Drawing Clustered Graphs as Topographic Maps

Martin Gronemann and Michael Jünger

Institut für Informatik, Universität zu Köln, Germany
{gronemann,mjuenger}@informatik.uni-koeln.de

Abstract. The visualization of clustered graphs is an essential tool for the analysis of networks, in particular, social networks, in which clustering techniques like community detection can reveal various structural properties.
In this paper, we show how clustered graphs can be drawn as topographic maps, a type of map easily understandable by users not familiar with information visualization. Elevation levels of connected entities correspond to the nested structure of the cluster hierarchy.
We present methods for initial node placement and describe a tree mapping based algorithm that produces an area efficient layout. Given this layout, a triangular irregular mesh is generated that is used to extract the elevation data for rendering the map. In addition, the mesh enables the routing of edges based on the topographic features of the map.

1 Introduction

Clustered graphs are able to express relationships between entities and, at the same time, hierarchies on those entities in the form of a nested system of sets of entities called clusters. This special combination has turned out to be very useful in many areas. A prominent example is the analysis and visualization of large software systems, in which the cluster hierarchy is usually formed by source code elements like classes, packages, or libraries. Visualization of software systems can reveal and prevent structural weaknesses of a system design. Our main motivation, however, has been the analysis and visualization of social networks. Here, the entities are persons whose bilateral relationships are captured by edges while nested communities can be expressed by clusters. In particular, citation and collaboration networks have been subjects of increased attention in recent years. Such networks can offer insights in the publication behavior in different research areas or can be used to measure the performance of people, institutions or other entities.

Clustered graph drawing is difficult because the relational data and the hierarchy must be visualized in one picture. For the relational data alone, a variety of graph drawing methods is available, and for the cluster structure alone, various tree drawing methods can be applied, because by their nested nature, the cluster system corresponds to a tree with the set of entities as the root. Graphs are usually drawn using geometric shapes for the nodes corresponding to the entities, and the relations correspond to the edges that connect nodes by lines or curves. For trees, similar methods can be applied, but there exist alternatives that can be used to create appealing drawings. Combining both in one drawing without producing visual clutter or occlusion is a difficult task.

Users of clustered graph drawing in the analysis and visualization of large software systems are typically software engineers, i.e., people who are used to (and happy with) technical drawings. In contrast, users of social network analysis and visualization are much less likely familiar with technical drawings. We have been wondering what would be the most intuitive clustered graph visualizations for these "non-technical" people and came up with the idea of trying to visualize social network data as topographic maps. In fact, hardly anyone can avoid learning to read "real" maps already as a child. Modern services like Google Earth or Google Maps have certainly boosted this ability. Therefore, such an approach is likely to find interest also outside social network analysis and visualization.

Topographic maps are detailed graphic representations of the physical features of an area. This makes them a crucial tool for land, air and nautic navigation. The graphical representation often includes contours or isoclines. These lines follow the contours of the terrain at a specific elevation level to visualize terrain features like hills and valleys. In addition, elevation levels have natural color encodings.

In this paper, we describe how we succeeded to draw clustered graphs as topographic maps. The basic idea is to generate a landscape where the nodes in different subtrees of the cluster hierarchy are separated by water, a valley, or a rift. With increasing distance from the root cluster, nodes will be located in the lowlands, the highlands, and ultimately, on mountain peaks. Given this basic idea, the contribution of this paper is to present and discuss a method for creating such drawings.

The remainder is structured as follows. Section 2 reviews related work on visualization of clustered graphs, tree mapping, edge routing and graph drawing in general. In Section 3, we present the major components of our method. As our main contribution, we will then

- describe the method for node placement based on the cluster hierarchy,
- show how we generate a 2.5D triangle mesh based on this placement and the cluster hierarchy,
- and explain how we use this mesh as a basis for an edge routing graph with whose help the edges can be drawn as quadratic curves following the terrain features.

In Section 4 we provide details of our implementation and the tools we used, including a short discussion of the performance. Finally, we present the results in Section 5 by applying our technique to selected instances followed by a conclusion and outlook in Section 6.

2 Related work

The method we propose builds on a number of previous developments in graph drawing and information visualization in general, of which we cover the most influential aspects.

