Institut für Theoretische Informatik
Lehrstuhl für Algorithmik I

Layout and Visualization of Large, Hierarchically Clustered Graphs

Studienarbeit
von

Jan Christoph Athenstädt

an der Fakultät für Informatik

Gutachterin:	Prof. Dr. Dorothea Wagner
Betreuende Mitarbeiter:	Dr. Martin Nöllenburg Dipl.-Inform. Marcus Krug Dr. Robert Görke

Bearbeitungszeit: 15. September 2011 – 14. Dezember 2011

KIT – Universität des Landes Baden-Württemberg und
nationales Forschungszentrum in der Helmholtz-Gemeinschaft

www.kit.edu

Abstract

We present a method to visualize large clustered graphs (>10,000 nodes). We first generate a 2D layout with regard to the existing cluster hierarchy and then visualize the layout in 3D with a landscape metaphor.
For the 2D layout, we use a force directed approach based on a modified Fruchterman-Reingold algorithm. Our goal is to represent single nodes and edges, as well as the overall structure and the levels of clustering. We thereby make sure that the internal structure of clusters becomes clear while taking into account inter-cluster edges. We also guarantee a layout-area for each cluster that is proportional to its size.
The 3D landscape is realized as a heightfield over the 2D layout. We use a Gaussian filter to generate the landscape, using characteristics of the layout such as node density as height information. The original graph is laid on top of the landscape with straight edges connecting the nodes.

Deutsche Zusammenfassung

Wir stellen ein Verfahren vor, um große, geclusterte Graphen (> 10.000 Knoten) zu visualisieren. Zunächst generieren wir ein 2D-Layout unter Berücksichtigung der bestehenden Hierarchie aus Clustern. Das generierte Layout visualisieren wir daraufhin in 3D mit Hilfe einer Landschafts-Metapher. Für das 2D-Layout benutzen wir ein kräftebasiertes Verfahren auf Basis des Fruchterman-Reingold-Algorithmus. Unser Ziel ist es, sowohl einzelne Knoten und Kanten als auch die Gesamtstruktur und die verschiedenen Clusterebenen darzustellen. Wir stellen dazu sicher, dass die interne Struktur von Clustern klar wird bei gleichzeitiger Berücksichtigung von Inter-Cluster-Kanten. Wir garantieren zudem eine Fläche für das Layout der einzelnen Cluster, die proportional zu deren Größe ist.
Die 3D-Landschaft wird als Höhenfeld über dem 2D-Layout generiert. Wir benutzen einen Gauss-Filter, um das Höhenfeld zu erstellen, wobei wir verschiedene Charakteristiken des Layouts, wie zum Beispiel die Knotendichte, als Höheninformation verwenden. Der ursprüngliche Graph wird anschließend über die Landschaft gelegt, wobei geradlinige Kanten die Knoten verbinden.

Contents

1. Introduction

1.1. Basic Concepts

Graphs are a fundamental concept not only in mathematics and computer science, but in many areas of research, as well as in our everyday lives. They provide a way to describe the relationships between objects. Mathematically, a graph $G = (V, E)$ consists of a set of vertices V and a set of edges E, where an edge is a pair $e = (u, v)$ with $u, v \in V$.

A typical example for data that can be represented as graphs are social networks. These can be either online platforms where every node represents a user and every edge their friendship-relations or they can be email-networks, where two nodes representing users are connected if the users have been exchanging mails.

Other examples for graphs include protein networks in biology or road- and railway-networks with cities as nodes and edges representing connections between the cities by roads or train routes.

While there are many known concepts and algorithms to analyze the characteristics of a graph, the easiest way for humans to get an idea of the structure and the relationships between nodes is a visual representation. Instead of the mathematical representation through adjacency lists or adjacency matrices, graphs can be drawn using dots to represent nodes and lines between these dots to represent edges. However, there is no single "right" way to draw a graph, and interpretation by humans might vary depending on the drawing.

Since the 1980s there have been a number of algorithmic approaches to graph drawing, including orthogonal drawings [Tam87] and circular layouts [DMM97]. However, the most common method for automatically generating a layout for an arbitrary graph are force-directed algorithms, first introduced by Peter Eades [Ead84]. These algorithms are based on a physical model and consider nodes as particles with repulsive forces pushing them away from each other. Edges are treated as "springs" between nodes, introducing attractive forces to pull connected nodes towards each other. The system is simulated over a limited number of discrete time steps starting from a random initial layout. These methods produce good results for small and medium sized graphs. For large graphs however, drawings based on force directed methods often seem very cluttered and overwhelming in their complexity.

To reduce the complexity of graphs and to classify nodes, a clustering of a graph can be helpful. The clustering organizes nodes into groups. These so called clusters can result

from given features of the nodes (e.g. the hometown of the user of a social network) or from a clustering algorithm that aggregates nodes that are strongly connected into a cluster. Newman describes the most important clustering methods in an article on "Detecting community structure in networks" [New04].

It is also possible to further aggregate clusters into higher level, more coarse grained clusters. Such a hierarchical clustering is also called *nested* clustering, since clusters on different levels "contain" each other. In the example with the hometown of a user, a higher level clustering could be the country in which the town is situated.

1.2. Contribution

We present an algorithm to lay out large, hierarchically clustered graphs and to visualize the layout as a 3D landscape. Our method allows users to interactively explore and manipulate the visualization. The goal is to provide a general overview of the graph and its cluster structure, while at the same time allowing to zoom in on details down to single nodes.

When we are given an unknown graph, we first want to get to know its general characteristics. Is it a very dense graph? Does it have a special geometric structure? Are there strongly connected parts that are clearly separated? Graph theoretical key figures are helpful, but a good visualization allows us to explore the graph and discover unknown properties that are not easily obtained from a theoretical analysis.

The problem with the visualization of large amounts of data is to find a good tradeoff between abstraction and completeness of the data to be shown. As stated in the "Visual lnformation-Seeking Mantra" by Shneiderman [Shn96] ("Overview first, zoom and filter, then details on demand."), it is important to aggregate features and show only the major parts at first in order not to overwhelm the user. Yet it should be possible to zoom in on details if needed. Clustering is a useful aggregation technique to emphasize the more global structure of the graph by dividing nodes into groups depending on their relationship to each other. We therefore focus on the visualization of clustered graphs.

When visualizing a clustering, there are two main aspects to take into account: the relationships between clusters and the internal structure of clusters. In Chapter 2 we describe a new algorithm to create a layout of the clustered graph in 2D space. When choosing our design goals, we kept the above aspects in mind. Based on our desiderata, we generate optically pleasing layout that is suitable for the generation of a 3D landscape. Our layout uses circular, non-overlapping areas for drawing clusters that correspond to the base shape of the hills we use for our landscape. We chose to set the area of the circles proportional to the number of nodes in the cluster.

On top of this 2D layout, we generate a 3D landscape (Chapter 3) that allows representing an additional attribute of the graph coded in the height-dimension. The goal is to make the graph more accessible through this representation and facilitate the detection of important features for the human mind. To support this, we extend the principle of abstraction and aggregation to our 3D representation. We use a Gaussian filter to smooth the landscape and suppress details for the sake of clarification. It is possible to change the size of the filter to increase or decrease the level of detail in (nearly) real-time. We implemented the generation of the landscape based on two different attributes: the density of nodes and the average degree of nodes in the neighborhood. We also added a feature to raise the landscape in the area covered by clusters to further emphasize the cluster structure in the 3D visualization.

Even though the representation as a landscape is a high degree of abstraction, the fact that we still show all nodes on top of the landscape allows exploring the original data by

zooming into the scene. We connect the nodes by straight edges, which make a distinction of single edges easier than in a 2D representation, since they can be viewed from different angles.

Our method allows the interactive exploration of the landscape as well as the rendering of still images or videos for presentations.

1.3. Previous Work

There have been several related papers on representing hierarchical data both in two and three dimensions as well as visualizing data with a landscape metaphor.

Eades et al. have published a number of algorithms for drawing clustered graphs. They describe a method on how to draw planar clustered graphs with convex cluster-regions [FCE95]. A later paper focuses on representations of the cluster levels on different layers in 3D space [EF97]. These methods focus on smaller, planar graphs though and provide mostly theoretical results, such as upper bounds for drawing areas.

Independently from our work, Didimo and Montecchiani [DM11] recently published a method for drawing large, hierarchically clustered graphs that uses similar design criteria to ours. They also use a multilevel approach but combine their force-directed technique with space-filling-algorithms. Like in our algorithm, they set cluster sizes proportional to the number of nodes they contain, although their layout regions are rectangular and not circular like ours. Their focus is mainly set on speed, which is why they only use a rough approximation to calculate the influence of adjacent clusters on nodes. They also do not attempt to visualize their graphs in 3D space. However, their method is a good and fast alternative algorithm for generating a 2D layout.

Animated Cone Trees by Robertson et al. [RMC91] provide a method to visualize hierarchical data in 3D space. The concept has been extended by Carriere and Kazman [CK95] to work on larger hierarchies. While providing a good representation of trees, these algorithms are not suitable for arbitrary graphs. The principle has been extended by Ho and Hong [HH06] who display a layout of arbitrary graphs in 3D space, whereas each cluster-layout is drawn in a plane. However, they also first reduce the graph to its maximum spanning tree and add the remaining edges after the layout process.

Lehmann and Kottler [LK07] present a method to visualize large clustered networks, yet their approach does not use fixed clusters but instead heuristically calculates a backbone of the network that implies a clustering.

The landscape metaphor has been first used by Chalmers [Cha93] to represent a body of documents and their relations. Over the years, several improvements have been made, such as the "Reshaped Landscape Metaphor" by Brandes and Willhalm [BW02]. A good overview by Dürsteler on projects that use the landscape metaphor for visualization of information can be found online [Dür05]. A very interesting approach that uses both the landscape metaphor and clustered data is described by Balzer et al. in their paper on software landscapes [BNDL04]. However, none of these approaches combine a graph-layout in a plane with a superimposed landscape, while still showing the original graph-structure including all nodes and edges.

Xu et al. [XCHT07] provide a such combination, by building a landscape on top of a layout in the ground plane. They use multivariate data and draw the original layout below a semi-transparent landscape. Their approach differs from ours in terms of the data to be visualized with the height field and the way the landscape is generated over the graph. They also focus on smaller graphs and do not use pre-clustered data.

Baur et al. developed a method to display the relations between autonomous systems on different levels in 3D space [BBGW05]. The resulting landscapes look similar to the ones generated by our method, but are generated in a different manner with focus on the coreness of the nodes.

2. Generating the 2D Layout

2.1. Motivation

Our landscape-visualization is based on a 2D layout of the graph in the ground plane. We use a multilevel, force-directed approach to generate the layout.

When calculating a layout for very large graphs, simple force-directed algorithms like the spring embedder by Peter Eades [Ead84] or the algorithm by Fruchterman and Reingold [FR91] are not suitable. They both require $\mathcal{O}(|V|^2 + |E|)$ time for each iteration, and to achieve pleasing results, the number of iterations should be in the order of the number of nodes in the graph. This leads to a running time in $\mathcal{O}(|V|^3)$ even for sparse graphs.

Today there exist a number of improvements to force-directed algorithms, including the introduction of data structures like quad-trees [BH86] and multi-level algorithms [Wal01]. These allow a relatively fast layout-process even for large graphs by aggregating forces. This reduces the running time to $\mathcal{O}(|V|^2 \cdot \log |V|)$.

However, if we already have a pre-computed cluster structure of the graph, we might not even want to layout the graph on a global scale, since this would possibly place nodes that are in the same cluster at different ends of the drawing. Additionally, we can use the cluster structure to reduce the complexity of the problem. Instead of calculating forces on a global level, we can break the problem down to smaller parts by separately calculating the layout of each cluster.

In this work, we use a hierarchical, cluster-based data structure to aggregate forces between nodes that lie in different clusters. We aggregate nodes within a cluster to a *cluster node* and add edges between these cluster nodes and from regular nodes to cluster nodes other than their own. This leads to a tree-like hierarchy with the original nodes of the graph as leaves and cluster nodes as their parents. This hierarchy is scalable by further clustering the cluster nodes and adding their clusters as grandparents, and so on. A detailed description of the hierarchy is given in the following section.

