 2 (Apr 2003), 143–171. doi:10.1023/A:1022371615202. 3

[Ngu18] NGUYEN Q. H.: INKA: an ink-based model of graph visualization. *CoRR abs/1801.07008* (2018). URL: http://arxiv.org/abs/1801.07008, arXiv:1801.07008. 2

[NNB*17] NACHMANSON L., NOCAJ A., BEREG S., ZHANG L., HOLROYD A. E.: Node overlap removal by growing a tree. *J. Graph Algorithms Appl. 21*, 5 (2017), 857–872. doi:10.7155/jgaa.00442. 3

[NRL08] NACHMANSON L., ROBERTSON G., LEE B.: Drawing graphs with GLEE. In *Graph Drawing* (2008), Springer, pp. 389–394. 3

[Pur97] PURCHASE H.: Which aesthetic has the greatest effect on human understanding? In *Graph Drawing* (Berlin, Heidelberg, 1997), DiBattista G., (Ed.), Springer Berlin Heidelberg, pp. 248–261. 2

[RB03] RIVADENEIRA W., BEDERSON B. B.: A study of search result clustering interfaces: Comparing textual and zoomable user interfaces. *Studies 21* (2003), 5. 3

[SAH00] SCHROEDER W. J., AVILA L. S., HOFFMAN W.: Visualizing with VTK: a tutorial. *Computer Graphics and Applications 20*, 5 (2000), 20–27. 3

[Sch11] SCHULZ H.: Treevis.net: A tree visualization reference. *IEEE Computer Graphics and Applications 31*, 6 (Nov 2011), 11–15. doi:10.1109/MCG.2011.103. 3

[See80] SEERY J. B.: Designing network diagrams. In *General Conf. Social Graphics* (1980), vol. 49, p. 22. 3

[SMO*03] SHANNON P., MARKIEL A., OZIER O., BALIGA N. S., WANG J. T., RAMAGE D., AMIN N., SCHWIKOWSKI B., IDEKER T.:

Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome research 13*, 11 (2003), 2498–2504. 3

[SSKB14] SAKET B., SIMONETTO P., KOBOUROV S., BÖRNER K.: Node, node-link, and node-link-group diagrams: An evaluation. *IEEE Transactions on Visualization & Computer Graphics* (2014), 2231–2240. 3

[SSKB15] SAKET B., SCHEIDEGGER C., KOBOUROV S. G., BORNER K.: Map-based visualizations increase recall accuracy of data. *COMPUTER GRAPHICS FORUM 34*, 3 (2015), 441–450. 3

[Tea20] TEAM T. O. D.: *OpenLayers*, 2020 (accessed December 22, 2020). URL: https://openlayers.org/. 8

[Tor52] TORGERSON W. S.: Multidimensional scaling: I. theory and method. *Psychometrika 17*, 4 (1952), 401–419. 3

[WEK04] WIESE R., EIGLSPERGER M., KAUFMANN M.: yfiles visualization and automatic layout of graphs. In *Graph Drawing Software*. Springer, 2004, pp. 173–191. 3, 8

[WHBT14] WOJTON M. A., HEIMLICH J. E., BURRIS A., TRAMBY Z.: *Sense-making of Big Data Spring Break 2013 - Visualization recognition and meaning making*. Report, Indiana University, Lifelong Learning Group, 2014. 2, 3

[ZBDS12] ZINSMAIER M., BRANDES U., DEUSSEN O., STROBELT H.: Interactive level-of-detail rendering of large graphs. *IEEE Transactions on Visualization and Computer Graphics 18*, 12 (Dec 2012), 2486–2495. doi:10.1109/TVCG.2012.238. 3