A ground-breaking early contribution to automatic graph drawing has been an article by Eades [8] in which he proposed a heuristic that is based on the idea that the nodes repel each other and the edges act as steel springs. The resulting layout is now called *force-directed layout*, a term introduced in the article of Fruchterman and Reingold [9]. Force-directed layout is widely used. After more than two decades of development,

there exist algorithms and according software tools that, based on the original idea, can deal with large graphs, see, e.g., Hachul et al. [16].

Also tree drawing is a well-studied problem and many layout styles and related algorithms exist in the literature. As an alternative to the node-link representation, Johnson and Shneiderman have proposed *tree maps* [17]. Most algorithms for generating tree maps follow the same principle. A predefined area/shape is recursively subdivided into smaller parts according to the input tree. The result is a nested set of shapes contained in the root shape. Most algorithms use rectangular shapes. The work of de Berg et al. [1] provides a detailed analysis of how to partition convex polygons such that the resulting shapes have a good aspect ratio. We use some of their results in this paper and, therefore, describe these in more detail in Section 3.

The visualization of clustered graphs that combine ordinary graphs with cluster trees has been studied quite intensively and many different approaches and styles exist, see [6] for an introduction. Garland et al. [11] combine centroidal Voronoi diagrams and force directed layouts to allocate screen space more efficiently. The proposed technique uses a hierarchical structure that produces Voronoi tree map-like layouts.

Ganser et al. [10] have proposed *GMap*, a system for drawing graphs as maps. Given a node partition, they show how to draw a graph as a political map using a Voronoi diagram based shape. The initial graph layout neglects the additional information given by the partition, like basic force-directed methods. Furthermore, a defragmentation algorithm is used in order to achieve a compact shape of the countries. In addition, sophisticated map coloring techniques are applied.

A very different approach for using maps in multidimensional data visualization are *self organizing maps (SOM)* [18]. A machine learning algorithm is used to distribute objects on a two dimensional map. Based on a feature vector, similar objects are placed close to each other. The different feature areas are encoded by colors.

In [19], Kuhn et al. propose a system for visualizing software based on the vocabulary used in the components. They use *Multidimensional Scaling (MDS)* to map a layout in a high-dimensional vector space down to two dimensions and draw thematic software maps. These maps are realized by creating a hill for every entity according to a normal distribution function. Instead of color encoded elevation information, contour lines and hill shading is used for visualizing the terrain features. Cortese et al. [5] use the topographic map metaphor for visualizing hierarchical networks in a radial style. Some real world networks are very dense so that bundling edges is necessary to reduce visual clutter. In [20], Lambert et al. use a hybrid quad tree/Voronoi approach for routing and bundling edges for a given layout. Qu et al. [24] use a Delaunay triangulation based approach. The bundling is achieved by clustering intersection points of the graph edges and Delaunay edges or by collapsing Delaunay edges. An even more adaptive approach is described by Dwyer and Nachmanson [7], where a visibility graph is used as a routing network.

3 The Algorithm

Before describing the different steps of our technique for drawing topographic maps, we briefly recall the definition of a clustered graph. A *clustered graph* consists of a graph

$G = (V, E)$ with node set V and edge set E as well as a tree $T = (V_T, E_T)$ whose leaves are exactly the nodes in V. Every inner node $C \in V_T$ of the tree is referred to as a *cluster node* that defines a subset of V consisting of all leaves of the subtree of T rooted at C. Thus T defines a hierarchy consisting of a nested system of subsets of V called *clusters*, the *root cluster* V corresponds to the root of T. Every cluster is thus endowed with a *cluster hierarchy level* that is the graph-theoretic distance of its corresponding cluster node to the root of T, the root cluster's hierarchy level is 0 and clusters whose corresponding cluster nodes have maximum distance from the root are at the top of the hierarchy. We shall later assign an elevation level to each node of G equal to the maximum cluster hierarchy level of the clusters it belongs to.

3.1 Clustering Method

The clustering method establishes an indirect link between the edge set and hierarchy. We will later apply the fat polygon partition, which is not aware of the underlying graph, and requires this link. In the following we will give a brief description of the clustering algorithm we use.

