Multi-level tree based approach for interactive graph visualization with semantic zoom

Felice De Luca(✉)[1], Iqbal Hossain[1], Stephen Kobourov[1], and Katy Börner[2]

[1] Department of Computer Science, University of Arizona
[2] Department of Information and Library Science, Indiana University

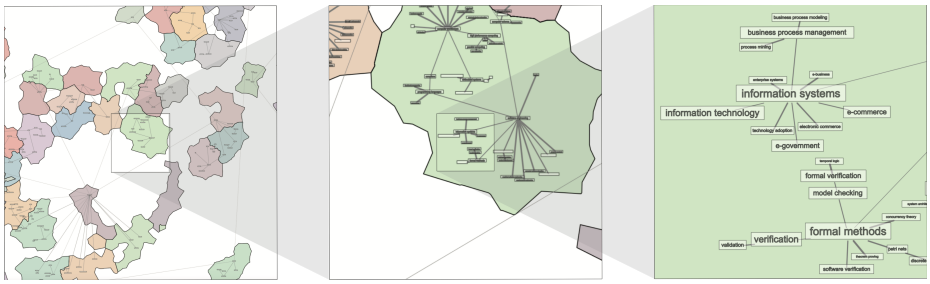


Fig. 1: Map-based semantic-zoom graph visualization using a multi-level tree that is representative, real, persistent, overlap-free labeled, planar, and compact.

Abstract. A recent data visualization literacy study shows that most people cannot read networks that use hierarchical cluster representations such as "supernoding" and "edge bundling." Other studies that compare standard node-link representations with map-like visualizations show that map-like visualizations are superior in terms of task performance, memorization and engagement. With this in mind, we propose the *Zoomable Multi-Level Tree* (*ZMLT*) algorithm for map-like visualization of large graphs that is representative, real, persistent, overlap-free labeled, planar, and compact. These six desirable properties are formalized with the following guarantees: (1) The abstract and embedded trees represent the underlying graph appropriately at different level of details (in terms of the structure of the graph as well as the embedding thereof); (2) At every level of detail we show real vertices and real paths from the underlying graph; (3) If any node or edge appears in a given level, then they also appear in all deeper levels; (4) All nodes at the current level and higher levels are labeled and there are no label overlaps; (5) There are no crossings on any level; (6) The drawing area is proportional to the total area of the labels. This algorithm is implemented and we have a functional prototype for the interactive interface in a web browser.

1 Introduction

Every day millions of people use maps to find restaurants, theaters, shops, the best routes to reach them. Google Maps, Apple Maps, OpenStreetMap are only some of the available services that use maps and all of them share in common the standard map interactions via panning and zooming. Panning moves the map in any direction while

keeping it at the same scale. Zooming changes the scale of the visualization by adding or removing details for a specific area. These two interactions are very familiar to people and provide a natural way to interact with large graphs. An important feature of map visualizations is that when zooming in or out, the information density remains roughly the same and the type of information (e.g., cities and roads) is also the same. Graph visualization systems that rely on meta-nodes (or super-nodes, cluster-nodes, etc.) and meta-edges (or edge bundling) violate this principle. With this in mind, we propose the *Zoomable Multi-Level Tree* (*ZMLT*) algorithm for map-like visualization of large graphs that is (1) representative, (2) real, (3) persistent, (4) overlap-free labeled, (5) planar and (6) compact:

1. The abstract and embedded trees represent the underlying graph appropriately at different levels of detail (in terms of the structure of the graph and the embedding thereof).
2. Real vertices and real paths from the underlying graph are shown at every level of detail.
3. If any node or edge appears in a given level, then they also appear in all deeper levels.
4. All nodes at the current and higher levels are labeled and there are no label overlaps.
5. There are no crossings on any level.
6. The drawing area is proportional to the total area of the labels.

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 [3], where we create the vertex filtration $V_1 \subset V_2 \subset \cdots \subset V_n = V$ with the most important vertices (highest weight) in V_1, the next most important vertices 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 Fig. 2. Note that extracting the best tree T_i that spans vertex set V_i may require the use of vertices that are not in V_i, which are called Steiner vertices.

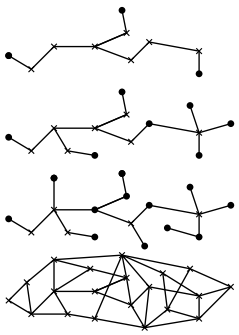


Fig. 2: Multi-level Steiner tree.

Just as in interactive map visualizations, at a minimum level of zoom, only the most important cities and highways are visible, and these are provided by the tree T_1 in our setting. 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 maps is shown (the entire underlying graph G, in our setting). The six guarantees above are also embodied in the ZMLT layout, which is implemented and has a functional prototype for the interactive interface in a web browser; see Fig. 1.

The rest of this paper is organized as follows: In Section 2 we describe related work, in Section 3 we give the details of ZMLT and in Section 4 we describe the visualization

tool to navigate and interact with the output of ZMLT. In Section 4.2 we compare the results of our approach to existing techniques. We conclude in Section 5.