Appendix A: DELG crossings-free initialization details

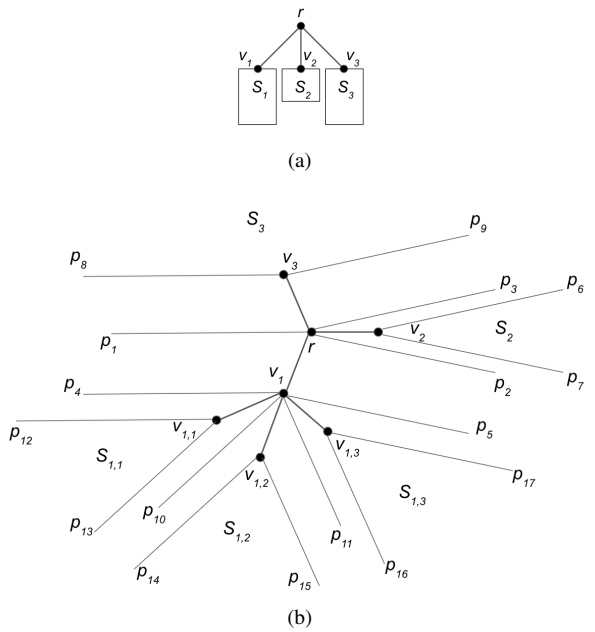


Figure 12: Illustrating different steps of Algorithm 6. The input graph is shown in Figure 12a. The division of angular regions based on the size of the subtrees is shown in Figure 12b.

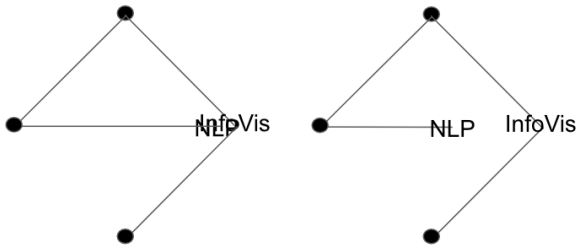
We illustrate the process by an example, see Figure 12. A symbolic representation of the input tree is shown in Figure 12a. Here the root node r has three children v_1, v_2 and v_3. The subtree rooted at v_i is denoted by S_i. The number of nodes in the subtrees S_1 and S_3 are equal and larger than the number of nodes in S_2. Algorithm 6 places the root vertex r and divides the space into three regions p_1rp_2, p_2rp_3 and p_1rp_3, see Figure 12b. Here we denote the angular region created by two segments p_1p_2 and p_1p_3 by $p_2p_1p_3$. The regions p_1rp_2, p_2rp_3 and p_1rp_3 are assigned to the nodes v_1, v_2 and v_3 respectively. The region p_2rp_3 is smaller than the other two regions because Algorithm 6 divides the regions proportionally to the size of the subtrees, and S_2 is smaller in size compared to the other two subtrees. Algorithm 6 now places the vertices in their regions. Consider the vertex v_1, the region assigned to v_1 is p_1rp_2. The vertex v_1 is placed on the bisector of rp_1 and rp_2 in such a way that the distance of rv_1 is equal to the desired edge length of the edge (r, v_1). Similarly, the algorithm places v_2 and v_3. The algorithm now translates the angular regions to the corresponding child vertices. For example, consider the child vertex v_2. We translate the line rp_3 to v_2p_6 and rp_2 to v_2p_7. The angular region for the subtree S_2 is $p_6v_2p_7$. Similarly, the regions of S_1 and S_3 are $p_4v_1p_5$ and $p_8v_3p_9$ respectively. Note that the region is unbounded, and one can draw edges with arbitrary length in a region without introducing any crossing with the edges of other regions. The algorithm continues this process recursively. For example, we assume that v_1 has three children $v_{1,1}, v_{1,2}$ and $v_{1,3}$. For simplicity we assume that each child subtree has an equal number of edges. The algorithm divides $p_4v_1p_5$ into three equal sized regions $p_4v_1p_{10}, p_{10}v_1p_{11}$ and $p_{11}v_1p_5$. We then place the children vertices in the corresponding regions and translate the regions.

Algorithm 6: Angular Region-wise Initialization(r, θ)

Result: A layout which is crossing free and preserves ideal edge length
Add the current node r;
Let n be the number of nodes in the tree;
for *each child* v *of* r **do**
  Let n_v be the nodes in the subtree;
  Angular region of $v, \theta_v = \theta \times \frac{n_v}{n}$;
  Initial layout(v, θ_v);
end

Appendix B: Final Overlap Removal

The second step of the algorithm does not necessarily remove all label overlaps. This is due to the the force-directed algorithm containing two different types of forces: collision force and edge force. In some cases, one force may act opposite to the other force., see Figure 13. For every remaining label overlap we run a post-processing step. 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 circle of radius 200. We denote the set of nodes in that circular region by V'. We then sample 100 random points from that circular region. For each of the 100 sample points we computed the distances from V'. Let p be the sample point that has the maximum minimum distance from V' and it will not introduce any crossing. If we find such a point, then we move v to p. In our experiment we set $\varepsilon = 1$.