2.2. The Data Structure

2.2.1. The Cluster-Hierarchy

Our algorithm works on graphs with a given hierarchy of clusters. Figure 2.1 shows a simple example of a graph $G = (V, E)$ with two nested levels of clusters. The original

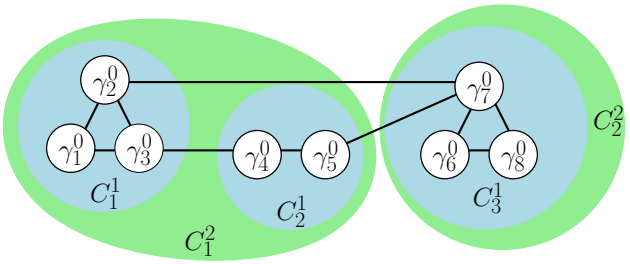


Figure 2.1.: A graph with level-0-nodes and two nested hierarchical clusterings

nodes of the graph are labeled $V = \{\gamma_1^0, \ldots, \gamma_n^0\}$. The first level of clusters (shaded in blue) consists of three clusters labeled C_i^1 for $i = 1, \ldots, 3$. The second level of clusters (shaded in green) is a more coarse-grained clustering with clusters labeled C_i^2 for $i = 1, 2$. The important concept of these hierarchies is, that nodes that are in the same cluster on a given level have to be in the same cluster on all higher levels in the hierarchy.

Formally, this leads to the following condition for clusterings:

Condition 1 (Hierarchy of clusters) *Let $l, k \in \mathbb{N}, k > l$ be two levels in the cluster hierarchy, γ_i^0, γ_j^0 level-0-nodes and C_a^l, C_b^k clusters on level l and k.*

$$\gamma_i^0, \gamma_j^0 \in C_a^l \wedge \gamma_i^0 \in C_b^k \Rightarrow \gamma_j^0 \in C_b^k$$

2.2.2. The Meta-Graph

With the graph and the given clusterings, we construct a meta-graph data structure. A meta-graph is implemented as a list of nodes per level. We refer to the original nodes of the graph as *level-0*-nodes. The nodes representing the clusters on the parent-level are called *level-1*-nodes, the grand-parents *level-2*-nodes, and so on. The *depth* of the meta-graph is the highest level of nodes in the graph and hence the depth of the hierarchical clusterings of the graph.

Figure 2.2 shows the meta-graph resulting from the input shown in Fig. 2.1. We will use this example in the following sections to explain how nodes and edges in the meta-graph are generated. Note that the clusters have now become nodes and are therefore renamed in a consistent way, i.e. cluster C_i^j has become node γ_i^j.

Note that the memory usage is limited to $\mathcal{O}(|E| \cdot d)$ with d being the depth of the clustering, since for every edge, at most $3 \cdot d$ new edges are created.

2.2.3. The Nodes

The node-class stores all information on the internal structure of the graph. It stores global and local neighbors of the node as well as an eventual subgraph, if the node is a cluster-node.

2.2.3.1. Global Neighbors

Each node has a list of incoming and a list of outgoing edges. These edges are *global* edges on the level of the node, i.e. in case of a level-0-node, the list contains all edges to other level-0-nodes without regard to the clusters. In case of a level-i-node, the list contains an edge to another level-i-node if there exists an edge between their respective children.

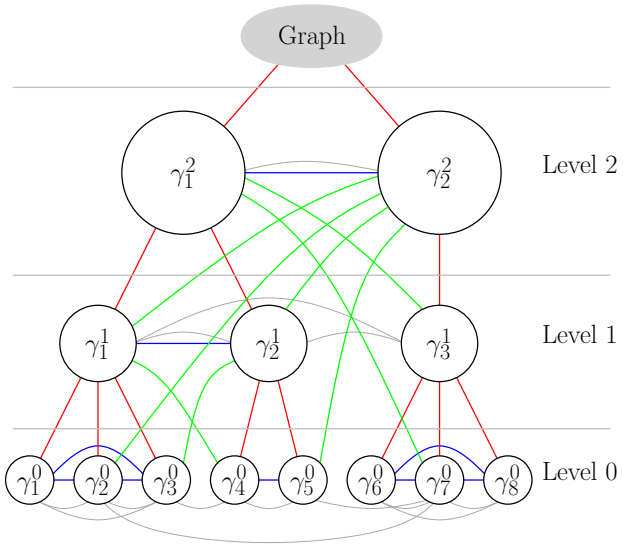


Figure 2.2.: The hierarchical cluster tree with parent-child-relations (red), sibling-sibling-relations (blue) and nephew-uncle-relations (green)

In Fig. 2.2, these edges are visualized in gray. In level 1, all nodes are connected, since for every pair of nodes there exist children that are interconnected. The same is true for the (two) level-2-nodes.

Note that the layout algorithm itself does not rely on these global edges. They are used to build up the data structure and could be deleted after the data structure has been set up. However, at least the global level-0-edges are necessary to draw the graph with all its original edges after the layout-process.

2.2.3.2. Local Neighbors

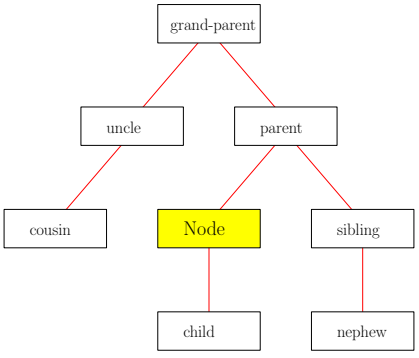


Figure 2.3.: The kinship nomenclature for the "relatives" of a node

Additionally to its global neighbors, a node has a list of incoming and outgoing edges for every level of the graph-hierarchy. To facilitate the nomenclature of relationships between nodes, we are, from now on, using notions of family kinship to refer to a node's "relatives". Fig. 2.3 provides an overview of the relations.

A level-i-node has edges to its *uncle*-nodes stored in its neighbor-list of level $(i+1)$. It stores edges to its *nephews* in his neighbor-list of level $(i-1)$. In his neighbor-list of level i it stores edges to its *siblings*, i.e. all nodes on the same level and with the same *parent*. These edges are a subset of the "global" edges.

The example in Fig. 2.2 shows edges between siblings in blue. Note that these edges are added if and only if the nodes have the same parent and if they are connected by a global edge (gray). In the special case of the highest level nodes (γ_1^2 and γ_2^2), the graph takes the role of the parent of the nodes, even though, strictly speaking, the graph itself is not a node.

Edges between uncles and nephews are shown green. These edges cross different levels to allow the accumulation of forces between a cluster and a single node in another cluster. This type of edge is added between a level-i-node A and a level-$(i+j)$-node B ($j \geq 1$) if A is connected through a global edge to one of B's descendants and if the ancestor of A in level $(i+j)$ is B's direct sibling.

For example, there is an edge between γ_2^0 and γ_2^2, because there is a global edge on level 0 (gray) between γ_2^0 and γ_2^7, which is one of γ_2^2's descendants and because there is a sibling-edge (blue) between γ_2^2 and γ_1^2, which is γ_2^0's ancestor on level 2.

Between γ_5^0 and γ_3^1 however, there exists no edge, even though γ_5^0 is connected to one of γ_3^1's children (γ_7^0). That is because there is no sibling-relation between γ_3^1 and γ_5^0's ancestor in level 1, γ_2^1. Yet, there is a sibling-relation between the ancestors of both nodes on level 2 (γ_1^2 and γ_2^2), which is why γ_5^0 and γ_2^2 (as well as γ_3^1 and γ_1^2) are connected.

To keep track of the inter-level-relations of the nodes, every node (except the ones on the highest level) has a pointer to its parent-node. Also, every node (except the ones on level 0) has a pointer to his *subgraph*. This is another meta-graph that contains all children, grandchildren and so on of the node. Its depth is thus by one less than the level of its containing node.

These parent-child relations are shown red in the example in Fig. 2.2.

2.2.4. The Edges

Each edge stores its weight and pointers to its adjacent nodes, i.e. source and target. Edges are stored as directed edges, even though in the algorithm they are treated as undirected edges. This makes it possible to avoid having two instances of the edge-class per existing edge.

Usually, the weight of an edge between two level-0-nodes is 1. The weight of an edge between two level-i-nodes A and B is the sum of the weight of all edges where the source node is a child of A and the target node is a child of B and vice versa. Since the weight of the edges between the children is again the sum of the weight of the edges between their children, weight is recursively "pushed up" to higher-level edges during the process of adding new clusterings. This results in the invariant, that the sum of the weight of all edges on a given level is the same for every level.

The weight of an edge between a level-i-node A and a level-$(i+k)$-node B is the sum of the weight of all edges between A and all children of B. This also includes the weight of the edges between A and potential grandchildren of B in case $k > 1$, since the edges to the children recursively "collect" the weight of lower-level edges.

2.3. Design of the Algorithm

For the positioning of nodes and clusters we use a multi-level force directed algorithm based on the approach by Fruchterman and Reingold [FR91]. While designing the algorithm, we kept in mind the representation as a landscape. We based our design on desiderata that optimize the layout of the clustered graph for the 3D representation as well as a 2D representation in the ground plane.

2.3.1. Desiderata

Our desiderata are the result of careful considerations and experiments with different layout approaches. Our primary goal was to preserve and show the structure of the cluster hierarchy in our layout, which requires **intra-cluster locality**. This means that we want to guarantee that nodes in the same cluster are placed close to each other in the final layout. We even guarantee that clusters are drawn without overlapping each other. This allows us to make the cluster structure visible by using textures showing the boundaries of clusters in the layout.

We further decided to lay out clusters in circular regions since this corresponds to the base shape of the Gaussian filter that we apply to the layout to calculate the elevation of the landscape. It also facilitates to determine whether two clusters are overlapping. Yet, we do not want to lay out every cluster for itself without regard to the connections to other clusters. We thus add an **inter-cluster attraction** force that takes into account the influence of adjacent nodes in other clusters on the local layout of a cluster. This pulls nodes towards the border of their cluster region if they are connected to nodes in other clusters. The number of edge-crossings within a cluster and thus the visual complexity is thereby reduced.

In order to avoid having many nodes placed in a small area, we require **Equal Node Density in Clusters**, guaranteeing that the layout area for clusters is proportional to the number of nodes within the cluster. Otherwise, clusters in the center of the layout would be very "compressed". Especially if we base the height of our landscape on the node density, this is an important factor. Our goal is to visualize the distribution of the node density within the cluster. Without requiring a layout area corresponding to the number of nodes in the cluster, the layout would produce very high peaks in the center of the graph caused by very dense clusters.

In the resulting 3D layout, it is possible to zoom into the landscape to examine a particular cluster. Yet we would like to avoid requiring the user to zoom in more than necessary. We thus want all clusters to make **Optimal Use of the Layout Area**. Our goal hereby is to use as much as possible of the available layout area for each cluster and avoid having strongly connected clusters use only a small part of the area. This helps to further improve clarity by drawing all parts of the graph as large as possible.

2.3.2. Our Approach

We extended the original Fruchterman-Reingold algorithm to meet our design goals. Yet we tried not to lose the original characteristics of the algorithm and not to slow down the process for clusters significantly.

2.3.2.1. Intra-Cluster Locality

The hierarchical approach of our algorithm guarantees that nodes in the same cluster are placed close to each other. In the first step, the highest level of the graph-layout is computed, in which each node represents a cluster. After the nodes have been placed, the algorithm is called recursively for their subgraphs, containing all nodes of the cluster. The subgraph gets laid out within a circular area around the position of its containing cluster node. This guarantees that in the final layout, nodes within clusters are drawn close to each other. Details on the implementation can be found in Section 2.4.1.

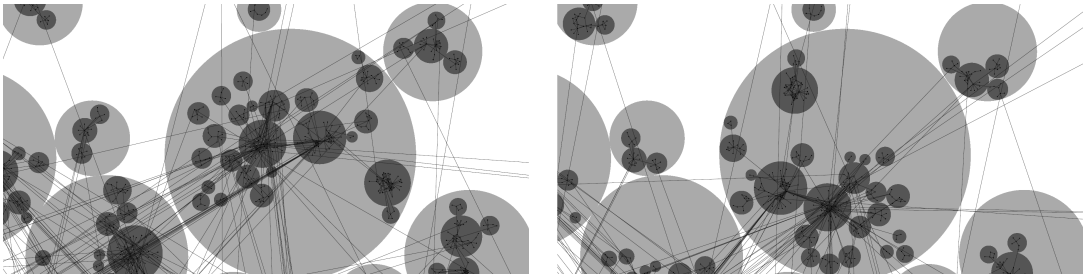
2.3.2.2. Inter-Cluster Attraction

To take into account the attracting forces of nodes in other clusters, we limit the local layout within the cluster to the first half of the iterations and then add attracting forces from uncle nodes to our displacement vector during the second half of the iterations.