For our instances we used the edge betweenness based algorithm proposed by Girvan and Newman [14]. Edge betweenness measures the number of shortest paths an edge is part of. See [3] for details on betweenness and variants. The algorithm of Girvan and Newman [14] for detecting communities is rather simple. It calculates betweenness for all edges and removes the one with the highest score. This procedure is repeated until the graph becomes disconnected. The connected components serve as clusters with cluster hierarchy level 1. The partitioning is recorded by creating the corresponding cluster subtrees and we can recurse on the connected components to obtain clusters with cluster hierarchy level 2, and so on, until no edges are left to remove.

The intuition is that if a part of the graph is well connected, the number of shortest paths using a given edge in this part is relatively low. Conversely, this number is relatively high in sparse parts of the graph. Thus edges connecting clusters will have a higher betweenness score compared to intra cluster edges. Removing edges with higher betweeness increases this effect until the graph becomes disconnected. We modified the above algorithm slightly such that in case there are multiple edges with the highest betweenness, we remove all of them simultaneously. This modification avoids artifacts in the hierarchy caused by some internal order of the edges.

We also use this algorithm for decomposing collaboration networks with weighted edges. The weights correspond to the amount of collaboration between two authors. For a weighted version of betweenness, see Brandes [3]. After some experiments we found that the most intuitive integration, that is to divide the final betweenness score of an edge by its weight, gives the best results.

3.2 Fat Polygon Partitioning Placement

In the following, we describe the placement of the nodes based on the work of de Berg et al. [1]. The basic idea of *fat polygon partitioning* is to create a nested structure of convex polygons for a weighted binary tree. In our case, the weight is chosen as the area required for a subtree, i.e. the number of graph nodes contained in the corresponding

cluster. At each internal node, a cutting line is chosen that subdivides the subtree's boundary polygon into two convex subpolygons with the appropriate area depending on the weights of the two children. We place the graph nodes, i.e., the leaves in the tree, in the centroid of the boundary polygon computed by the partitioning. The objective is to subdivide in a top down manner while obtaining polygons with a small aspect ratio. Following the notation of [1], the aspect ratio of a convex polygon P is defined as asp(P) $=$ diam$(P)^2/$area(P), where diam(P) is the diameter and area(P) the area of P. The diameter of a polygon P is the maximum distance between two vertices of P.

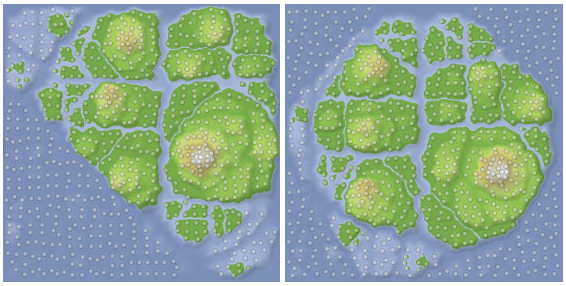
In the following, we describe our implementation of the greedy algorithm presented in [1] to obtain a partition with a good aspect ratio. For more details we refer the reader to the original paper [1]. We are given a convex polygon P with k vertices and a parameter $0 < a \leq \frac{1}{2}$, where a is the fraction of the area we require for the smaller child. We want to find a direction for a cutting line that partitions P into two subpolygons P_1 and P_2 such that area$(P_1) = a \cdot$ area(P) and area$(P_2) = (1 - a) \cdot$ area(P).

When given a cut direction, we choose the orientation of the cut perpendicular to this direction. Finding such a cut is easy for a convex polygon. However, for a given cut direction and $a < 1/2$, we can always cut in two ways. In that case, we choose the cut where the maximum aspect ratio of the two polygons is minimized. In order to find the direction for a cut with a good aspect ratio, de Berg et al. [1] distinguish between two main cases.

In the first case, when we want to cut off a small piece, that is when $a \leq 1/k^2$, we take the bisector at the vertex with the smallest interior angle as cut direction. In the second case, when $a > 1/k^2$, we work a little harder for finding a balanced cut. In this case the proof in [1] contains two subcases that distinguish different shapes of P. When P has a good aspect ratio, i.e., asp$(P) \leq k^6$, we are allowed to choose any direction for a cut. In order to avoid too many diagonal cuts along the diameter, we cut horizontally or vertically, depending on which results in the best aspect ratio. Otherwise, that is when $a > 1/k^2$ and asp$(P) > k^6$ holds, we choose the direction of the line representing the diameter of the polygon. This is the line that connects any two vertices of P with the maximum distance to each other.