(a) (b)

Figure 13: The corresponding edge of "NLP" has a larger desired edge length. Hence, due to edge force, the algorithm provides label overlaps as shown in Figure 13a. By applying post-processing we can remove such overlap as shown in Figure 13b.

Appendix C: More Images of Last.fm Graph

We include more images of the last.fm graph, including 14, 15, and 16 and encourage the reader to go to the website to explore more.

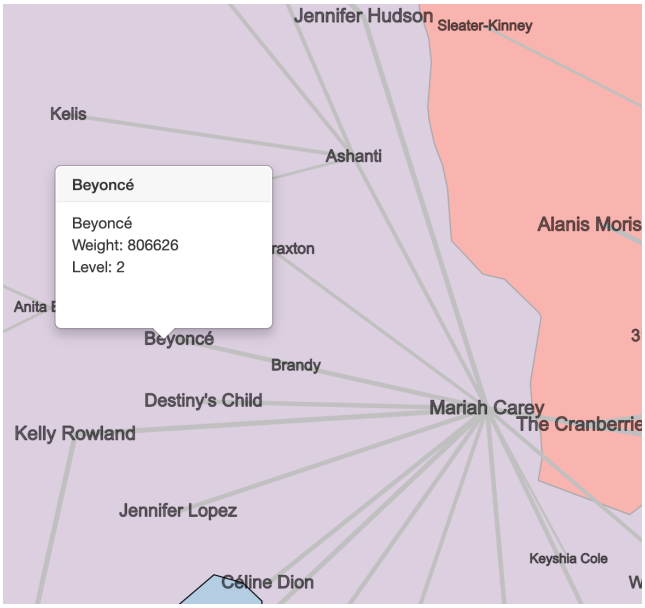


Figure 14: Node click in ZMLT visualization of the Last.fm graph.

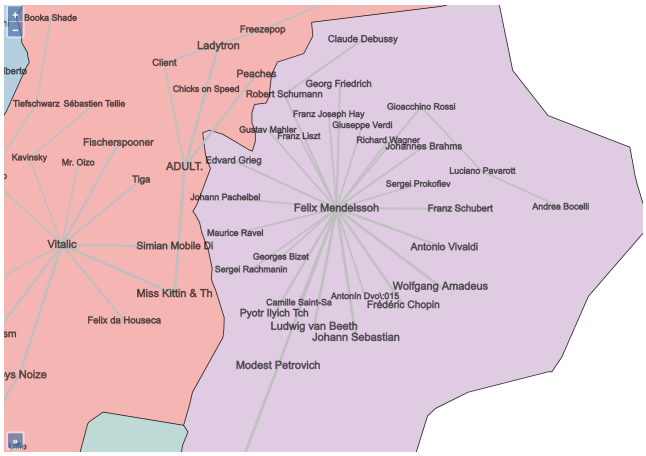


Figure 15: Classical Composers in Last.fm Graph.

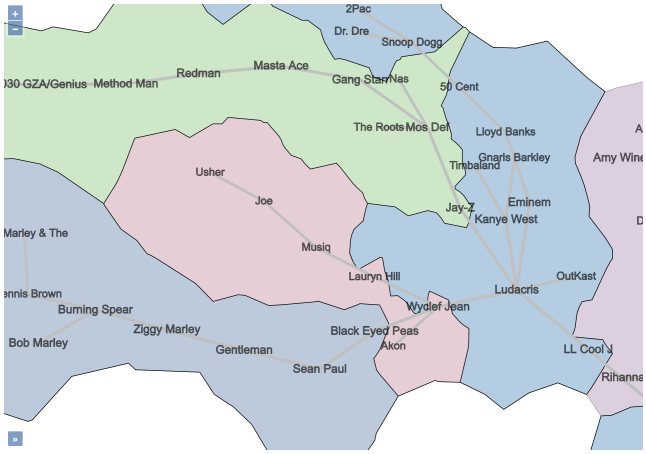


Figure 16: A view of several genres

Appendix D: More Images of the Google Topics Graph

We show zooming from the highest level to the lowest in the Google Topics graph in the following images: 17, 18,19, 20, and21.