However, we do not calculate the attracting forces in the same way we calculate them between nodes within a cluster, since the uncles are generally located at a large distance compared to the distances between inter-cluster nodes. Treating uncle edges like regular edges leads to nodes that are heavily pulled against the borders of the layout region and therefore cause quite unpleasant results.

Instead we limit attracting forces from uncles to the borders of the layout-region. We thus calculate the direction from the center of our layout region towards the center of the uncle node, normalize it and multiply it by 0.8 times the radius of our layout-region. This gives us a "virtual" uncle position that we use to calculate our attracting forces. The 80% factor helps to prevent nodes from being pulled too much towards the border of the layout region but still makes the influence of the uncle nodes visible. A more detailed description of this process is given in Section 2.4.2 and the pseudo code of the clamping function is shown in Figure 2.11.

Figure 2.4 shows the same layout with and without inter-cluster attraction. In Fig. 2.4(a), only a local layout has been computed, whereas Fig. 2.4(b) shows how nodes are pulled towards connected clusters on all levels.



(a) (b)

Figure 2.4.: A layout without and with inter-cluster attraction

A similar idea was used by Ho and Hong in their paper on drawing clustered graphs in 3D [HH06]. Their "Inter-Cluster Occlusion Minimization"-force pulls nodes towards adjacent nodes in other clusters. However, their approach differs from ours, since they use the full attracting force and establish a strong gravity force pulling nodes towards the center if they are pulled out of the layout region. The problem with this approach is, that nodes "stick" to the borders of the layout-region while their adjacent nodes within the cluster get pulled back towards the center. This results in a rather unpleasant looking layout.

2.3.2.3. Equal Node Density in Clusters

To assure a layout-area for a cluster that is proportional to the number of its nodes, we first take the weight of the cluster-node (i.e. the sizes of the clusters) and multiply it by the repulsive forces in our layout algorithm. This pushes large nodes further away from each other, but does not yet assure a large enough layout region. We therefore "blow up" the layout to assure that every node offers enough space for its subgraph to be laid out properly.

This blow-up process works as follows:

We first calculate the area that clusters are allowed to cover. We decided to heuristically choose one fourth of the drawing area, i.e. one fourth of the area of the parent cluster. The choice for one fourth is based on the fact that the worst case for a packing problem

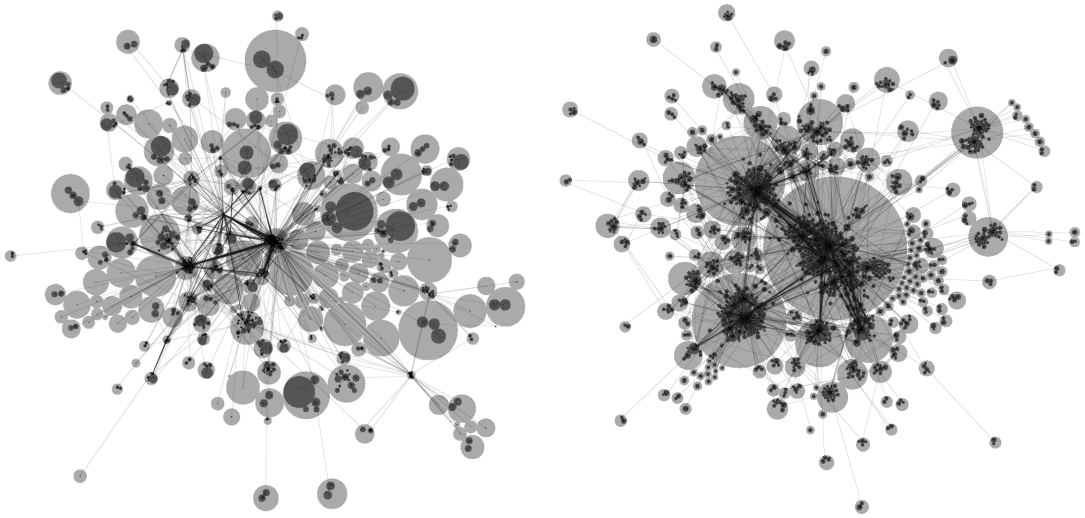
of smaller circles into a larger one is to have two circles with each half the radius of the larger one. This means, that we can allow at most half of the area of the large circle for the sum of the area of the smaller ones. To allow for more room for the smaller circles to resemble their original layout, we further reduce this area by one half.

We then distribute this area among the clusters proportionally to their weight (i.e. the number of nodes in the cluster) and calculate the radii corresponding to the size of the clusters.

Finally, we check for every pair of clusters whether they overlap, and if they do, we add a vector pointing away from the overlapping cluster to the repulsive force of the node. After limiting the repulsive forces to not push nodes too far during a single iteration, we apply them to the nodes. The pseudo code of the blow-up procedure is given in Section 2.4.3.

This assures that, at the end of the process, each cluster has its appropriate size. Our experiments showed, that this method preserves the initial layout of the clusters quite well. The relative position of cluster-nodes towards each other and the angle between them stays roughly the same since the forces that push cluster-nodes away from each other are based on their initial positions.

Figure 2.5 shows the same graph with and without equal node density in clusters. In Fig. 2.5(a) the areas per cluster have not been set before the blow-up-procedure. Instead, half of the distance to the next cluster has been set as cluster size, what explains the large areas at the outskirts of the graph. It is easy to see the large density of nodes in the center of the graph, which makes a distinction between single nodes or even large clusters almost impossible. Figure 2.5(b) shows the same input, but with cluster-areas proportional to the number of nodes they contain.



(a) (b)

Figure 2.5.: The same layout before and after blowing clusters up to their proportional area

2.3.2.4. Optimal Use of the Layout Area

Usually, when using force directed methods for drawing graphs, it is very hard to estimate a good “length” for the springs. If the graph is very sparse and has a large diameter, the spring length should be relatively small for the graph to fit into the layout area. For very

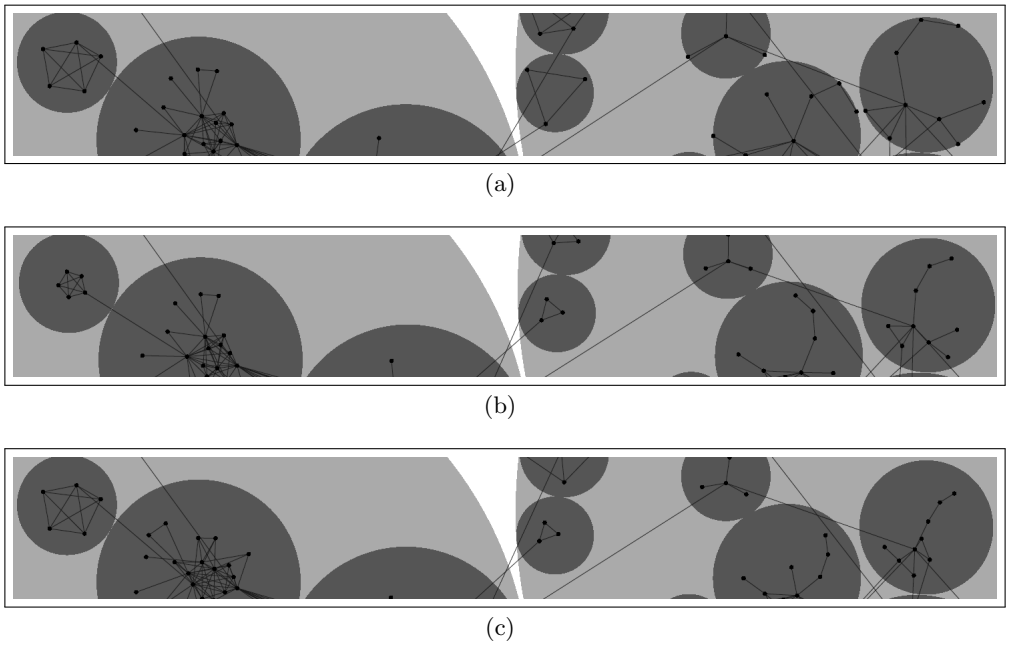
dense graphs however, the same spring length would cause the graph to get pulled very close together and a lot of the layout area would be wasted.

Our approach is to start with a very long spring length and dynamically reduce it if too many nodes get clamped at the border of the layout region. Our initial spring length is the radius of the circular layout region. If more than 10% of the nodes got clamped at the border of the layout region during an iteration, the spring length is reduced by one tenth.

Note that this process only gets applied during the first half of the iterations. During this period, only intra-cluster forces are used. This avoids nodes being clamped because they got pulled towards the border of the layout region by an uncle node.

We also added another type of forces to make sure that the layout is pulled towards the center of the layout area and does not "stick" to a border. This *center force* is calculated as follows: We square the distance of the node from the center of the layout-region, divide it by the desired radius of our layout region and use this term to scale the force that pulls the node towards the center of the layout region.

Figure 2.6 shows a comparison between fixed spring lengths and our adaptive spring length. In Fig. 2.6(a), the spring length was set to five times the radius of the cluster divided by the number of nodes. Note that the "clique" consisting of the K_5-graph in the leftmost cluster has a good size compared to the layout area, whereas in the case of the "chain" in the rightmost cluster, the spring length is too long. This causes many nodes to stick to the border of the cluster-region. Figure 2.6(b) shows the same layout with the spring length set to the radius divided by the square root of the number of nodes. This spring length is more suitable for the "chain" on the right but causes the "clique" on the left to shrink unnecessarily. The same clusters laid out using our approach with dynamically reduced spring lengths are shown in 2.6(c). Note that the spring length adapts to the internal graph-structure of the cluster and both the "clique" and the "chain" make good use of the given layout area.



(a) (b) (c)

Figure 2.6.: Comparison of different methods to calculate the spring length

The pseudo code in the following section provides further details on the calculation of spring lengths and forces.

2.4. The Algorithm

2.4.1. The Recursive Layout Procedure

The main layout-procedure for a graph (Fig. 2.7) first generates the layout-parameters based on the number of nodes to be laid out. If the graph is further clustered (i.e. its depth is greater than zero), the available area for the subgraph of each node is calculated and assigned. Then the layout of the graph is computed with the modified Fruchterman-Reingold algorithm (see Section 2.4.2) and the nodes are "blown up" to provide the area for their subgraphs. Finally the modified Fruchterman-Reingold procedure is called recursively for each subgraph. If the graph is not clustered, it just gets laid out with the modified Fruchterman-Reingold algorithm, which marks the end of the recursion. Section 2.3 explains the underlying design criteria in greater detail.

Function layoutFRmultiLevel(radius, center)

topNodes $\leftarrow$ list of nodes on the top level of the graph
springLength $\leftarrow \frac{\text{radius}}{|\text{topNodes}|}$
iterations $\leftarrow 10 \cdot |\text{topNodes}|$
temperature $\leftarrow \frac{\text{radius}}{4}$
if *depth > 0* **then**
    availableArea $\leftarrow \frac{\pi \cdot \text{radius}^2}{4}$
    foreach *node* $\in$ *topNodes* **do**
        subGraphArea $\leftarrow$ availableArea $\cdot \frac{\text{node.weight}}{\sum_{n \in \text{topNodes}} n.\text{weight}}$
        node.desiredRadius $\leftarrow \sqrt{\frac{\text{subGraphArea}}{\pi}}$
    layoutFR(radius, center, springLength, temperature, iterations)
    layoutFRblowUp(radius, center, 20)
    foreach *node* $\in$ *topNodes* **do**
        node.subGraph.layoutFRmultiLevel(node.desiredRadius, node.position)
else
    layoutFR(radius, center, springLength, temperature, iterations)

Figure 2.7.: The recursive layout procedure

2.4.2. The Modified Fruchterman-Reingold Algorithm

The modified Fruchterman-Reingold algorithm in Fig. 2.8 sets the position of the node to the center of the circular layout region, if there is only one node in the graph. Otherwise for the given number of iterations, the repulsive and attractive forces (*fRep* and *fAttr*) for each node are calculated as in the original paper by Fruchterman and Reingold [FR91]. Additionally, we calculate a force *fCenter* pulling nodes towards the center of the layout region. All forces are added up and applied to the node (See Fig. 2.10). A counter keeps track of the number of nodes that got clamped at the border of the circular layout region. During the first half of the iterations, the spring length gets reduced by 10% if more than 10% of the nodes got clamped (See Section 2.3.2.4 for more details). During the second half of the iterations, attraction by uncles of the node is added to the attractive force after being clamped at 80% of the radius (see also Section 2.3.2.2 and Fig. 2.11). The "temperature" of the system is linearly reduced, causing its movements to slow down and come to an end at the end of the iterations (simulated annealing).