2 Related Work

Large Graph Visualization: Most graph layout algorithms use a force-directed [26,28] or a stress model [12,44] 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 [29, 38] or use parallel computation as in VxOrd [11, 22]. GraphViz [27] uses the force-directed method sfdp [40] that combines the multi-scale approach with a fast n-body simulation [7]. Stress minimization was introduced in the more general setting of multidimensional scaling (MDS) [45] and has been used to draw graphs as early as 1980 [58]. Simple stress functions can be efficiently optimized by exploiting fast algebraic operations such as majorization. Modifications to the stress model include the strain model [61], PivotMDS [12], COAST [32], and MaxEnt [33]. 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 [13].

Graph layout algorithms are provided in libraries such as GraphViz [27], OGDF [18], MSAGL [51], and VTK [56], but they do not support interaction, navigation, and data manipulation. Visualization toolkits such as Prefuse [39], Tulip [5], Gephi [8], and yEd [62] support visual graph manipulation, and while they can handle large graphs, their rendering often does not: even for graphs with a few thousand vertices, 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 inculdes ASK-GraphView [2], topological fisheye views [1, 36], and Grokker [53]. Software applications such as Pajek [24] for social networks, and Cytoscape [59] for biological data provide limited support for multi-level network visualization. Zinsmanier *et al.* [64] 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-vertices and meta-edges, which make interactions such as semantic zooming, searching, and navigation counter-intuitive, as shown by Wojton *et al.* [63]. 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 [54, 55]. In this context, *GraphMaps* [48] visualizes a graph using the map metaphor and provides different zoom levels, driven by an embedding of the entire underlying graph. GRAM [14] also visualizes a graph using the map metaphor and provides different zoom levels but neither GraphMaps nor GRAM guarantees all six 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 Fig. 3.

Overlap Removal and Topology Preservation : In theory, vertices are usually treated as point objects, but in practice, vertices are labeled and these labels must be shown in

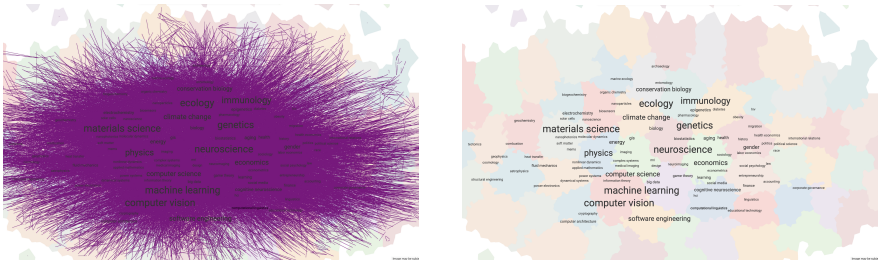


Fig. 3: Screenshots from the GRAM [14] system with and without edges.

the layout. Overlapping labels is a major problem for most graph layout algorithms, and it severely affects the usability of the visualization. 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 significantly modified layout with increased stress, larger drawing area, and with the introduction of crossings even when the starting vertex configuration is crossings-free.

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.* [47] 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 [35], Gansner and Hu [31], and Nachmanson *et al.* [50] 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.

Post-processing a layout while preserving the topology of the input layout is a difficult task. *Pred* [9] is a post-processing force directed algorithm applied on a given layout that aims to improve layout quality metrics while preserving the number of crossings. *Impred* [60] is an improved version that speeds-up the force computation and provides better results. This algorithm is not designed to remove the node overlaps but, with some appropriate modifications described in Section 3.4, it can be used for that purpose, while ensuring that the number of crossings remains unchanged.

Steiner Trees: The Steiner tree problem is one of Karp's original 21 NP-Complete problems [41]. A polynomial time approximation scheme exists if the underlying graph is planar [10], but not for general graphs [19]. More importantly, the Steiner tree can be efficiently approximated within a factor of $\ln 4 + e < 1.39$ [16]. In the vertex-weighted Steiner tree problem, the cost of the tree is determined by the weights of the vertices included in the solution (rather than the weights of the edges). This version of problem is also NP-hard [49] but can be approximated within a logarithmic factor [37, 42]. The classic Steiner tree problem has also generalized to a multi-level setting [6, 17, 20], 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*. [3] showed that the standard (edge-weighted) Steiner tree problem can be efficiently approximated

to within a constant factor. Similarly, Darabi *et al.* [21] described a multi-level version of the vertex-weighted Steiner tree, and showed that it can be approximated as well as the single-level version.