The above algorithm constructs a proper polygon partition based on the structure and weights of the tree. We can now place each graph node at the centroid of its polygon and obtain a layout for the next step. But two problems arise: First, the shapes of the clusters do not look very appealing. In order to fix that, we apply some post processing to the boundary polygons after their computation, but before they are used as boundary polygons for their children. The idea is to round off sharp corners by cutting off small fractions, in our implementation about two percent of a polygon. Investing small amounts of area for a better shape is inspired by the partitions with slack in [1], here for general convex polygons rather than rectangles.

The second problem is that we can only use a binary tree, thus a transformation of a non binary tree into a binary tree is required. We are in the lucky situation that most of the internal nodes of the tree generated by the aforementioned cluster algorithm are binary, but some of them are not. We replaced the original method of [1] by another variant proposed later by de Berg et al. [2]. The presented algorithm for creating a binary tree produces unpleasant drawings in some cases. The reason is that many real-



(a) Fully balanced (b) Right heavy balanced

Fig. 1: Comparison of the two balancing approaches. The left picture (a) shows the original balancing. In (b), the result of our modified version is shown.

world instances like, e.g., the graph drawing collaboration network [13] contain one big connected component and many small ones. The algorithm classifies the subtree containing the big connected component as a "heavy" child, putting it alone in one subtree. All the other smaller connected components are assigned to the second subtree. When this node is then partitioned, the large connected component ends up on one side, while all the small components are located on the opposite side (see Figure 1a).

In order to solve this problem, a slightly different transformation is used. The idea is to push the big child further down the tree. In the given order, we assign nodes to the first subtree until the weight ratio of that subtree exceeds c/k^2 for some constant c (our implementation uses $c = 2$). If no such node exists, the smallest element is used. Then we assign the rest to the second subtree, and recurse on these. As a result, the small pieces will fill up the corners first, and the large ones will be placed last, thus in the middle. Figure 1b displays the effect of this strategy.

Using the above algorithm results in a convex polygon for each node of the tree. These polygons are nested according to the hierarchy provided as input. For all leaves we calculate the centroids of their polygons and place the corresponding nodes there. This layout is then used as an input for the next step, the creation of the mesh that will be used to model the map.

3.3 Mesh Generation

As a first step, a conforming Delaunay triangulation for the nodes and the boundary polygon is constructed. Before we show how to obtain the triangle mesh that models the terrain features, we give some insights into the relationship of triangles and clusters. Consider a triangle of the Delaunay triangulation like displayed in Figure 2 and the lowest common ancestors in the cluster tree. We observe the following properties:

- We can associate with each edge $e = (u, v)$ a cluster C_e with $C_e = \text{LCA}(u, v)$
- At least two of the three associated clusters are equal.

The first property is easy to see. Each vertex in the triangulation corresponds to a leaf in the cluster tree or is an additional point inserted for the conforming Delaunay trian-

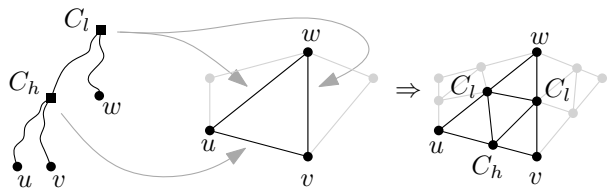


Fig. 2: Mapping of the cluster tree nodes onto the triangulation.

gulation. In the latter case, the vertex is part of the boundary polygon and we associate it with the root. Thus the lowest common ancestor is a cluster node.

The second property is based on the fact that for three nodes u, v, and w, the LCA of two pairs of three possible pairs must be the same. When considering the tree in Figure 2, it becomes clear that without loss of generality

$$\text{LCA}(\text{LCA}(u, v), w) = \text{LCA}(u, w) = \text{LCA}(v, w)$$

holds. The idea of associating clusters with edges makes the subdivision of the triangles straightforward. We split a triangle into four subtriangles by splitting each edge in the middle (see Figure 2 for details). To each newly inserted node, we assign the cluster associated with the split edge.