Function layoutFR(radius, center, springLength, temperature, iterations)

coolingPerStep $\leftarrow \frac{\text{temperature}}{\text{iterations}}$
topNodes $\leftarrow$ list of nodes on the top level of the graph
if $|topNodes| = 1$ **then**
  topNodes[0].position $\leftarrow$ center
else
  for *iterations* **do**
    foreach $nodeU \in topNodes$ **do**
      fRep $\leftarrow 0$
      fAttr $\leftarrow 0$
      foreach $nodeV \in topNodes$ **do**
        dir $\leftarrow$ nodeU.position - nodeV.position
        if $\|dir\|^2 \neq 0$ **then**
          fRep $\leftarrow$ fRep + $\frac{\text{springLength}^2}{\|\text{dir}\|^2} \cdot \text{dir} \cdot \text{nodeU.weight} \cdot \text{nodeV.weight}$
    neighbors $\leftarrow$ all local neighbors of nodeU in its level
    foreach $nodeV \in neighbors$ **do**
      dir $\leftarrow$ nodeV.position - nodeU.position
      fAttr $\leftarrow$ fAttr + $\frac{\|\text{dir}\|}{\text{springLength}} \cdot \text{dir} \cdot \text{edgeweight(nodeU,nodeV)}$
    if *second half of iterations is reached* **then**
      foreach *ancestorLevel* **do**
        neighbors $\leftarrow$ all neighbors of nodeU in the ancestorLevel
        foreach $nodeV \in neighbors$ **do**
          dir $\leftarrow$ clampCircular(nodeV.position, center, radius) - nodeU.position
          fAttr $\leftarrow$ fAttr + $\frac{\|\text{dir}\|}{\text{springLength}} \cdot \text{dir} \cdot \text{edgeweight(nodeU,nodeV)}$
    dir $\leftarrow$ center - nodeU.position
    fCenter $\leftarrow \frac{\|\text{dir}\|}{\text{radius}} \cdot \text{dir}$
    disp $\leftarrow$ fRep + fAttr + fCenter
    nodeU.disp $\leftarrow$ min(disp.length, temperature) $\cdot$ normalize(disp)
    foreach $node \in topNodes$ **do**
      numClamped $\leftarrow$ numClamped + node.applyForces(center, radius - node.desiredRadius)
    if *in first half of iterations AND* $\frac{numClamped}{|topNodes|} > 0.1$ **then**
      springLength $\leftarrow$ springLength $\cdot$ 0.9
  temperature $\leftarrow$ temperature - coolingPerStep

Figure 2.8.: The modified Fruchterman-Reingold algorithm

```
Function layoutBlowUp(radius, center, forceReduction)
topNodes ← list of nodes on the top level of the graph
done ← FALSE
if |topNodes| > 1 then
    while NOT done do
        done ← TRUE
        foreach nodeU ∈ topNodes do
            fRep ← 0
            foreach nodeV ∈ topNodes do
                dir ← nodeU.position - nodeV.position
                if ‖dir‖² ≠ 0 then
                    desiredDistance ← nodeU.desiredRadius + nodeV.desiredRadius
                    overlap ← desiredDistance - dir.length
                    if overlap > 0 then
                        done ← FALSE
                        fRep ← fRep + normalize(dir) · (overlap + 0.1 · desiredDistance)
            nodeU.disp = fRep / forceReduction
        foreach node ∈ topNodes do
            node.applyForces(center, radius - node.desiredRadius)
```

Figure 2.9.: The blow-up function

2.4.3. The Blow-Up Procedure

The blow-up procedure (Fig. 2.9) iterates over all pairs of nodes and checks the distance between their centers. A check if the distance equals zero is made to avoid comparing nodes with themselves since all other nodes should have a distance greater than zero from each other after the layout process. The distance is then compared with the sum of the radii of both nodes. If they overlap, a vector pointing away from the overlapping cluster is added to the repulsive force vector of the node. The length of the added vector is the overlap plus 10% of the sum of the nodes' radii. The 10% are added to ensure a minimal length of the vector, since otherwise, slightly overlapping clusters would only have a very small force pushing them away from each other.

After the repulsive force vector of a cluster-node has been added up by forces from all overlapping nodes, it is divided by the force reduction parameter and set as the node's displacement vector. The force reduction is used to limit the movements of the system and to avoid nodes being pushed too far by strong forces.

Finally the forces are applied (See Fig. 2.10), whereas the radius given to the applyForces-function is reduced by the desired radius of the node. That way, the layout-area of a cluster-node is fully contained in the layout-area of its parent cluster.

Function int applyForces(center, radius)

```
position ← position + disp
dirToCenter ← center - position
if dirToCenter.length > radius then
    position ← center - normalize(dirToCenter) · radius
    return 1
else
    return 0
```

Figure 2.10.: The function to apply the forces to a node

Function point clampCircular(unclePosition, center, radius)

```
dirToUncle ← unclePosition - center
if dirToUncle.length ≤ radius then
    return unclePosition
else
    dirToUncle ← normalize(dirToUncle) · radius · 0.8
    return center + dirToUncle
```

Figure 2.11.: The function that clamps the forces for inter-cluster attraction

2.5. Results

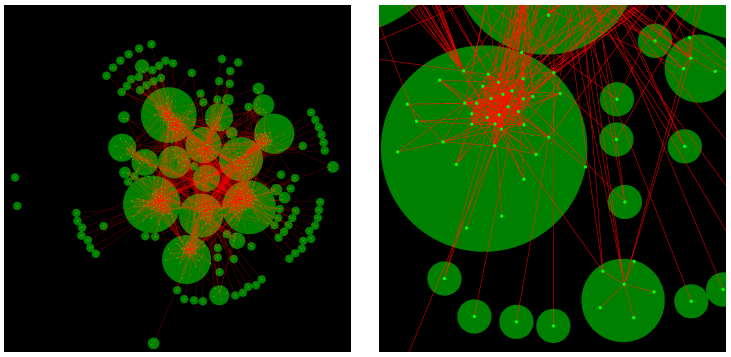
Since the blow-up procedure makes sure that no two clusters are overlapping, it is possible to draw the entire layout on a 2D plane. Our tool provides the functionality to generate a 2D visualization both as bitmap and as PDF-file. We further output an ASCII-file with the coordinates and radii for all clusters and nodes. Refer to Appendix A for details.

Below are the results of our algorithm for three example graphs at different zoom levels. The first graph (Fig. 2.12) is an email network of the computer science department at KIT provided by Robert Görke [Gör10] with 472 nodes and a cluster-depth of one. There are several clusters with single nodes, that are pushed towards the outskirts of the graph. There are also two unconnected nodes, that were pushed away from the main graph and were positioned at the left border of the layout region.

The second graph (Fig. 2.13) is the network of users of the PGP encryption algorithm. The underlying data was first used by Boguñá et al. [BnPSDGA04]. It consists of 10,680 nodes and is divided into two levels of clusters with 197 and 1209 cluster nodes. There are several very large clusters in the center and smaller ones in the outskirts. Note that the bigger a cluster is and the fewer connections it has to the rest of the graph, the further it gets pushed away from the main graph due to its weight.

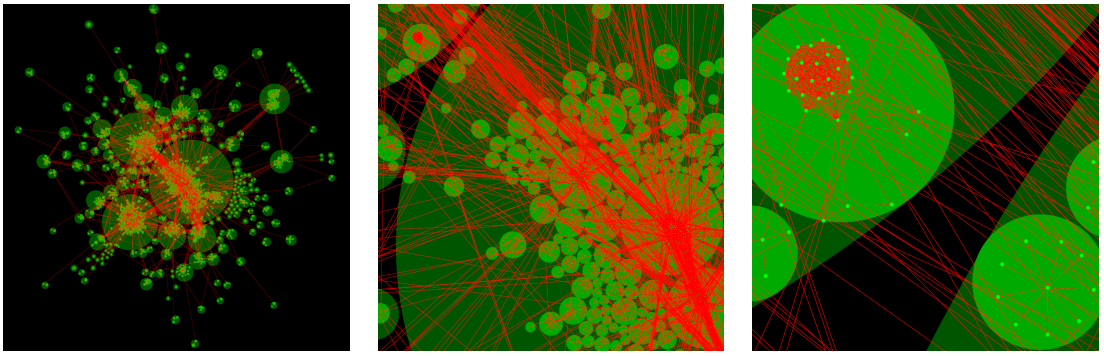
The third graph (Fig. 2.14) was generated by randomly placing 100,000 nodes in a square and adding edges between nodes that are closer to each other than a fixed threshold. The clustering was done by successively dividing the square in 25 equal smaller squares, each one representing a cluster. The nodes are divided in 3 levels of clustering with 25, 625 and 15,599 cluster nodes. Note that our algorithm was able to reconstruct the initial geometry of the graph starting from a random position of the nodes.

For more examples, please refer to Appendix C.



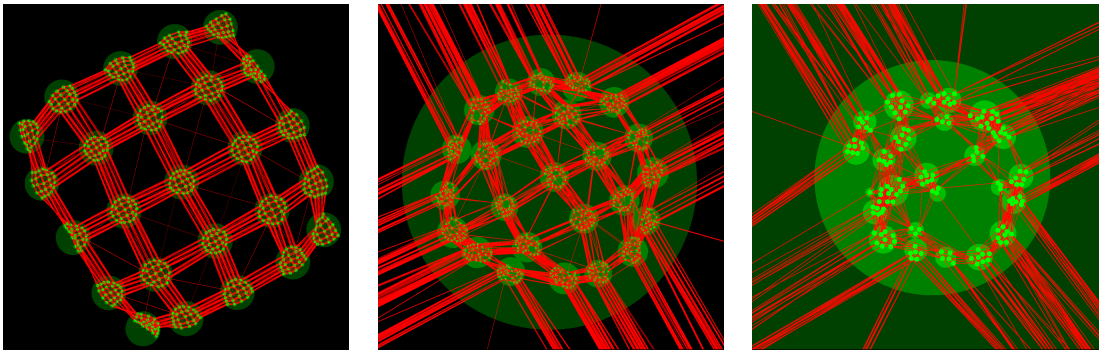
(a) (b)

Figure 2.12.: A graph with 472 nodes and a cluster hierarchy with one level, dividing the nodes into 112 clusters



(a) (b) (c)

Figure 2.13.: A graph with 10680 nodes and a cluster hierarchy with two levels, dividing the nodes into 197 level-2-clusters and 1209 level-1-clusters



(a) (b) (c)

Figure 2.14.: A generically generated graph with 100,000 nodes and a cluster hierarchy with three levels, dividing the nodes into 25 level-3-clusters, 625 level-2-clusters and 15,599 level-1-clusters

3. Visualization with the Landscape Metaphor

In the following sections we describe how we use the 2D layout from Chapter 2 to generate a 3D landscape. We first generate a mesh and determine the height of each vertex to develop a heightfield for the landscape. We then add the nodes and edges of the original graph and place them on the landscape. Finally we experiment with different textures to further emphasize the landscape or give additional information.

3.1. The Heightfield

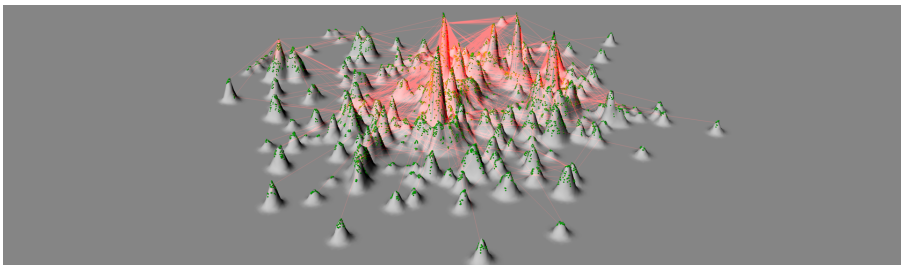
We generate a heightfield over the graph layout in the ground plane by applying a Gaussian filter over the properties of the graph we want to display as height-values. Figure 3.1 shows the pseudo code for the creation of a heightfield based on the node density. For every vertex of the mesh, we add up the values of all level-0-nodes stored in *nodesArray* multiplied by a Gaussian filter. That way, close nodes have a stronger impact on the height of a vertex.