3 The Zoomable Multi-Level Tree Drawing Algorithm

As discussed in the previous section, there are many graph drawing algorithms and systems, and there is a also a great deal of work on visualizing trees; see *Treevis.net* [57] which summarizes more than 300 techniques. However, none of these can guarantee the six desirable layout properties that we need: representative, real, persistent, overlap-free labeled, planar and compact. For example, sfdp [40] produces great drawings but cannot guarantee that trees are drawn without crossings. Other algorithms designed specifically for crossing-free tree layouts (e.g., Walker, orthogonal, radial, hierarchical) do not produce representative views of the underlying graph, as they impose a layer-by-layer visualization in graphs where there is no such structure. Such algorithms are also not designed for labeled data. In order to guarantee overlap-free labeled layouts, these algorithms require significant scaling and this violates the compactness property; see Fig. 4.

(a) Orthogonal (b) Walker (c) Radial (d) Hierarchical

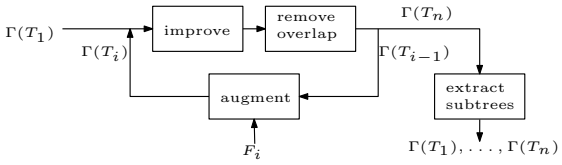
Fig. 4: Layouts of a the large tree T_8 that is used as a running example in this paper, produced by several dedicated tree drawing algorithms in Tulip [5].

With this in mind, we designed the Zoomable Multi-Level Tree (ZMLT) algorithm that guarantees all six of the desired properties. We begin with an overview of ZMLT as a block diagram shown in Fig. 5.

The input of this algorithm is a weighted graph with non-negative weights on edges and nodes and the general framework consists of the following steps:

1. *Layered tree extraction*: Given weighted graph, extract a set of trees that are persistent, real and representative of the graph.
2. *First-layer drawing*: Draw the first (smallest) tree in a planar fashion.
3. *Improve*: Improve the quality of the produced drawing while maintaining planarity.
4. *Remove overlap*: Remove label overlaps while maintaing planarity and aiming for compactness.
5. *Augment*: Add to the current tree the forest of subtrees needed to obtain the next (larger) tree, while maintaining palarity.

6. Repeat steps 3-5 until the last layer is reached.
7. *Extract subtrees*: Extract the layouts of all tress $T_1, T_2, \ldots$ from the drawing of final tree that spans all the vertices of the graph.



Symbol	Definition
G	Graph
T_i	Tree at level (layer) i
$E(X)$	Edges of X
$V(X)$	Vertices of X
F_i	Forest i. $T_i \setminus T_{i-1}$
$\Gamma(X)$	Drawing of X

Fig. 5: ZMLT framework to extract and compute layouts for the multi-level tree hierarchy (left) and a table of notation used (right).

3.1 Layered tree extraction

Input: Weighted graph G
Output: Set of trees $\{T_1, \ldots, T_n\}$ such that $T = T_1 \subset T_2 \subset \cdots \subset T_n$
Module: Multi-level Steiner tree extraction algorithm

This module deals with the construction of the (abstract) multi-level tree from the given weighted graph. We use a *multi-level vertex-weighted Steiner tree (MVST)* [21], defined and discussed in the Introduction. 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 high levels in the multi-level hierarchy. While the Steiner tree problem usually asks for the min cost tree, this is easy to fix by simply choosing the reciprocal of the weights (assuming all weights are positive and greater than 1). Note that the multi-level Steiner tree step above results in the creation of abstract (not yet drawn) trees. Before starting the actual layout computation, we preprocess the nodes and the edges. First, we assign different font sizes to the vertex labels, depending on the level they belong to. Second, we eliminate vertex and edge weights.

Rationale: The multi-level Steiner tree approach used in this module ensures the *representative* property, as important vertices and edges belong to the higher levels. Further the multi-level Steiner tree approach guarantees the *real* property as real vertices and real paths are represented in every tree. Finally, this approach also guarantees the *persistent* property, as by construction we have that $T_1 \subset T_2 \subset \cdots \subset T_n$.

Differentiating the font sizes helps identify the more important vertices even on deeper levels when there is a mix of nodes from different levels. We associate with each label the enclosing rectangle and these are used compute and remove overlap in later stages. We forget the weights from this point onward since they already played their role in the extraction of the multi-level Steiner tree.

3.2 First-layer drawing

Input: T_1.
Output: Planar $\Gamma(T_1)$
Module: Crossing free drawing algorithm.

The input at this step is the tree at the first layer of the hierarchy, namely T_1. Here the desired output is a planar drawing of the unweighted, unlabeled tree. Algorithms that guarantee a planar drawing of T_1 and do not impose additional features (such as hierarchical visualization) are suitable for this module.

Rationale: Here we aim to compute a *representative* layout for T_1 while ensuring the *planar* property.

3.3 Drawing-improvement

Input: $\Gamma(T_i)$.
Output: Improved $\Gamma(T_i)$.
Module: Modified *ImPred* algorithm.

This module improves the drawing of $\Gamma(T_i)$, the output of the previous step. The algorithm used for this module is *ImPred* [60], a force directed algorithm that aims to improve the layout of a drawing without changing the number of crossings in the input layout. To preserve the number of crossings of the input drawing, this algorithm uses three forces: (i) node-to-node repulsion force ($\mathbf{F}_v^r$), (ii) edge attraction force ($\mathbf{F}_v^a$), (iii) node-to-edge repulsion force ($\mathbf{F}_v^e$). The last force prevents a vertex from crossing any edge and ensures that no crossing is removed or added. We modified the desired edge-length distances in order to have longer edges at higher level trees and shorter edges for lower level trees.