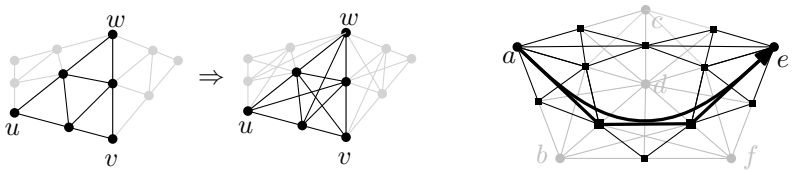
All nodes of the triangle mesh are now either cluster or graph nodes. These mesh nodes are now "lifted" by computing their *elevation levels* that are simply chosen as their cluster hierarchy levels. The result is that each cluster node forms a valley because it is an inner node of the tree, thus has less distance to the root. The graph nodes of the mesh form peaks surrounded by cluster nodes which are all lower.

This mesh will serve as a basis for both drawing the map and routing the edges. Before we describe the edge routing, we make some further improvements to the mesh. Since the shape of the elevation model might be a bit coarse, we further subdivide the triangles and move the newly inserted vertices closer to the corner points. The idea is that, for aesthetic reasons, wide valleys should have a flat bottom instead of a very light slope. This results in hills with a more uniform slope and the flat area in valleys increases.

The result of the procedure described above is a 2.5D triangle mesh that covers seamlessly the complete boundary polygon. A software rasterizer is then used to interpolate the elevation levels given at the corners of a triangle to obtain a raster based representation of the elevation map.

3.4 Edge Routing

Now we use the mesh as a routing network that depends on the terrain features. The idea is to apply a shortest path based edge routing algorithm that is aware of the clustering by using the terrain mesh as a routing network. We follow a very general framework that consists of two major steps, namely, the construction of the routing network and the computation of a shortest path for each edge in order to obtain the control points for the curve.



(a) (b)

Fig. 3: In (a) the extra edges connecting the corners with the opposite vertex are displayed. On the right (b) an example of a routing network with the subdivisions and shortcut edges is shown. Allowed edges and nodes are drawn black, forbidden edges and nodes are drawn gray.

Recall from the mesh generation section that we have subdivided each triangle by inserting vertices that correspond to clusters in the tree. Each graph node is associated with one node in the routing network. However, for a cluster, there are usually many nodes in the routing network. After some experiments it turned out that the generated curves look more pleasant when we allow shortcuts. Extra edges connect the graph nodes with the opposite cluster node, allowing a direct connection without making unnecessary turns when using one of the adjacent cluster nodes. Figure 3a displays a triangle of the routing network with the shortcut edges.

The costs for the routing edges used by the shortest path algorithm are chosen as a mixture of the Euclidean distance in the plane and the elevation difference. The extra costs for the elevation difference makes the edge routing aware of the terrain features. For some constant c (we set $c = 1/4$ after some experiments) the distance function $d(e)$ for an edge $e = (u, v)$ is defined as:

$$d(e) = c \cdot (\text{level}(u) + \text{level}(v) - 2 \cdot \text{level}(\text{LCA}(u, v))) + (1 - c) \cdot \|p_u - p_v\|,$$

where the level and coordinates of a node are normalized.

Given an edge $e = (u, v)$, we are looking for a shortest path starting at the routing node that corresponds to u and ends at the routing node that corresponds to v. Only cluster nodes are allowed in-between, because we do not want an edge to be drawn through another node. Notice that the number of nodes and edges of the routing network are linear in the number of graph nodes.

An example is given in Figure 3b where the path is used as control points for a curve representing the edge from a to e. The black edges are those that allowed for a shortest path. The gray edges are forbidden in order to avoid drawing through a node.

4 Performance and Implementation Details

All graph and tree map related code has been written in C++ using the *OGDF - Open Graph Drawing Framework* [23]. The conforming Delaunay triangulation for the mesh is created with CGAL [4].

For rendering, we wrote a custom rasterizer that transforms the 2.5 dimensional mesh to a *digital elevation model (DEM)* for use in *GIS - geographic information systems*. Furthermore, data required for drawing nodes and edges are converted into in a more GIS-friendly format. These are then used as input for *Mapnik* [21], a tool for drawing "real" maps and developing mapping applications. This enables us to use other tools from the GIS tool chain like the *GDAL - Geospatial Data Abstraction Library* [12] that is used for extracting contour lines from the DEM and to calculate the shading for the resulting image.