Function heightFieldNodeDensity(mesh, nodesArray, sigma, scaleFactor)

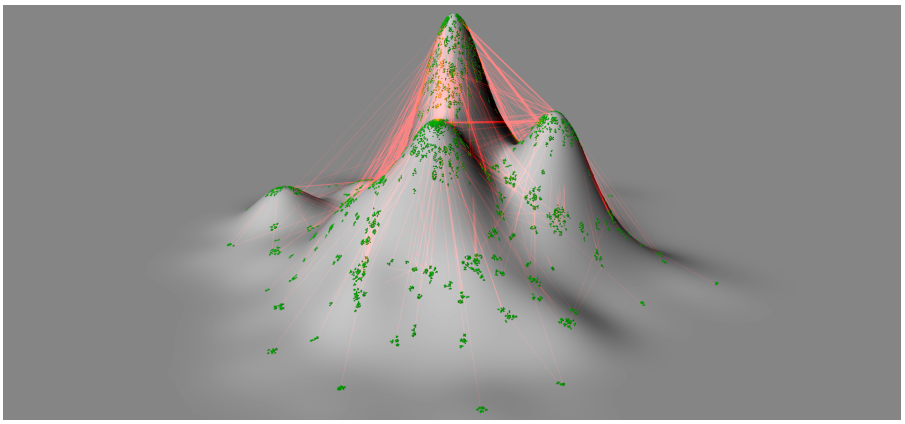
foreach *point* $P \in$ *mesh* **do**
    foreach *node* $N \in$ *nodesArray* **do**
        sqrDist $\leftarrow \|N - P\|^2$
        P.height = P.height + scaleFactor $\cdot$ N.value $\cdot \frac{1}{2\pi \cdot \text{sigma}^2} e^{-\frac{sqrDist}{2 \cdot \text{sigma}^2}}$

Figure 3.1.: The generation of the heigtfield based on node density

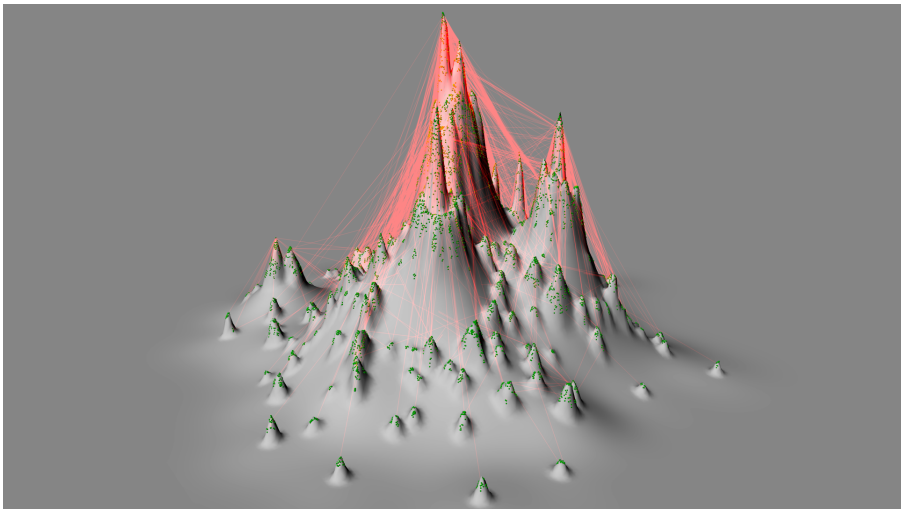
The sigma-parameter is used to set the variance σ for the Gaussian filter. This influences the distance in which nodes still have a noticeable influence on the height of the field. A small sigma means a more detailed landscape, that at the same time looks rather cluttered. Figure 3.2(a) shows the landscape of a graph with $\sigma = 0.1$. A large sigma smooths the landscape and forms nice hills instead of high peaks. This goes for the price of lost details. Figure 3.2(b) shows the same graph, this time with $\sigma = 1.0$. A combination of the two is also possible. We just apply the filter twice with different values for σ and add up the heights. This is shown in Fig. 3.2(c) and allows us to get an impression of both the overall structure of the landscape and the details of small clusters.



(a)



(b)



(c)

Figure 3.2.: The landscape with different σ-values for the Gaussian filter ($\sigma = 0.1$ for 3.2(a), $\sigma = 1.0$ for 3.2(b)) and a superimposition of the two heightfields

Function heightFieldNodeDegree(mesh, nodesArray, sigma, scaleFactor)

foreach *point* $P \in$ *mesh* **do**
  divisor = 0
  degFactor = 0
  foreach *node* $N \in$ *nodesArray* **do**
    sqrDist $\leftarrow \|N - P\|^2$
    divisor = divisor + $\frac{1}{2\pi \cdot \text{sigma}^2} e^{-\frac{sqrDist}{2 \cdot \text{sigma}^2}}$
    degFactor = degFactor + N.degree $\cdot \frac{1}{2\pi \cdot \text{sigma}^2} e^{-\frac{sqrDist}{2 \cdot \text{sigma}^2}}$
  if *divisor* > 0 **then**
    P.height = P.height + $\frac{\text{degFactor}}{\text{divisor}}$

Figure 3.3.: The generation of the heigtfield based on the average degrees of the nodes

Figure 3.3 shows the algorithm to generate a heightfield based on the average *degree* of nodes (that is the number of edges incident to a node). This time we calculate two Gaussians over the graph: one that represents the density and another one that is weighted by the degrees of the nodes. We divide the weighted Gaussian by the one representing the density and use the result as height information (after having checked for division by zero).

The result is shown in Fig. 3.4. As one would expect, the highest average node degree is in the center of the drawing. However, there are also noticeable elevations towards the borders. Compared to the landscape based on the node density of the same graph, the highest peaks have shifted to some smaller clusters that are strongly interconnected. Notice also the regions around the outer clusters. They resemble the outer region of a Voronoi diagram, which is because the height of every point on the mesh represents the average degree of its closest cluster.

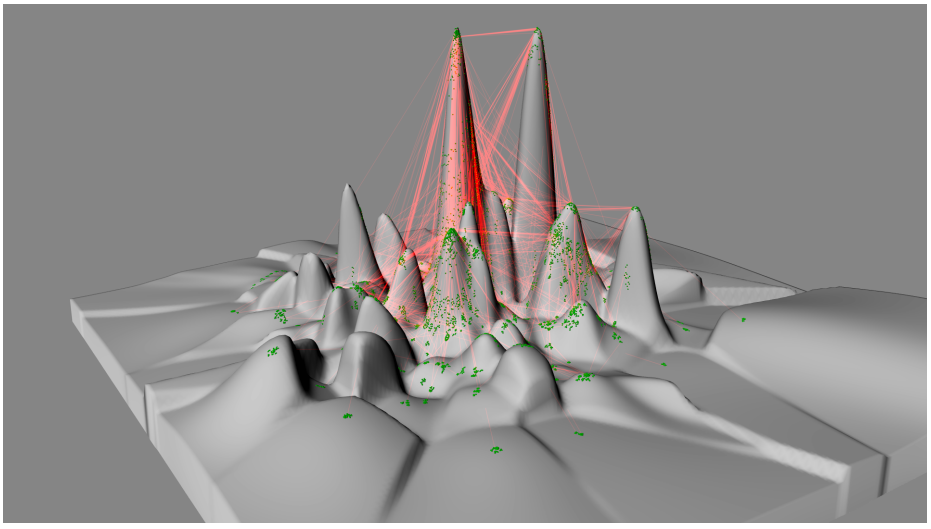


Figure 3.4.: The graph with the average nodes degree as coded in the height information

We also tried additionally raising the mesh by a fixed value at the locations of the cluster areas. This makes the underlying cluster structure visible (Fig. 3.5).

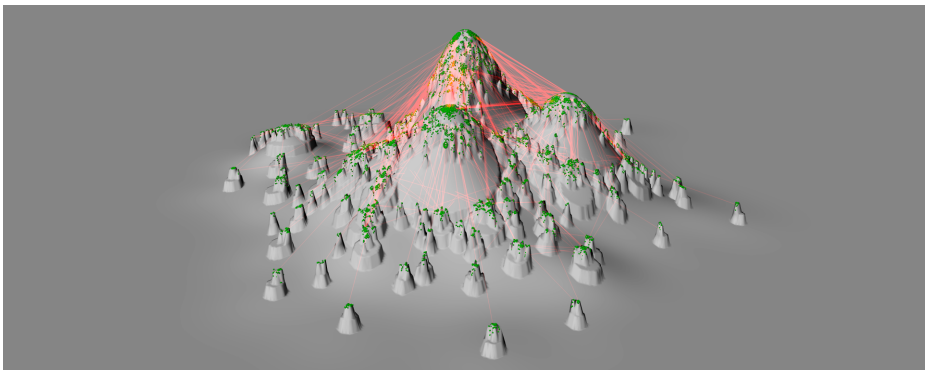
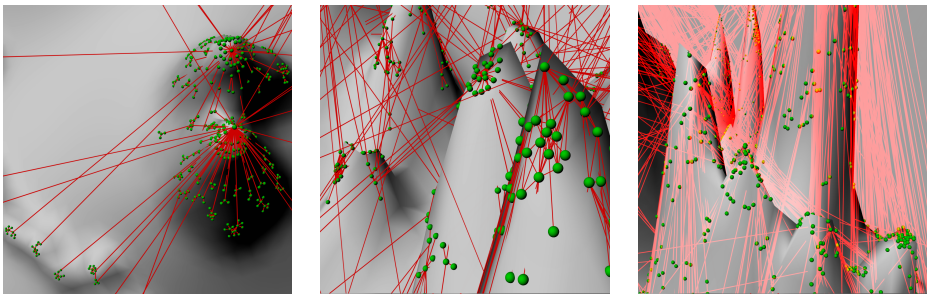


Figure 3.5.: The graph with $\sigma = 1.0$ combined with an elevation of the cluster structure

3.2. Nodes and Edges

We display nodes as spheres and edges as straight lines between nodes. We keep the coordinates of nodes in the ground plane and set their height-values according to the values of the heightfield at the nodes' positions. For edges, we use a technique called "Color Accumulation". This means that we use a rather dark color for the line of a single edge. If there is another edge in the background, the values of their colors get added and they therefore appear lighter. That way, large accumulations of edges glow brighter than single edges and therefore give a better idea of the structure of the graph. Figure 3.6 shows some details of the graph after zooming into particular regions.

Since we use straight lines as edges, we cannot guarantee that all edges will be visible from an outside view of the landscape. It is quite likely, that some edges intersect the landscape and are therefore only partially visible. Some of them might even lie completely below a "hill". However, most of the edges that "disappear" are edges within clusters. This has the positive effect, that the complexity of the drawing is automatically reduced without losing the information on the general structure of the graph that is mainly determined by inter-cluster edges. It is also possible to visualize the landscape from "below" and thereby examine the otherwise hidden edges (Fig. 3.7). Another option if it is desired to show all edges is to remove the geometry of the landscape. Just the location of nodes and edges in space already give an impression of the structure of the landscape (Fig. 3.8).