Rationale: This step is particularly meaningful from T_2 and deeper, when an already drawn high-level tree T_i is combined with forest of subtrees from the next level to obtain T_{i+1}. As we will see in Step 3.5 (Augment layer), forest of subtrees is drawn in a sub-optimal way in order to ensure that it can be added while guaranteeing planarity. This makes the drawing improvement stage essential in the multi-level layout pipeline.

3.4 Remove label overlap

Input: $\Gamma(T_i)$
Output: $\Gamma(T_i)$
Module: Modified ImPred algorithm in *remove overlap mode*.

In this module we add labels and remove overlaps. We do this by adding another *Label-to-Label-repulsion (llrep)* force to ImPred. As in other force-directed algorithms, this force uses a repulsion force parameter δ. At each iteration we compute $llrep_{uv}$ for every pair of vertices u and v as follows. Let (x_u, y_u) and $2w_u, 2h_u$ be the coordinate, box width, and height of a vertex u, respectively. Let (x'_u, y'_u) be the new force of u. This repulsion force is zero when vertices u and v do not overlap each other i.e., $w_u + w_v - abs(x_u - x_v) < 0$ or $h_u + h_v - abs(y_u - y_v) < 0$. Otherwise the force is applied in the direction of the line through the vertices $(x'_u, y'_u) = (-\delta, s*(x_u - \delta - x_v))$ and $(x'_v, y'_v) = (\delta, s * \delta + y_u - y_v)$, here $s = \frac{y_v - y_u}{x_v - x_u}$. Note that this approach reduces label overlap iteratively. In our experiments all overlap can be removed, but we also post-process by scaling in order to guarantee that there are no overlapping labels.

Rationale: ImPred, with the additional Label-to-Label-repulsion force, works best locally. Thus the progressive multi-layer drawing algorithm has a good chance of removing overlaps without needing to resort to scaling.

3.5 Augment-layer

Input: $\Gamma(T_i)$, F_{i+1}.
Output: Preliminary drawing of tree T_{i+1}.
Module: Monotone drawing algorithm to add the forest.

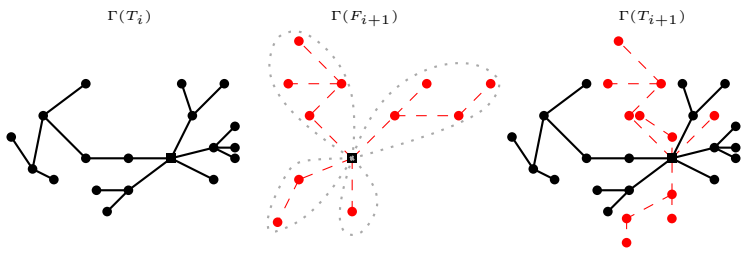


Fig. 6: Augmentation: $\Gamma(T_i)$ is augmented with $\Gamma(F_{i+1})$ to obtain $\Gamma(T_{i+1})$. In this example, the common vertex of T_i and F_{i+1} is shown as a square. Each subtree rooted in the common vertex (enclosed by dotted gray region) is drawn in monotone way and placed in the cones of $\Gamma(T_{i+1})$ around the common vertex, based on the size and width of the cones. If crossings occur the subtree is scaled down until there are no crossings.

This step adds to T_i the vertices and edges of the forest F_{i+1}, so that $T_i \cup F_{i+1} = T_{i+1}$. The goal of this module is to start with a good layout of T_i and initial layouts of the forest F_{i+1} and create a planar drawing of T_{i+1}.

A monotone drawing of a tree is a straight-line drawing such that, for every pair of vertices, there exists a path that monotonically increases with respect to some direction [4]. Since a monotone drawing of a tree is always a planar drawing, we use it to add the drawing of $\Gamma(F_{i+1})$ to $\Gamma(T_i)$ yielding $\Gamma(T_{i+1})$.

Forest F_{i+1} is made of trees rooted in vertices of T_i and those are the only vertices in common between F_{i+1} and T_i. The common vertices define where the rooted trees should be placed in the input drawing. Each such subtree is placed in the middle of the cone defined by two consecutive edges around the common vertex, in decreasing order by size of the tree and the size of the cone. After placing the monotone drawing of the subtree a crossing check is performed. If a crossing is generating by this placement, the subtree is scaled down. This process continues until adding the subtree creates no crossings and until all components of the forests are placed; see Fig. 6.

Rationale: The main objective here is to combine T_i with F_{i+1} and obtain a planar drawing of T_{i+1}. The improvement of the layout and overlap removal are performed in other steps.