Instance	$\|V\|$	$\|E\|$	$\|C\|$	Clustering	Fat Polygon	Edge Routing
gdea_2011	819	1 880	1 366	3.02	0.14	0.09
netscience	1 589	2 742	2 363	0.1	0.23	0.11
composers	2 815	11 429	4 708	4 888.42	1.22	5.50
PGPgiantcompo	10 680	24 316	16 640	-[1]	2.83	21.55
cond-mat	16 726	47 594	26 235	-[1]	4.03	70.18

Table 1: Runtime in seconds for the three phases for instances of different size.

The performance of our approach is mainly governed by a few steps. Table 1 shows the runtime[2] of the clustering algorithm, the fat polygon partitioning and the edge routing on a few instances.

The used clustering algorithm is not practical for large dense graphs. The algorithm suggested by Brandes [3] takes time $O(|V||E|)$ for calculating the betweenness, and this results in a total running time of $O(|V||E|^2)$. For the fat polygon partitioning, to our knowledge, no bound is known. But the time required suggests that it scales well, compared to the edge routing, even for larger instances with 16 726 nodes. The running time for the edge routing can be bounded by $O(|V|^2 \log |V|)$ when running a single source shortest path query for all edges incident to one node at the same time. For the instances presented here, this is acceptable. But with increasing graph size, the algorithm takes much more time and leaves room for improvement.

5 Results

In Figure 4, the co-authorship in the Proceedings of the International Symposium on Automatic Graph Drawing is visualized. The network is taken from the *Graph Drawing E-Print Archive* [13] and covers articles from 1995 to 2011. The graph has been constructed by creating edges between all authors of an article. The edge weight is set to $1/(\text{number of authors} - 1)$ for compensation, so that each paper contributes 1 unit of collaboration for an author. The map displays the different communities in graph drawing. Usually a mountain or hill contains one or two authors who have published not only many papers, but also for a long time and can be considered backbones of the

[1] Instances have been clustered in parallel on a different machine with better performance due to the input size.