Appendix E: Evaluation

We include images that show the total output from each of our algorithms and the direct approach. These show each level (T_1 through T_8). The images serve to highlight the differences in the layout using each method.

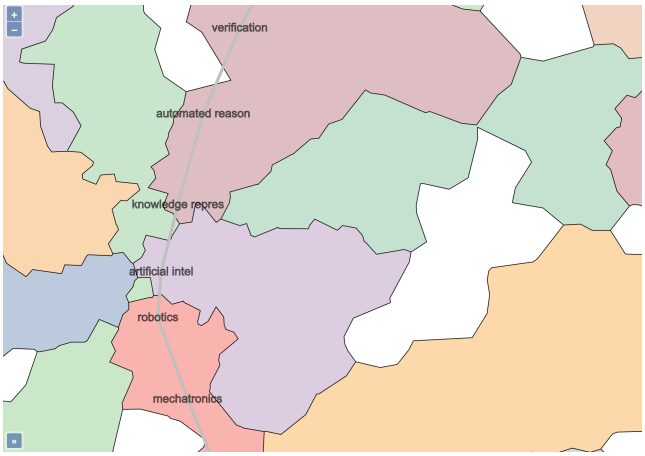


Figure 17: A zoomed out view, showing only the highest level

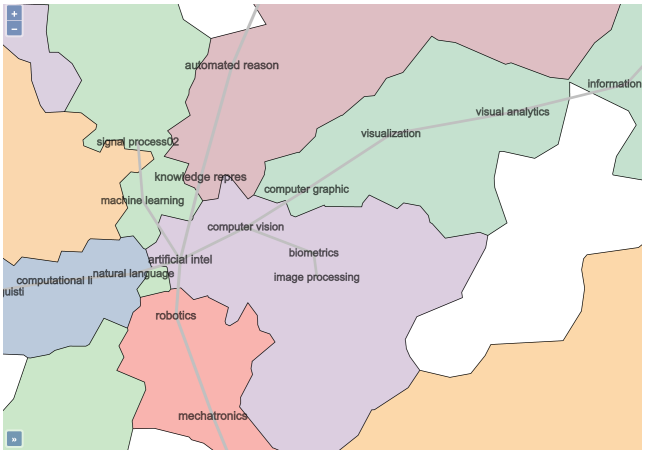


Figure 18: A zoomed out view showing two levels

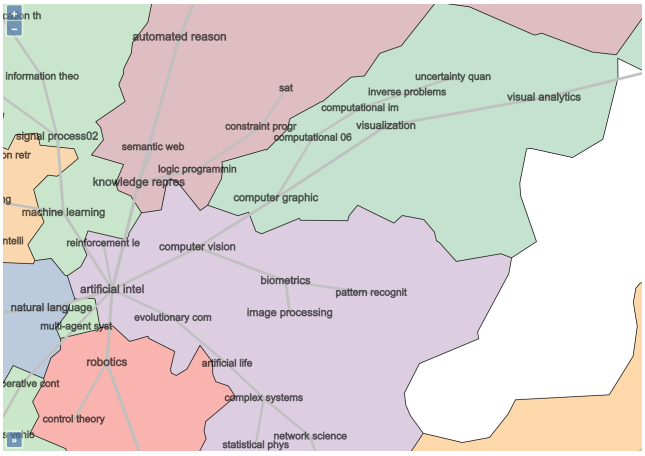