3.6 Extract subtrees

Input: $\Gamma(T_n)$, all edges
Output: $\Gamma(T_1), \ldots, \Gamma(T_n), \Gamma(G)$
Module: Adding edges.

This step actually computes the drawings for all the trees in the multi-level hierarchy. We simply extract from $\Gamma(T_n)$ the layouts for all subtrees $\Gamma(T_1), \ldots, \Gamma(T_n)$. We also generate the layout for the entire graph, $\Gamma(G)$, by adding to T_n all the edges of G.

4 Interactive Web Browser Interface and Prototype

In this section we describe a functional browser interface and a prototype deployment of ZMLT with a specific dataset.

4.1 Web Browser Interface

The output the algorithm contains the positions of all the nodes in all the trees, as well as in the underlying graph. We modify the node-link representation into a map-like one using *GMap* [30]. Finally we extract GeoJSON files and use openlayers to display the data in the browser. We briefly describe the process.

Node Clustering: By default, we use MapSets [43] to cluster the graph and show the clusters as contiguous regions on the map, although other GMap [30] options are also available.

Generating Map Layers: GeoJSON [15] 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*, *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, 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 is 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; see the Fig. 7.

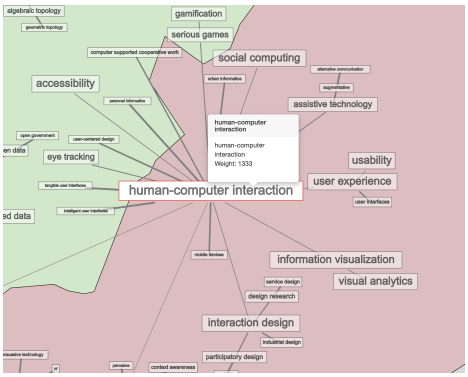


Fig. 7: Node click in ZMLT visualization.

4.2 Prototype ZMLT Deployment

In this section we show the results of our technique applied to the GRAM Topics Network [14]. This network is a weighted topic graph extracted from Google Scholar with 5947 nodes and 26695 edges. We used the ZMLT algorithm with 8 levels, determined by vertex weights. Tree T_1 contains the heaviest 5% of the vertices, tree T_2 contains the heaviest 15% of the vertices, and so on until tree T_8 which contains all vertices.

The exact percentages used are $(5, 15, 30, 40, 60, 70, 85, 100)$. All vertices are labeled using 8 different font sizes. The vertices in tree T_1 have font size 30, the vertices in tree T_2 have font size 25, and so on until tree T_8 where vertices have font size 8. The exact font sizes are $(30, 25, 20, 15, 12, 10, 9, 8)$. The prototype is available here: `http://uamap-dev.arl.arizona.edu:8086/zmlt.html`.

4.3 Evaluation

We compare ZMLT to a simpler implementation relying on off-the-shelf tools using several quality metrics.

Direct Approach: Our goals of ensuring planarity and labeling all vertices at the current and higher levels 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 drawing trees in a radial tree layout fashion [62]. We produced this layout by drawing T_8 using the 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 *Direct Approach*.

Quality Metrics: ZMLT uses the graph weights only for the extraction of the trees. As both the direct approach and ZMLT draw unweighted trees, then we use the following three (unweighted) metrics:

- Edge length uniformity (`EU`) evaluates the normalized standard deviation of edge lengths in the layout;
- Stress (`ST`) captures how well the geometric distances in the layout between pairs of vertices reflect their graph-theoretic distances in the graph;
- Blank space (`BS`) measures the ratio between the sum of the areas of labels and the area of the drawing.

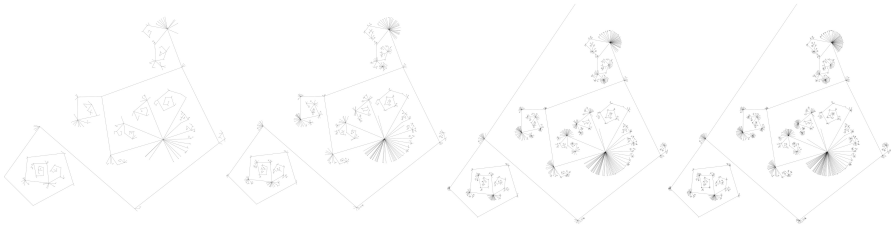
These metrics[3] have been used in many prior graph drawing studies [12, 23, 25, 34, 44, 46, 52]. Given Γ, a straight-line drawing of graph G, `EU` and `ST` are defined as follows:

$$\texttt{EU}(\Gamma) = \sqrt{\sum_{e \in E} \frac{(l_e - l_{avg})^2}{|E| l_{avg}^2}} \qquad \texttt{ST}(\Gamma) = \sum_{i,j \in V} w_{ij} (\parallel p_i - p_j \parallel - d_{ij})^2$$