[2] Machine with Core i7 2.7 GHz and 8 GB RAM

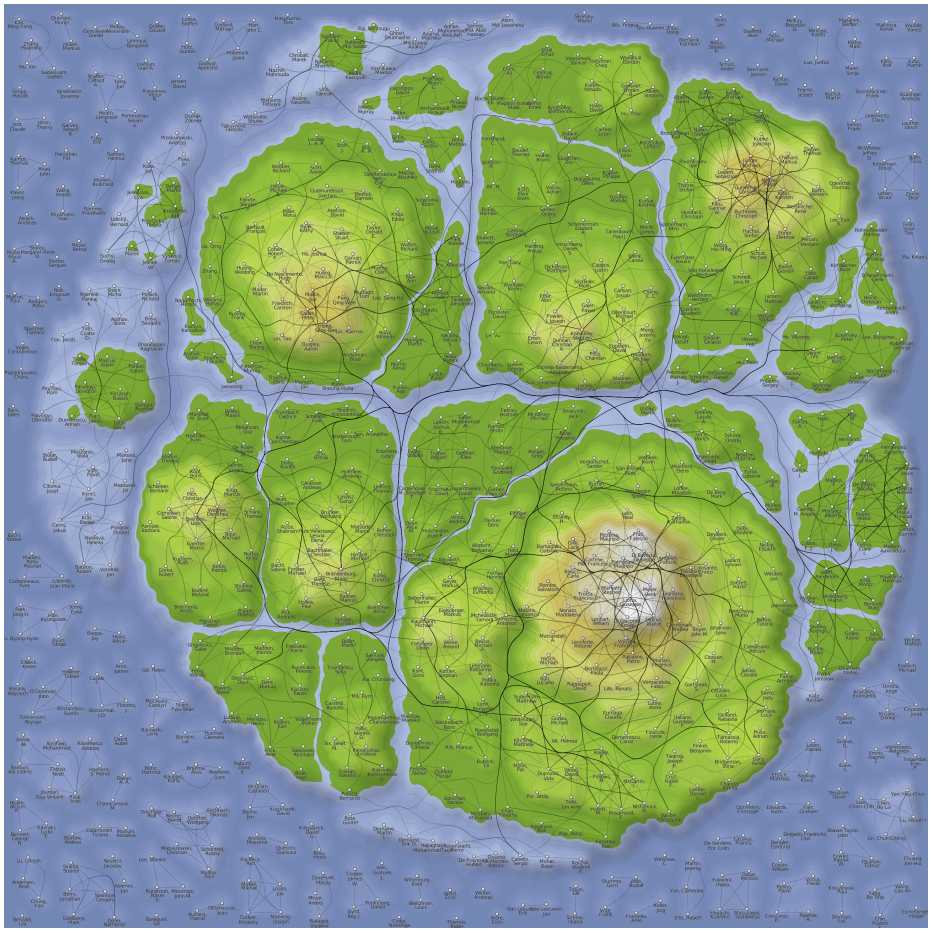


Fig. 4: Graph drawing collaboration graph with 819 nodes and 1880 edges.

research area. We have developed a prototype web interface for this network to offer a convenient way to explore authors, publications and their relations [13].

The graph in Figure 5 displays the co-authorship of scientists working in network science. Unlike the previous instance, this graph contains, besides one big connected component, a few medium sized components. The effect of the modified tree transformation on their shape is clearly visible.

6 Conclusion and Future Work

In this paper we have presented our approach for drawing clustered graphs as topographic maps. Besides the use of fat polygon partitioning, we showed how to extract a mesh from the layout that models the terrain features based on the clustering.

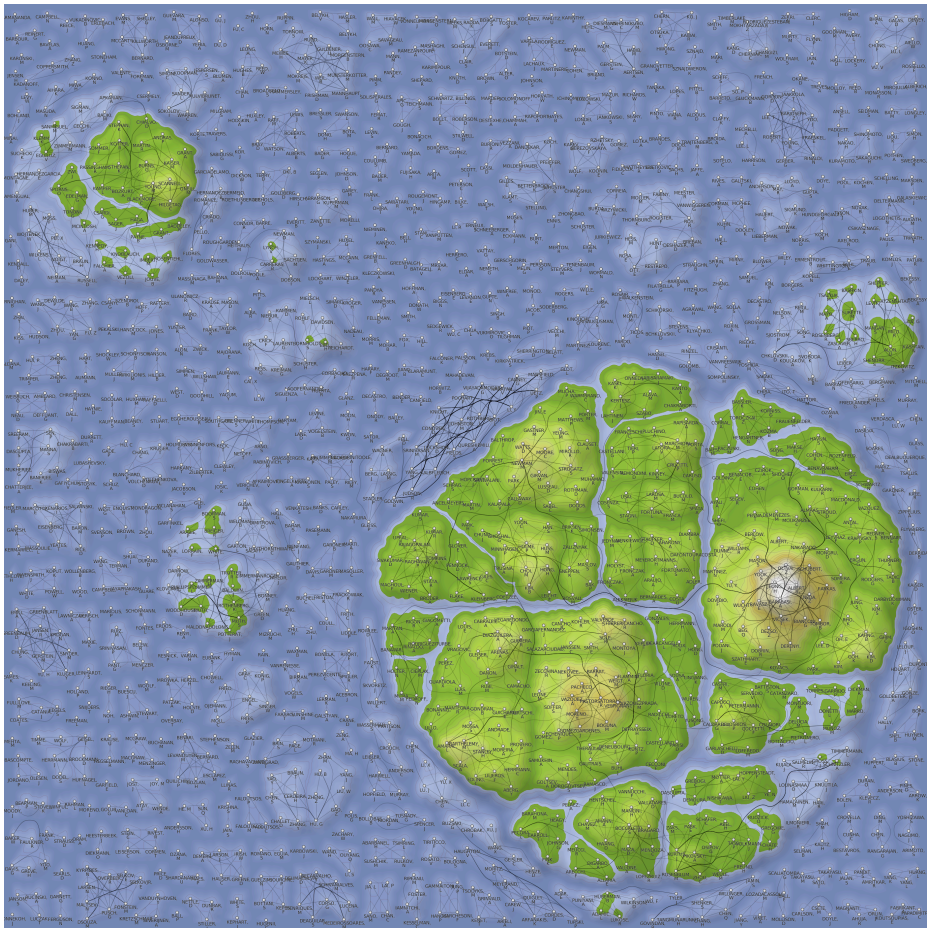


Fig. 5: Network science collaboration graph [22] with 589 nodes and 2742 edges.