(a) (b) (c)

Figure 3.6.: Details of the landscape (3.6(a) and 3.6(b) use regular edge coloring, 3.6(c) uses Color Accumulation)

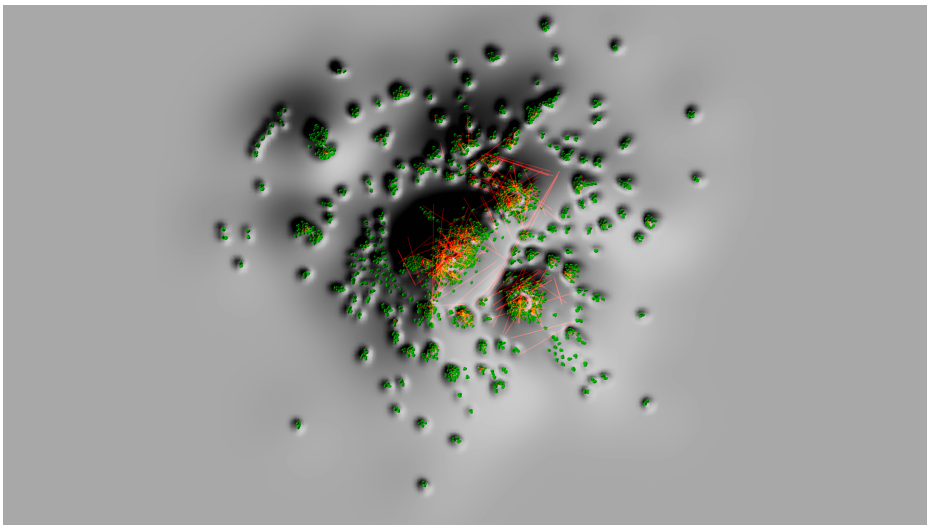


Figure 3.7.: The graph from Fig. 3.2(c) shown from “below”

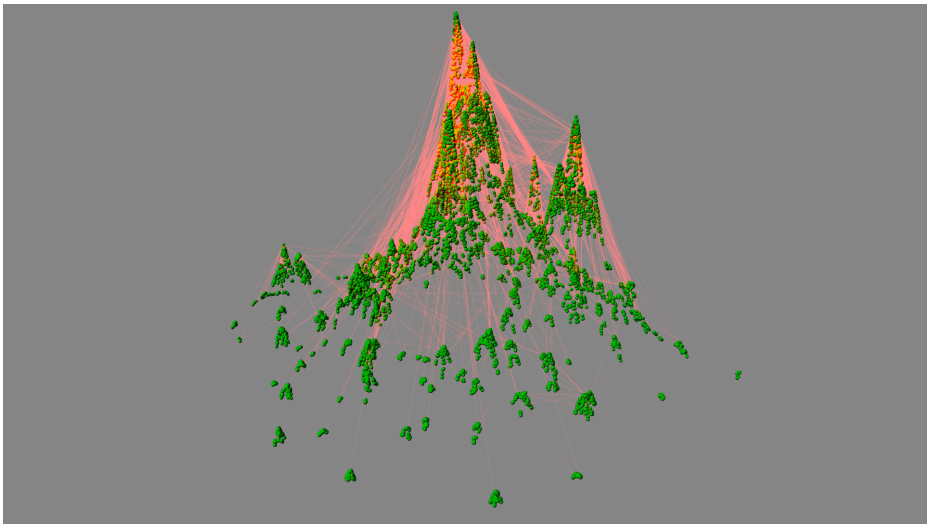


Figure 3.8.: A visualization without the underlying geometry

3.3. Textures

To get a general impression of the landscape, a simple gray Lambert shader like the one in Fig. 3.2 is sufficient. However, a well chosen texture can help to convey more information. An obvious choice would be a texture representing clusters and their boundaries. Figure 3.9 shows an example of such a texture.

When using a still image instead of an interactive graphic or animation, it can also be helpful to make the color of the surface dependent on the height. Figure 3.10 shows an example with different colors emphasizing the heights. Another example of such a texture which additionally makes the heightfield look more landscape-like is a ramp with colors representing different colored vegetation for different heights. The color theme of Figure 3.11 shows the graph as a tropical island. An advantage of this coloring is also, that even a look from the birds-eye perspective allows conclusions about the height of the landscape.

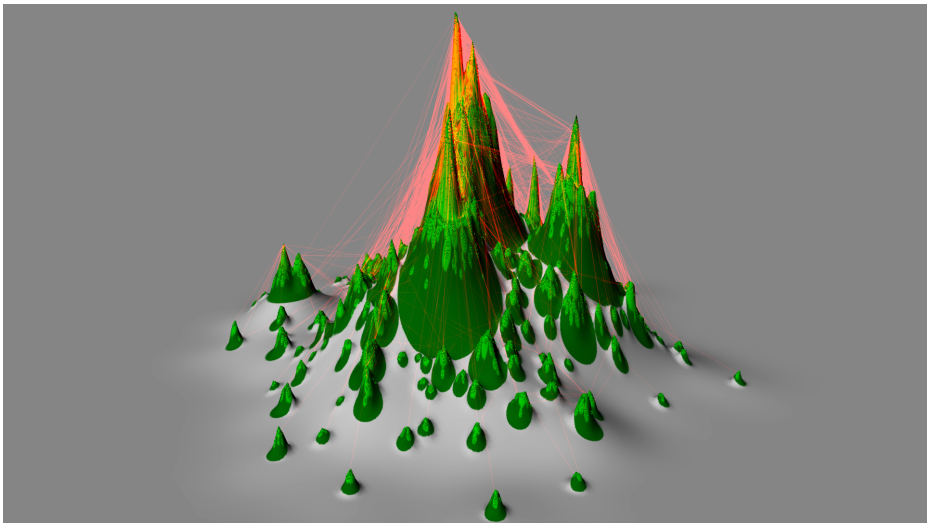


Figure 3.9.: A visualization with the cluster-structure as texture

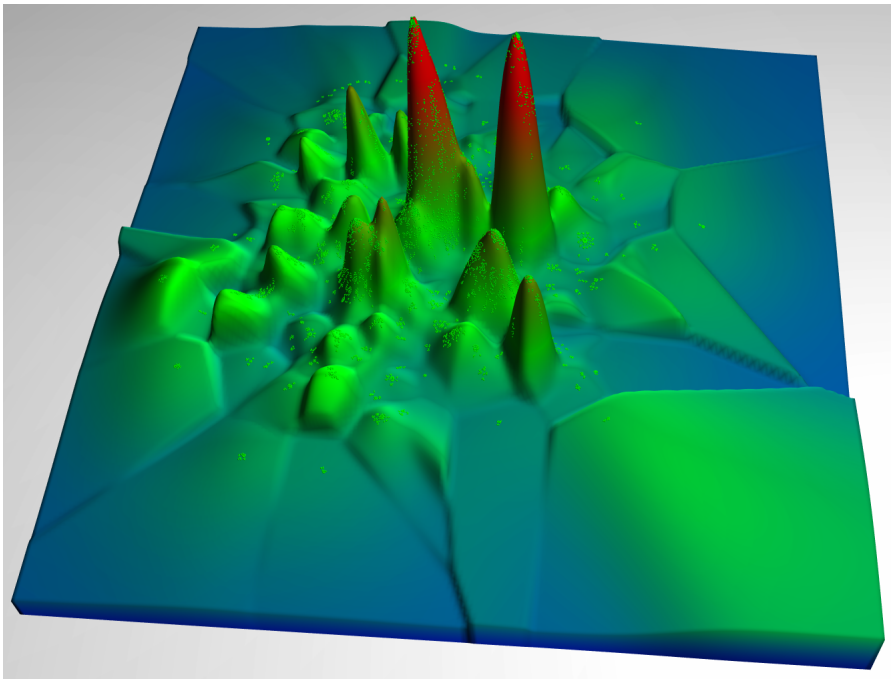
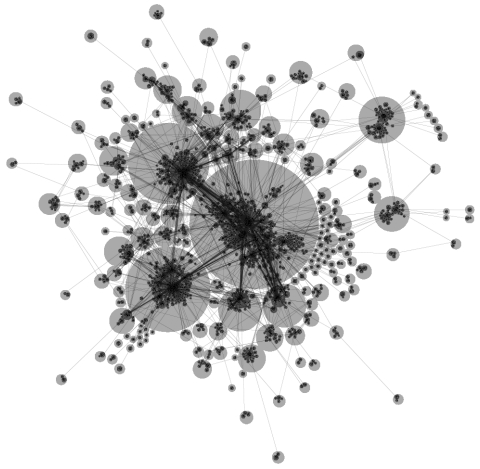
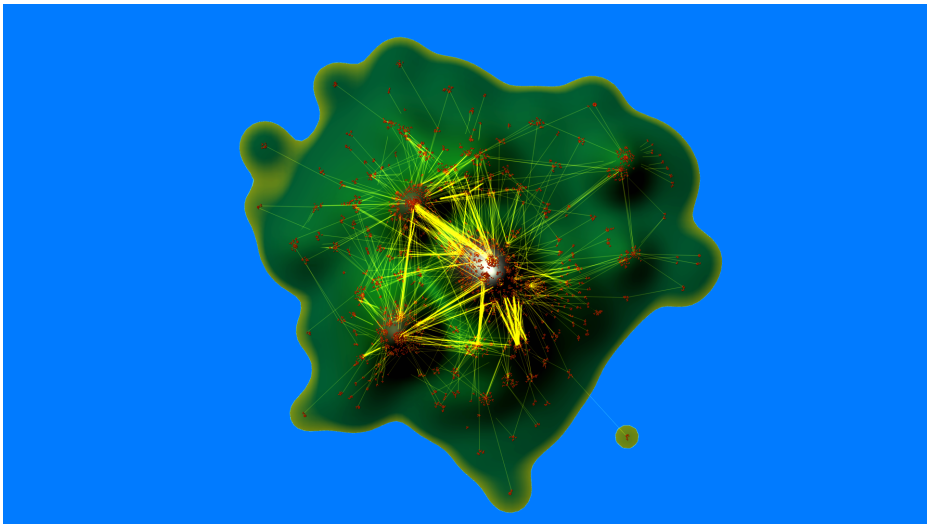


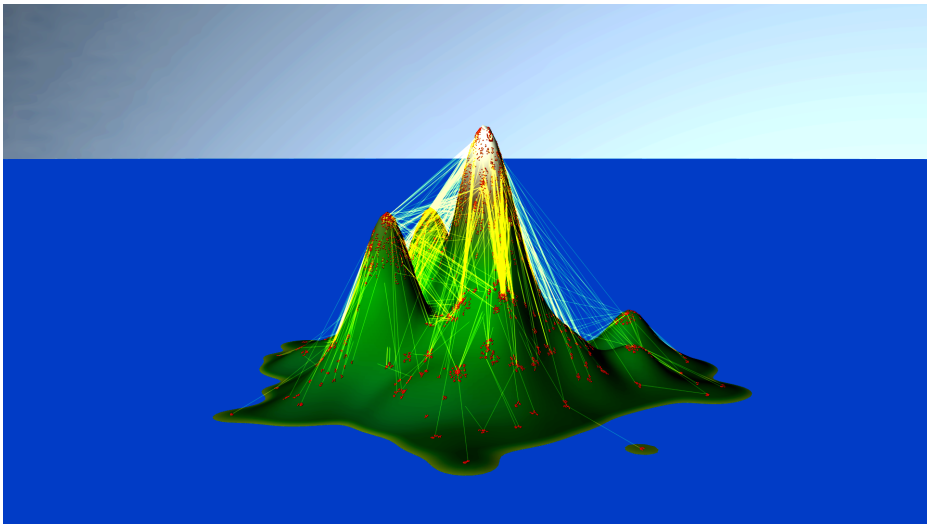
Figure 3.10.: The landscape from Fig. 3.4 with a ramp that emphasizes the elevations. For reasons of clarity, edges are suppressed in this rendering.



(a)



(b)



(c)

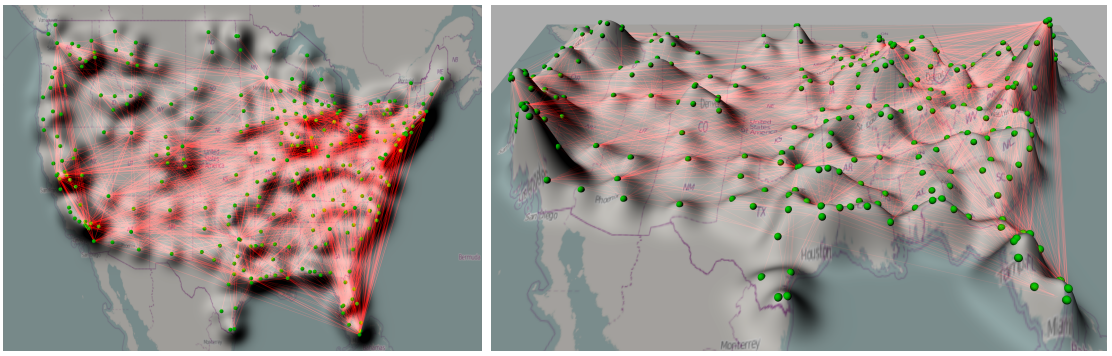
Figure 3.11.: The 2D layout, the resulting island from the birds-eye perspective and a side-view

3.4. Pre-Laid-Out Graphs

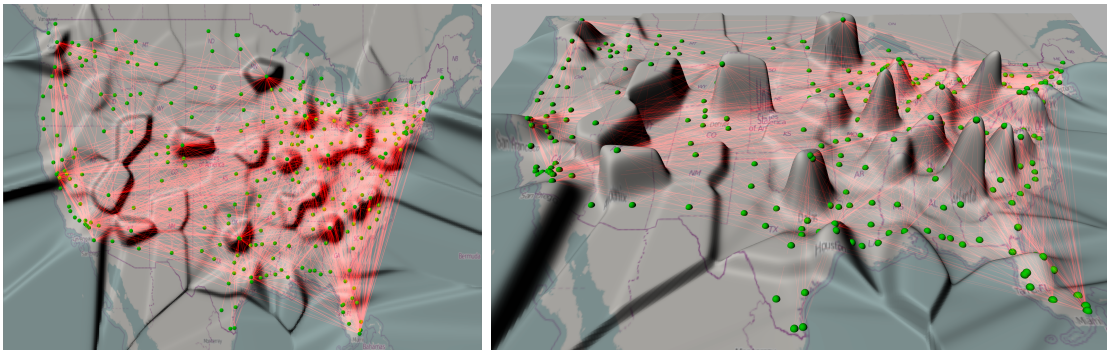
Another interesting application of our landscape metaphor is the visualization of given layouts. The most obvious choice for such a layout is geographical content. We used a dataset on US domestic flight connections from Batagelj and Mrvar [BM06] that we modified to only represent the 48 contiguous states. It shows all US airports and how they are linked by flight connections. We used a texture from "OpenStreetMap"[1] to show the correlation between airports and their geographical location.