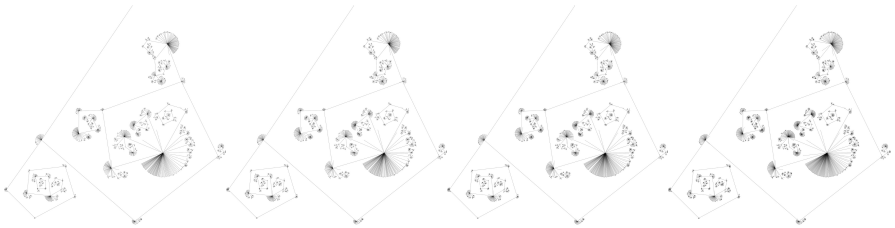
where l_e is the length of edge e, l_{avg} is the average length of all edges, p_i and p_j are the coordinates of vertices i and j in Γ, d_{ij} is the graph theoretic distance between vertices i and j in G and $w_{ij} = d_{ij}^{-2}$.

Results: Figure 8 shows the layouts of each layer computed by the direct approach while Fig. 9 shows the layouts produced by our technique. Even though details are impossible to see, some significant differences stand out visually. For example, in order to achieve compact layout, the direct approach wraps the trees around in a spiral fashion, which is not a good representation for the underlying graph. There are also very large differences in the lengths of the edges produced by the direct approach.

[3] All metrics are available at `https://github.com/felicedeluca/graphmetrics`

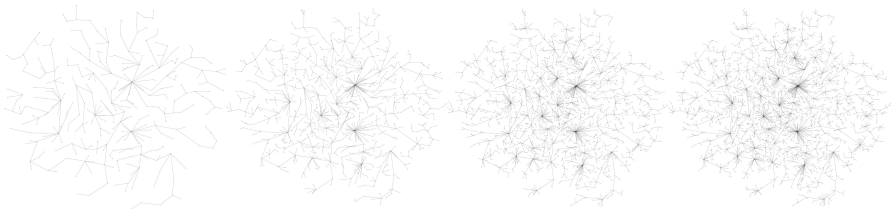


(a) T_1 (b) T_2 (c) T_3 (d) T_4

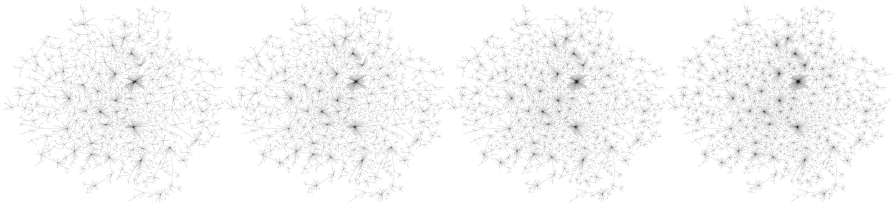


(e) T_5 (f) T_6 (g) T_7 (h) T_8

Fig. 8: Tree hierarchy 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

Fig. 9: Tree hierarchy drawn with ZMLT.

The metric analysis in Table 1 confirms the observations above. The table reports the values for the three measures computed on each of the 8 trees (representing the graph on different level of detail). Low values in all metrics (EU, ST, BS) correspond to better results and ZMLT has lower values in every table entry.

	EU	ST	BS
T_1	1.77	15502	9670
T_2	2.07	123164	4130
T_3	2.46	479681	2860
T_4	2.52	851388	2480
T_5	2.54	1894046	2100
T_6	2.55	2552986	1980
T_7	2.58	3736676	1840
T_8	2.60	5138014	1740

Direct Approach

	EU	ST	BS
T_1	0.46	6844	1590
T_2	0.63	54857	760
T_3	0.71	213369	466
T_4	0.74	376005	405
T_5	0.76	842806	346
T_6	0.77	1141010	326
T_7	0.78	1680726	303
T_8	0.80	2326448	287

ZMLT

Table 1: Metric-based layout evaluation of the Direct Approach and ZMLT.

5 Conclusions and Future Work

We presented ZMLT, a multi-level graph visualization approach based on the map metaphor which guarantees that the layout has the following properties: (1) representative, (2) real, (3) persistent, (4) labeled overlap-free, (5) planar and (6) compact. ZMLT extracts from a given graph a multi-level tree using a multi-level Steiner tree approach which are used to provide semantic zooming for interaction with the underlying graph. code for extracting the multi-level tree, generating layouts, measuring quality, and the interactive visualization system are available at `https://github.com/enggiqbal/mlgd`. The prototype is available here: `http://uamap-dev.arl.arizona.edu:8086/zmlt.html`.

There are several natural directions for future work. To leverage the map metaphor better, we would like to represent edges and paths in a way that more closely resembles a road-network. Defining *highwayness* and optimizing it will likely lead to better visualization and interaction. ZMLT currently relies on multi-level trees, but perhaps a better approach would be use a multi-level graph spanner (sparse subgraphs that approximately preserve shortest path distances).