Figure 19: A view showing levels 1-4

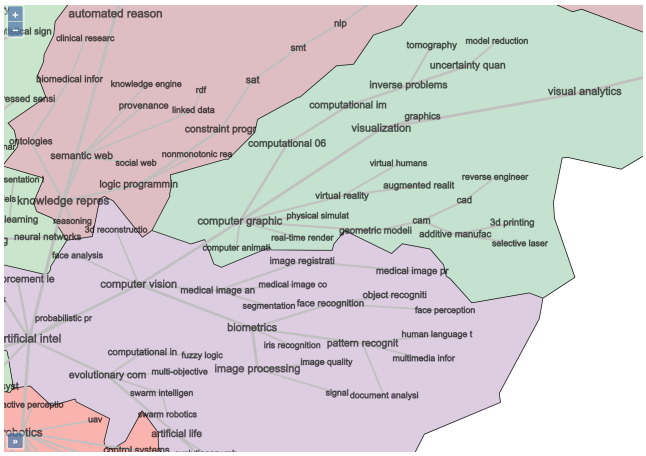


Figure 20: A view showing levels 1-6

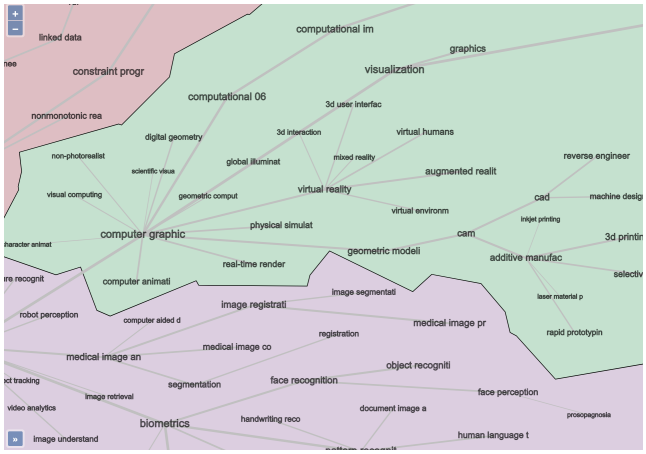
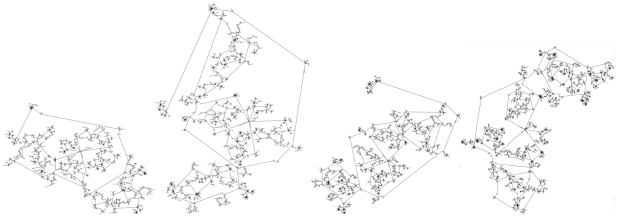


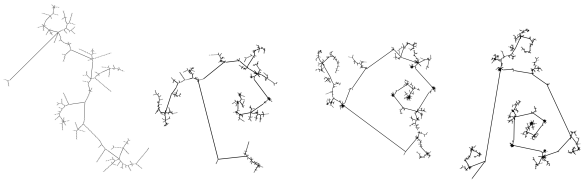
Figure 21: A view showing all levels

(a) T_1 (b) T_2 (c) T_3 (d) T_4

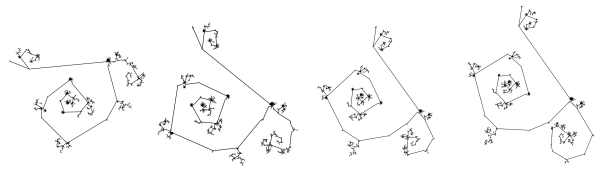


(e) T_5 (f) T_6 (g) T_7 (h) T_8

Figure 22: Overview of the tree hierarchy structure of the Google Topics graph drawn with the Direct Approach.