Figure 3.12 shows two different heightfields of the graph, one based on node density and the other one based on node-degrees.

Especially the comparison between the two images provides some interesting insights. While the density of the nodes is very high in large metropolitan areas such as New York City and Los Angeles, the average node degree is rather small. This is caused by many small airports, that reduce the influence of the large airports of these areas. On the other hand, there are some areas with a rather small number of airports but a high degree of connectivity. Examples are major hubs like Atlanta, Mineapolis, Denver and Seattle.



(a)



(b)

Figure 3.12.: The US-airline graph with elevations based on airport density and node-degree (i.e. connections from and to an airport)

[1] © OpenStreetMap contributors, CC-BY-SA, `http://www.openstreetmap.org/`

4. Evaluation

4.1. Performance

In the performance analysis, we focus on the 2D layout algorithm. The performance of the 3D part mainly depends on the visualization software that is used and is less interesting from an algorithmic point of view.

For reasons of simplicity, we did not optimize our algorithm for speed. As mentioned earlier, there exist many improvements to force-directed algorithms. All improvements could also be applied to our modified Fruchterman-Reingold algorithm.

The running time of our method depends heavily on the cluster structure of our graph. If the cluster hierarchy is deep, all cluster-nodes have a similar, relatively small size and inter-cluster-edges are rare, the layout of large graphs is quite fast, even with our non-optimized method. If large clusters and many inter-cluster-edges exist in the graph, the advantage of our method compared to a non-hierarchical algorithm diminishes.

4.1.1. Theoretical Analysis

Let k be the maximum number of children of a node in the meta-graph and d be the depth of the meta-graph. Thus, k can serve as an upper bound for all cluster sizes. If the graph is *well balanced*, that is all clusters have size k, the cluster hierarchy corresponds to a k-ary tree with depth d. The modified Fruchterman-Reingold algorithm is thus called at most $1 + k + k^2 + k^3 + \ldots + k^{d+1}$ times (once for every node in the tree), which is in $\mathcal{O}(k^{d+1})$.

The modified FR-algorithm's running time for every cluster is in $\mathcal{O}(k \cdot (k^2 + |E_C|))$ where E_C includes inter- and intra-cluster edges of the cluster-node C. For dense graphs (i.e. the number of edges is in $\mathcal{O}(|V|^2)$), this leads to running times in $\mathcal{O}(k \cdot (k^2 + k \cdot |V|)) \subseteq \mathcal{O}(k^2 \cdot |V|)$ for every cluster. This results from the possibility that the k nodes in the cluster can be connected to all nodes outside of the cluster by $\mathcal{O}(k \cdot |V|)$ inter-cluster-edges.

Yet in most areas of interest, the graphs to be visualized are typically sparse (i.e. the number of edges is in $\mathcal{O}(|V|)$). All the example graphs we used in Section 4.1.2 fulfill this condition. Their ratio of edges to nodes is less than 10:1. For such graphs, we can calculate with an average running time per cluster in $\mathcal{O}(k^3)$, since on average, $|E_C| \subseteq \mathcal{O}(k)$ for each cluster.

The running time of the blow-up procedure is also in $\mathcal{O}(k^3)$: In every iteration, we iterate over all k^2 pairs of nodes. After at most 10 iterations in the worst case, one node has to be

"non overlapping", since nodes are pushed by at least one tenth of their desired distance away from each other. This leads to a running time of at most $10k \cdot k^2$ which is in $\mathcal{O}(k^3)$ until all nodes are non-overlapping.

Overall, the running time of our algorithm is thus in $\mathcal{O}(k^{d+1} \cdot k^3) = \mathcal{O}(k^{d+4})$ for sparse graphs. This sounds prohibitive at the first glance, but when the cluster hierarchy is well balanced or close to being well balanced, k is in the order of $\sqrt[d]{|V|}$, since the $|V|$ nodes are equally distributed among the k^d clusters on the lowest level of the k-ary cluster tree. This leads to a running time of $\mathcal{O}(\sqrt[d]{|V|}^{d+4}) = \mathcal{O}(|V| \cdot \sqrt[d]{|V|}^4)$. We thus have a cubic running time in hierarchies with two levels of clusters and quadratic running time in hierarchies with four levels of clusters. This requires a relatively balanced cluster structure though.

4.1.2. Experimental Results

In practice, as predicted in the theoretical analysis, the running time depends heavily on the structure of the graph. We tested our algorithm on several graphs with different cluster structures. All tests were conducted on an Intel® Core™2 Duo T9600 machine running at 2.8 GHz with 4 GB of RAM. Table 4.1 shows our results for a number of example graphs.

Graph	# Nodes	# Edges	Cluster Depth	Largest Cluster	Running Time
email	472	2845	1	112	0.5 s
hep-th	8,361	15,751	2	1,376	577.1 s
PGP	10,680	24,316	2	283	34.6 s
astro-ph	16,706	121,251	2	1,070	317.1 s
cond-mat	16,726	47,594	2	1,250	432.8 s
as-22july06	22,963	48,436	2	1,236	1,152.4 s
as-22july06	22,963	48,436	4	1,236	672.5 s
cond-mat-2005	40,421	175,691	2	1,871	3,646.1 s
Generic_10^5	100,000	880,222	3	25	10.7 s
Generic_10^6	1,000,000	7,846,050	3	98	1,298.8 s
Generic_10^6	1,000,000	7,846,050	4	36	152.6 s

Table 4.1.: The results of our tests with different example graphs

The graphs *email*, *PGP* and *Generic*_10^5 are the ones described in Section 2.5. *Generic*_10^6 has been generated in the same way as *Generic*_10^5. The other graphs have been used in the "10th DIMACS Implementation Challenge - Graph Partitioning and Graph Clustering"[1]. They have been clustered with an implementation of the Blondel clustering algorithm [BGLL08]. The graphs have been created by Mark Newman and mostly been used in his article on "The structure of scientific collaboration networks" [New01]. The graphs *cond-mat*, *hep-th* and *astro-ph* describe coauthorships between scientists in the areas of "Condensed Matter", "High Engery Theory" and "Astrophysics". *as-22july06* is a snapshot of autonomous systems in the internet.

The results of the tests confirm the general assumptions of the theoretical analysis. The comparison of the running times for *hep-th* and *PGP* show the strong influence of the size of the biggest cluster: Even though *hep-th* only has about 80% of the size of *PGP*, due to its 1376-nodes-cluster, the running time is about 15 times as long.

The generic graphs show the positive influence of a well balanced cluster structure on the running time. *Generic*_10^6 with 1,000,000 nodes and four levels of clustering is layouted about 3.5 times faster than *hep-th*, even though it has more than 100 times the number of nodes.

[1] `http://www.cc.gatech.edu/dimacs10/archive/clustering.shtml`

The comparison between the two clusterings of *Generic_*10^6 shows the effects of the depth of the clustering. The first clustering divided the 1,000,000 nodes into 25 level-3-clusters, 25^2 level-2-clusters and 25^3 level-1-clusters. The second clustering divided the nodes into 16 level-4-clusters, 16^2 level-3-clusters, 16^3 level-2-clusters and 16^4 level-1-clusters. The finer clustering reduced the running time to less than one eigth of the more coarse grained clustering.

This effect is also visible in the *as-22july06*-dataset. However, the numbers of cluster-nodes per level in the case of the 4-level-clustering are 36, 55, 265 and 2839, whereas the 2-level-clustering only uses the clusterings with 36 and 2839 cluster-nodes. This does not change the size of the largest cluster (1236), and therefore does not provide a better balanced tree. The gain in speed is thus not as significant as with the *Generic_*10^6 dataset.

4.2. Outlook and Open Questions

One important goal for future improvements would be to optimize the algorithm for speed. The introduction of simple acceleration techniques like geometric data structures could already result in major improvements. Also a more sophisticated schedule for the simulated annealing in the layout algorithm could provide better results with less iterations.

Another point would be the meta-graph data structure. Right now, it is not optimized in terms of memory usage. For example, every cluster stores a full list of its nodes. Also, it would be possible to get rid of the distinction between a node and its subgraph by establishing a structure of inheritance where a graph extends the node class.

In terms of the 2D layout, there would be room for further improvements regarding the shape of clusters. Currently, all clusters have a circular layout region. It would be interesting to experiment with layout regions that adapt to the shape of their internal layout. This would allow a better use of the available area. However, the shape of the clusters would no longer be corresponding to the shape of the Gaussian filter and the boundaries of clusters might not be as clear as with circular layout regions.

For the generation of the 3D layout, it might be interesting to experiment with more characteristics of the graph for the calculation of the height. Also a feedback loop that influences the 2D layout depending on the shape of the landscape would be an aspect for future work. In order to make results visually more appealing, it would be possible to include more elements of real landscapes. Nodes, for example, could be visualized as rocks or vegetation. Edges could be drawn in a parabola-like shape hanging between mountaintops and as arcs over mountains. That would also solve the problem of edges that are hidden below the surface in our current visualization.

An interesting topic for future research would be a study on the perception of graphs visualized with our method compared to simple 2D layouts. This would allow to quantify whether our method noticeably improves the understanding of the structure of large graphs. Fabrikant et al. have conducted a user study on the effectiveness of landscape metaphors in the visualization of a body of documents [FMM10]. They came to the conclusion that users generally did not understand the origins of the landscape and in what way the characteristics were influenced by the original data. However, their study was based on landscapes with no further information about the original data. Our approach is to display all the original data and add another dimension through the use of the landscape metaphor. We therefore would expect better results with our method.

5. Conclusion

Our method allows a visualization of large graphs by giving an overview of the general structure through the landscape representation while still allowing users to zoom in on low-level details of the original graph.

We approach the problem of overwhelming complexity in the visualization of large graphs with the principles of aggregation and abstraction: We use the given clustering to aggregate nodes to cluster-nodes, causing nodes within a cluster to be drawn close to each other. The 3D landscape we generate on top of the layout provides an abstraction from the original layout and allows displaying a feature of interest of the graph in the height dimension. The Gaussian filter adds another means of aggregation by averaging over the nodes in the area and thereby reducing noise caused by single extreme values.

Our 2D layout (Chapter 2) sets the basis for the landscape representation. We treat clusters on different levels the same way we treat nodes at the lowest level. This emphasizes the aggregation the clustering provides by accumulating nodes. By using circular layout regions, we adapt to the shape of the Gaussian filter that is going to generate the elevations of the heightfield. Since all clusters are non-overlapping, it is possible to output the layout as vector- or raster-graphic, which can already give a good impression of the graph's structure (Section 2.5).

The 3D landscape helps to display certain features of the graph and/or its 2D layout. We implemented methods to set the height based on node density and on the average node-degree, but other options are possible, too. The examples in Chapter 3 as well as in Appendix C show that the landscape in our visualization exposes features of the graph at the first glance that would otherwise require a time-consuming examination of each cluster. The experiments in Section 3.4 show that the representation as a landscape does not only make sense for clustered graphs with a layout generated by our algorithm, but also for given layouts of graphs.