References

1. Abello, J., Kobourov, S., Yusufov, R.: Visualizing large graphs with compound-fisheye views and treemaps. pp. 431–441 (2005)
2. Abello, J., Van Ham, F., Krishnan, N.: Ask-GraphView: A large scale graph visualization system. Visualization and Computer Graphics, IEEE Transactions on 12(5), 669–676 (2006)

3. 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: D'Angelo, G. (ed.) 17th International Symposium on Experimental Algorithms (SEA 2018). Leibniz International Proceedings in Informatics (LIPIcs), vol. 103, pp. 15:1–15:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2018), http://drops.dagstuhl.de/opus/volltexte/2018/8950
4. Angelini, P., Colasante, E., Di Battista, G., Frati, F., Patrignani, M.: Monotone drawings of graphs. In: Brandes, U., Cornelsen, S. (eds.) Graph Drawing. pp. 13–24. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
5. 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: Alhajj, R., Rokne, J. (eds.) Encyclopedia of Social Network Analysis and Mining, pp. 1–28. Springer (Aug 2017)
6. Balakrishnan, A., Magnanti, T.L., Mirchandani, P.: Modeling and heuristic worst-case performance analysis of the two-level network design problem (1994)
7. Barnes, J., Hut, P.: A hierarchical O(N log N) force calculation algorithm. Nature 324, 446–449 (Dec 1986)
8. Bastian, M., Heymann, S., Jacomy, M.: Gephi: An open source software for exploring and manipulating networks (2009)
9. Bertault, F.: A force-directed algorithm that preserves edge crossing properties. In: Kratochvíyl, J. (ed.) Graph Drawing. pp. 351–358. Springer Berlin Heidelberg, Berlin, Heidelberg (1999)
10. Borradaile, G., Kenyon-Mathieu, C., Klein, P.: A polynomial-time approximation scheme for steiner tree in planar graphs. In: SODA. vol. 7, pp. 1285–1294 (2007)
11. Boyack, K.W., Klavans, R., Börner, K.: Mapping the backbone of science. Scientometrics 64(3), 351–374 (2005)
12. Brandes, U., Pich, C.: Eigensolver methods for progressive multidimensional scaling of large data. In: Graph Drawing. pp. 42–53. Springer (2007)
13. Brandes, U., Pich, C.: An experimental study on distance-based graph drawing. In: Graph Drawing. pp. 218–229. Springer (2009)
14. 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. pp. 31:1–31:9. AVI '18, ACM, New York, NY, USA (2018)
15. Butler, H., Daly, M., Doyle, A., Gillies, S., Schaub, T., Schmidt, C.: The geojson format specification. Rapport technique 67 (2008)
16. 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. pp. 583–592. ACM (2010)
17. Charikar, M., Naor, J., Schieber, B.: Resource optimization in qos multicast routing of real-time multimedia. IEEE/ACM Transactions on Networking 12(2), 340–348 (April 2004)
18. 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 pp. 543–569 (2011)
19. Chlebík, M., Chlebíková, J.: The steiner tree problem on graphs: Inapproximability results. Theoretical Computer Science 406(3), 207–214 (2008)
20. Chuzhoy, J., Gupta, A., Naor, J.S., Sinha, A.: On the approximability of some network design problems. ACM Trans. Algorithms 4(2), 23:1–23:17 (May 2008), http://doi.acm.org/10.1145/1361192.1361200
21. 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 arXiv:1811.11700 (Nov 2018)