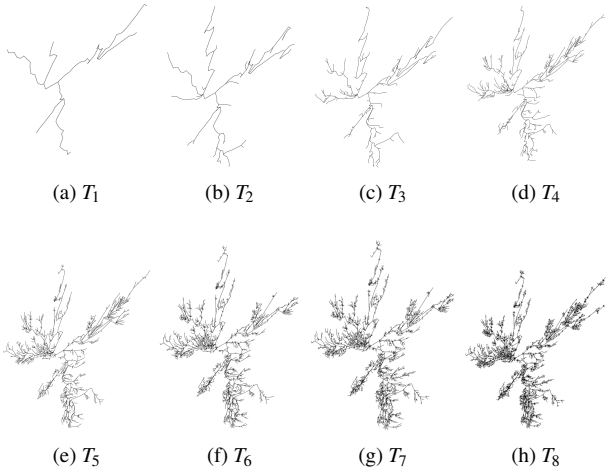


(a) T_1 (b) T_2 (c) T_3 (d) T_4



(e) T_5 (f) T_6 (g) T_7 (h) T_8

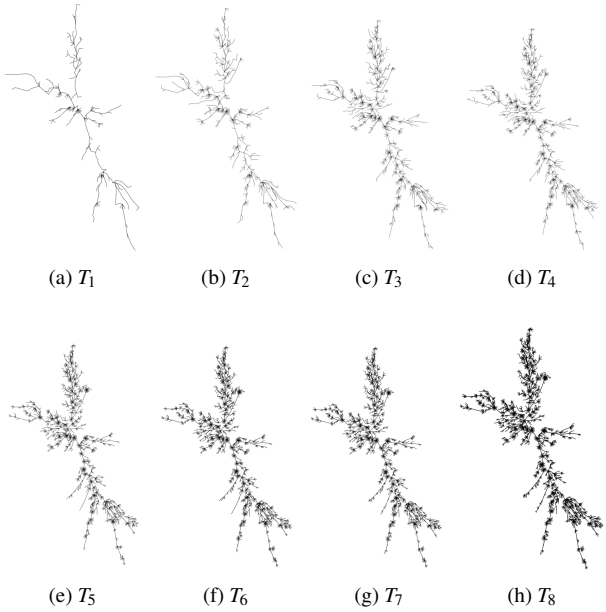
Figure 23: Overview of the tree hierarchy structure of the Last.fm graph drawn with the Direct Algorithm.



(a) T_1 (b) T_2 (c) T_3 (d) T_4

(e) T_5 (f) T_6 (g) T_7 (h) T_8

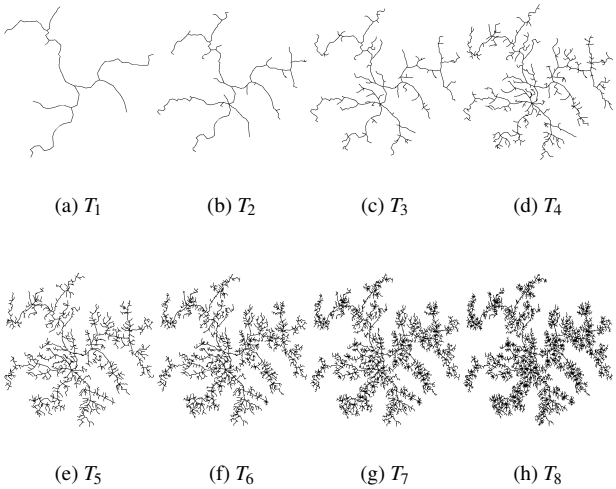
Figure 24: Overview of the tree hierarchy structure of the Google Topics graph drawn with DELG Algorithm.



(a) T_1 (b) T_2 (c) T_3 (d) T_4

(e) T_5 (f) T_6 (g) T_7 (h) T_8

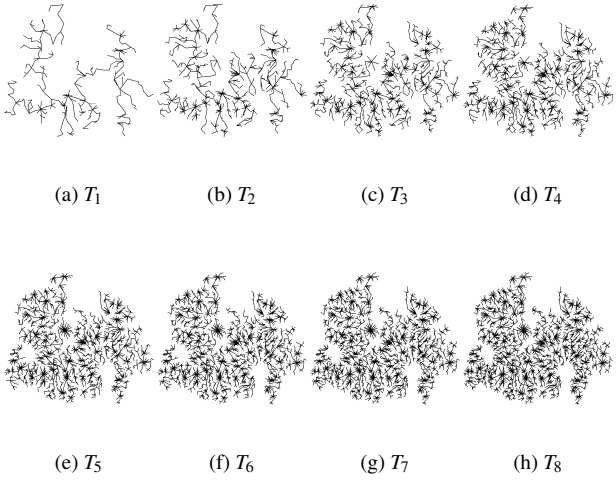
Figure 25: Overview of the tree hierarchy structure of the Last.fm graph drawn with DELG Algorithm.



(a) T_1 (b) T_2 (c) T_3 (d) T_4

(e) T_5 (f) T_6 (g) T_7 (h) T_8

Figure 26: Overview of the tree hierarchy structure of the Google Topics graph drawn with CG Algorithm .



(a) T_1 (b) T_2 (c) T_3 (d) T_4

(e) T_5 (f) T_6 (g) T_7 (h) T_8

Figure 27: Overview of the tree hierarchy structure of the Last.fm graph drawn with CG Algorithm.