As mentioned in the previous section, the clustering algorithm should be replaced by a more scalable approach for large instances. Furthermore, we believe that the visual appearance can be improved by modifying the tree map approach so that it produces more natural looking shapes. In addition, the placement by fat polygon partitioning is not directly aware of the adjacencies induced by the edges of the input graph. As a result, the placement relies heavily on the clustering method used and does not take edge length into account.

While the proposed method works well for the applications in Section 5, it may fail when the edges and the clustering are largely unrelated. Such an example is discussed in [15], and it is demonstrated how the degrees of freedom of the partitioning algorithm can be exploited with moderate extra effort in order to obtain pleasing maps due to increased “edge-awareness”.

References

1. de Berg, M., Onak, K., Sidiropoulos, A.: Fat polygonal partitions with applications to visualization and embeddings. CoRR abs/1009.1866 (2010)
2. de Berg, M., Speckmann, B., van der Weele, V.: Treemaps with bounded aspect ratio. In: Proceedings of the 22nd International Symposium on Algorithms and Computation. pp. 260–270 (2011)
3. Brandes, U.: On variants of shortest-path betweenness centrality and their generic computation. Social Networks 30(2), 136–145 (2008)
4. CGAL - Computational Geometry Algorithms Library, `http://www.cgal.org/`
5. Cortese, P.F., Battista, G.D., Moneta, A., Patrignani, M., Pizzonia, M.: Topographic visualization of prefix propagation in the internet. IEEE Trans. Vis. Comput. Graph. 12(5), 725–732 (2006)
6. Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: Graph Drawing: Algorithms for the Visualization of Graphs. Prentice Hall (1999)
7. Dwyer, T., Nachmanson, L.: Fast edge-routing for large graphs. In: Proceedings of the 17th International Symposium on Graph Drawing. pp. 147–158 (2010)
8. Eades, P.: A heuristic for graph drawing. Congressus Numerantium 42, 149–160 (1984)
9. Fruchterman, T.M.J., Reingold, E.M.: Graph drawing by force-directed placement. Softw. Pract. Exper. 21(11), 1129–1164 (1991)
10. Gansner, E.R., Hu, Y., Kobourov, S.G.: Gmap: Visualizing graphs and clusters as maps. In: PacificVis. pp. 201–208. IEEE (2010)
11. Garland, M., Kumar, G.: Visual exploration of complex time-varying graphs. IEEE Transactions on Visualization and Computer Graphics 12, 805–812 (2006)
12. GDAL - Geospatial Data Abstraction Library, `http://gdal.org/`
13. GDEA - Graph Drawing E-Print Archive, `http://gdea.informatik.uni-koeln.de/`
14. Girvan, M., Newman, M.E.J.: Community structure in social and biological networks. Proceedings of the National Academy of Sciences 99, 7821–7826 (2002)
15. Gronemann, M., Jünger, M., Kriege, N., Mutzel, P.: MolMap: Visualizing molecule libraries as topographic maps (2012), submitted to GD 2012
16. Hachul, S., Jünger, M.: Drawing large graphs with a potential-field-based multilevel algorithm. In: Proceedings of the 12th International Symposium on Graph Drawing. pp. 285–295 (2004)
17. Johnson, B., Shneiderman, B.: Tree-maps: a space-filling approach to the visualization of hierarchical information structures. In: Proceedings of the 2nd Conference on Visualization '91. pp. 284–291 (1991)
18. Kohonen, T.: Self-Organizing Maps. Springer (2000)
19. Kuhn, A., Loretan, P., Nierstrasz, O.: Consistent layout for thematic software maps. In: Proceedings of the 2008 15th Working Conference on Reverse Engineering. pp. 209–218 (2008)
20. Lambert, A., Bourqui, R., Auber, D.: Winding roads: Routing edges into bundles. Computer Graphics Forum 29(3), 853–862 (2010)
21. Mapnik, `http://mapnik.org/`
22. Newman, M.E.J.: Finding community structure in networks using the eigenvectors of matrices. Phys. Rev. E 74, 036104 (2006)
23. OGDF - Open Graph Drawing Framework, `http://www.ogdf.net/`
24. Qu, H., Zhou, H., Wu, Y.: Controllable and progressive edge clustering for large networks. In: Proceedings of the 14th International Symposium on Graph Drawing. pp. 399–404 (2007)