22. 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. pp. 23–30 (2001)
23. De Luca, F., Di Giacomo, E., Didimo, W., Kobourov, S., Liotta, G.: An experimental study on the ply number of straight-line drawings. In: Poon, S.H., Rahman, M.S., Yen, H.C. (eds.) WALCOM: Algorithms and Computation. pp. 135–148. Springer International Publishing, Cham (2017)
24. De Nooy, W., Mrvar, A., Batagelj, V.: Exploratory social network analysis with Pajek, vol. 27. Cambridge University Press (2011)
25. Dunne, C., Ross, S., Shneiderman, B., Martino, M.: Readability metric feedback for aiding node-link visualization designers. IBM Journal of Research and Development 59, 14:1–14:16 (05 2015)
26. Eades, P.: A heuristic for graph drawing. Congressus Numerantium 42, 149–160 (1984)
27. 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. pp. 483–484. Springer-Verlag (2001)
28. Fruchterman, T.M.J., Reingold, E.M.: Graph drawing by force-directed placement. Software: Practice and Experience 21(11), 1129–1164 (1991)
29. Gajer, P., Goodrich, M., Kobourov, S.: A fast multi-dimensional algorithm for drawing large graphs. Computational Geometry: Theory and Applications 29(1), 3–18 (2004)
30. Gansner, E.R., Hu, Y., Kobourov, S.: Gmap: Visualizing graphs and clusters as maps. In: 2010 IEEE Pacific Visualization Symposium (PacificVis). pp. 201–208 (March 2010)
31. Gansner, E.R., Hu, Y.: Efficient node overlap removal using a proximity stress model. In: Tollis, I.G., Patrignani, M. (eds.) Graph Drawing. pp. 206–217. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
32. Gansner, E.R., Hu, Y., Krishnan, S.: Coast: A convex optimization approach to stress-based embedding. In: Graph Drawing. pp. 268–279. Springer (2013)
33. Gansner, E.R., Hu, Y., North, S.: A maxent-stress model for graph layout. Visualization and Computer Graphics, IEEE Transactions on 19(6), 927–940 (2013)
34. Gansner, E.R., Koren, Y., North, S.: Graph drawing by stress majorization. In: Pach, J. (ed.) Graph Drawing. pp. 239–250. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
35. Gansner, E.R., North, S.C.: Improved force-directed layouts. In: Proceedings of the 6th International Symposium on Graph Drawing. pp. 364–373. GD '98, Springer-Verlag, Berlin, Heidelberg (1998), http://dl.acm.org/citation.cfm?id=647550.729069
36. Gansner, E., Koren, Y., North, S.: Topological fisheye views for visualizing large graphs. TVCG 11(4), 457–468 (July 2005)
37. Guha, S., Khuller, S.: Improved methods for approximating node weighted steiner trees and connected dominating sets. Information and computation 150(1), 57–74 (1999)
38. Hadany, R., Harel, D.: A multi-scale algorithm for drawing graphs nicely. Discrete Applied Mathematics 113(1), 3–21 (2001)
39. 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. pp. 421–430. ACM (2005)
40. Hu, Y.: Efficient, high-quality force-directed graph drawing. Mathematica Journal 10(1), 37–71 (2005)
41. Karp, R.M.: Reducibility among combinatorial problems. In: Complexity of computer computations, pp. 85–103. Springer (1972)
42. Klein, P., Ravi, R.: A nearly best-possible approximation algorithm for node-weighted steiner trees. Journal of Algorithms 19(1), 104–115 (1995)
43. Kobourov, S.G., Pupyrev, S., Simonetto, P.: Visualizing graphs as maps with contiguous regions. EuroVis14, Accepted to appear 4 (2014)

44. 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. pp. 137–144. IEEE (2002)
45. Kruskal, J.B., Wish, M.: Multidimensional Scaling. Sage Press (1978)
46. Maaten, L.v.d., Hinton, G.: Visualizing data using t-sne. Journal of machine learning research 9(Nov), 2579–2605 (2008)
47. Marriott, K., Stuckey, P., Tam, V., He, W.: Removing node overlapping in graph layout using constrained optimization. Constraints 8(2), 143–171 (Apr 2003)
48. 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,. pp. 108–119. INSTICC, SciTePress (2018)
49. Moss, A., Rabani, Y.: Approximation algorithms for constrained node weighted steiner tree problems. SIAM Journal on Computing 37(2), 460–481 (2007)
50. Nachmanson, L., Nocaj, A., Bereg, S., Zhang, L., Holroyd, A.E.: Node overlap removal by growing a tree. J. Graph Algorithms Appl. 21(5), 857–872 (2017)
51. Nachmanson, L., Robertson, G., Lee, B.: Drawing graphs with GLEE. In: Graph Drawing. pp. 389–394. Springer (2008)
52. Ortmann, M., Klimenta, M., Brandes, U.: A sparse stress model. In: Hu, Y., Nöllenburg, M. (eds.) Graph Drawing and Network Visualization. pp. 18–32. Springer International Publishing, Cham (2016)
53. Rivadeneira, W., Bederson, B.B.: A study of search result clustering interfaces: Comparing textual and zoomable user interfaces. Studies 21, 5 (2003)
54. Saket, B., Scheidegger, C., Kobourov, S.G., Borner, K.: Map-based visualizations increase recall accuracy of data. COMPUTER GRAPHICS FORUM 34(3), 441–450 (2015)
55. Saket, B., Simonetto, Kobourov, S., Börner, K.: Node, node-link, and node-link-group diagrams: An evaluation. IEEE Transactions on Visualization & Computer Graphics pp. 2231–2240 (2014)
56. Schroeder, W.J., Avila, L.S., Hoffman, W.: Visualizing with VTK: a tutorial. Computer Graphics and Applications 20(5), 20–27 (2000)
57. Schulz, H.: Treevis.net: A tree visualization reference. IEEE Computer Graphics and Applications 31(6), 11–15 (Nov 2011)
58. Seery, J.B.: Designing network diagrams. In: General Conf. Social Graphics. vol. 49, p. 22 (1980)
59. 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), 2498–2504 (2003)
60. Simonetto, P., Archambault, D., Auber, D., Bourqui, R.: Impred: An improved force-directed algorithm that prevents nodes from crossing edges. Computer Graphics Forum 30(3), 1071–1080 (2011)
61. Torgerson, W.S.: Multidimensional scaling: I. theory and method. Psychometrika 17(4), 401–419 (1952)
62. Wiese, R., Eiglsperger, M., Kaufmann, M.: yfiles—visualization and automatic layout of graphs. In: Graph Drawing Software, pp. 173–191. Springer (2004)
63. 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)
64. 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), 2486–2495 (Dec 2012)