Our 3D visualization still includes all nodes and edges of the original graph. Even though edges might not all lie above the surface, it is possible to examine all edges by hiding the geometry of the landscape or by choosing a view from below the surface. The edges that intersect the surface or that lie completely below the landscape are mostly intra-cluster-edges. Thus the fact that they are hidden when viewed from above leads the attention to the remaining edges that are more important to understand the relationships between clusters.

6. Appendix

A. File Formats

A.1. The Input

Our tool supports two different input formats for Graphs: GraphML and an ASCII-based format. The GraphML format corresponds to the output of the "'yEd Graph Editor"' [1]. The ASCII format consists of floating point numbers and integers written in a .txt-file with the following syntax: The first entry is an integer and indicates the number of nodes in the graph. The next $|V|$ pairs of floats give the initial x- and y-coordinates for each node. The following $|E|$ pairs of integers indicate the indices of nodes that are connected by an edge.

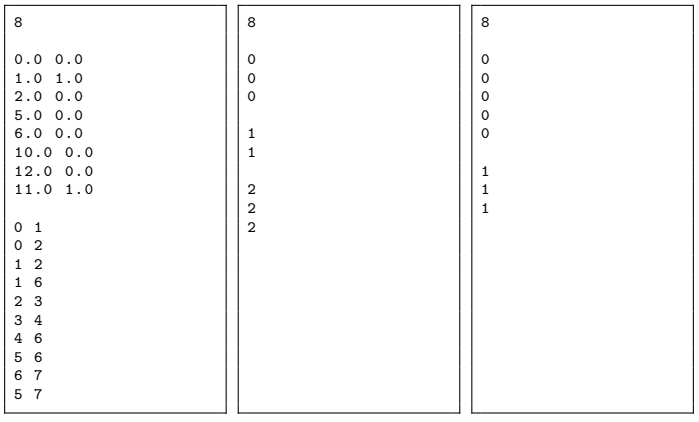


Figure A.1.: The ASCII-representation and two levels of clustering of the example graph

The information about clusters is provided by an ASCII-file for each cluster level. The syntax is the following: The first integer is the number of nodes in the graph. The following $|V|$ entries indicate the cluster-ID for each of the nodes. Nodes with the same ID are in the same cluster. Note that these IDs do not necessary need to be consecutive. Figure A.1

[1] `http://www.yworks.com/en/products_yed_about.html`

shows the ASCII-input as well as the first and second level of clusterings for the example graph in Fig. 2.1.

A.2. The Output

We have developed an ASCII-convention to store the output of the layout-process. This file contains information about the positions of nodes and clusters as well as the radii of the clusters. We use the filename extension *.cgl*, standing for *Clustered Graph Layout*.

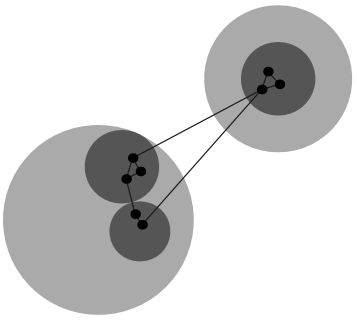
The first entry in the file is an integer indicating the depth of the graph. This is followed by an integer for every level of the graph, starting with level 0, that gives the number of nodes in that level.

The following part of the file contains geometric information about the graph. The nodes and clusters are stored as float-triples. The list starts with the level-0-nodes, followed by level-1-nodes, and so on. The first two floats of a node indicate its x- and y-coordinates. For level-0-nodes, the third value is reserved for the weight of the node with a default value of 1.0. The first two floats of a cluster indicate the position of its center, the third float its radius. Since only the relative positions of the layout are relevant, all coordinates are normalized, such that all coordinates are in the range from 0.0 to 1.0.

The file ends with a list of all edges, represented as pairs of integers containing the ID of source- and target-nodes. Figure A.2 shows a *.cgl*-file describing a layout of our example graph as well as the corresponding graphical representation.

```
2

8 3 2

0.397330700665186 0.530670314509916 1.000000000000000
0.381230266121971 0.502830087646693 1.000000000000000
0.367494842525572 0.546413027416599 1.000000000000000
0.386399762148849 0.619633918064828 1.000000000000000
0.400648286751533 0.641728406393218 1.000000000000000
0.662732885276458 0.322746826112462 1.000000000000000
0.649808638615143 0.359976429675922 1.000000000000000
0.686718368255583 0.349209463680375 1.000000000000000

0.357728834623272 0.520888449243106 0.076546554461974
0.395135810386348 0.655498597270771 0.062500000000000
0.683055552083802 0.337650516188971 0.076546554461974

0.308683978594845 0.631607893085064 0.197642353760524
0.683055552083802 0.337650516188971 0.153093108923949

0 1
0 2
1 2
1 6
2 3
3 4
4 6
5 6
6 7
5 7
```

(a)



(b)

Figure A.2.: The output of our example graph as “Clustered Graph Layout” (.cgl) file and the corresponding graphical output

B. The Maya-Plugin

For the landscape-visualization of the graph, we decided to use the 3D animation software Autodesk® Maya®[2]. This software is the industry standard for 3D modeling and animation and offers a large toolbox for modifying meshes and textures. It also provides a very powerful API for writing C++ plugins that integrate seamlessly into the main program.

Our *graphLandscapeCreator* plugin registers the following nodes and commands:

- **GraphLandscapeCreator** (deformer node)

  This is a deformer node that is applied to a selected mesh by the command `deformer -type graphLandscape`. Its properties allow the selection of a .cgl-file, the modification of sigma and the scaleFactor for elevations based on the density of nodes, the setting of a value for a cluster elevation and the modification of the sigma for elevations based on the average node degree.

- **displayNodes** (command)

  This command generates nodes at their appropriate positions on a selected mesh. It also generates a *nodeShader* (in a standard green lambert tone) and assigns it to the nodes.

- **displayEdges** (command)

  This command generates the edges between the nodes and assigns them a standard *edgeShader* in a dark red. Note that the `displayNodes` command has to be called first in order to have the node's positions for the edges to connect to.

- **Graph Texture** (2D texture node)

  This node is a texture with the clusters based on the input of the selected .cgl-file. It allows to set background and cluster colors.

Our Maya plugin uses Particles to display nodes and edges. This has practical reasons, since Maya has problems handeling several tens of thousands of items in its DAG, even if they are all just instances. Particles however are just stored as points and computed very efficiently. Nodes particles are set to the rendertype *Spheres* whereas edges are *Streaks* with the velocity vector pointing in the direction of the incoming node. The drawback for particles is, that they are only visible when the render settings are set to *Maya Hardware*. If software rendering is crucial for the landscape, putting nodes and landscape in different renderlayers would be a workaround.

[2] A free student version of this software is available for download at `http://usa.autodesk.com/maya/trial/`

C. More Results

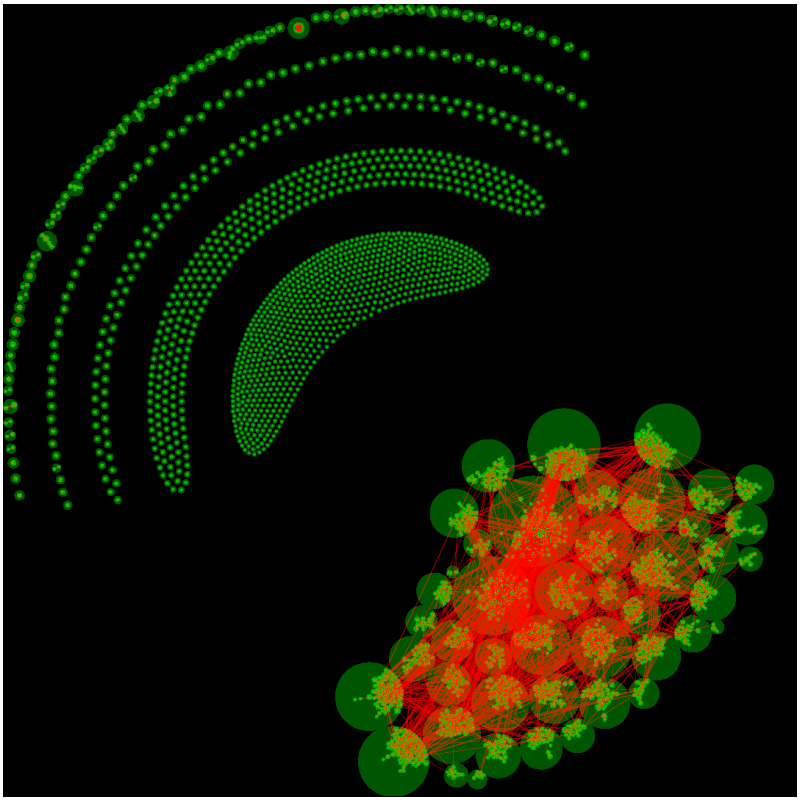
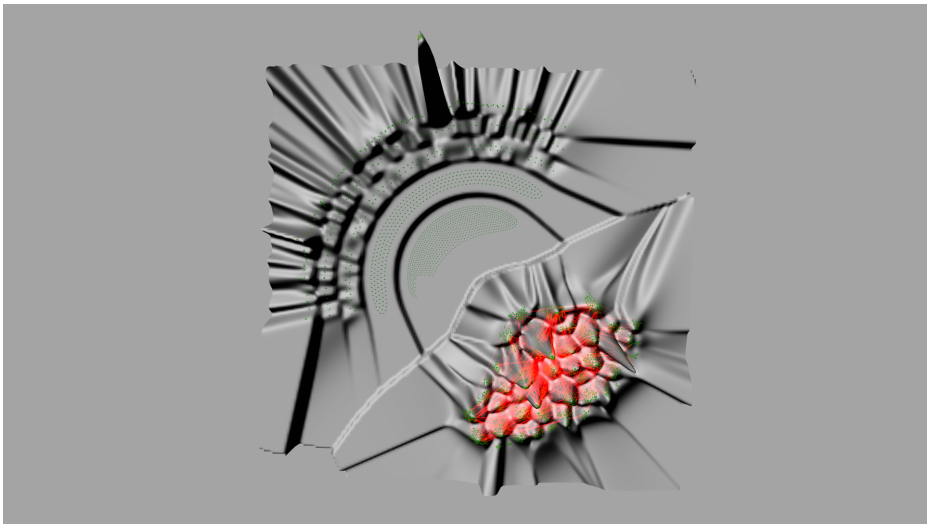
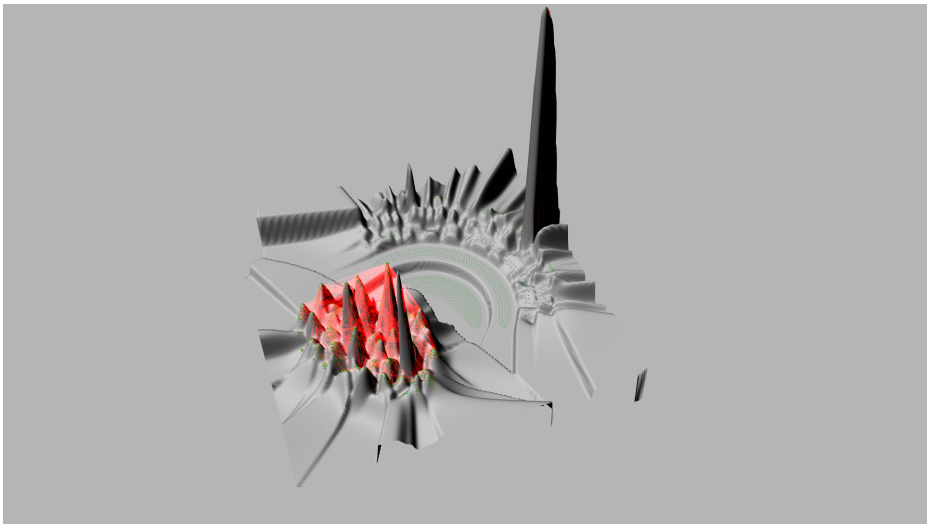


Figure C.3.: The graph *hep-th* with 2 levels of clustering. Note the division between the main part of the graph and unconnected clusters that are pushed away from the main graph. The repulsive force is stronger if the weight of a cluster is higher, which is why the "free" clusters are ordered according to their weight.



(a)



(b)

Figure C.4.: The 3D representation of *hep-th* with 2 levels of clustering and the height based on the average node degree. Note how the average node degree relates to the weight of the unconnected clusters. It is also interesting to examine the high peak at one of the unconnected clusters. It is caused by a clique of 24 highly connected nodes in the top left part of the original layout.

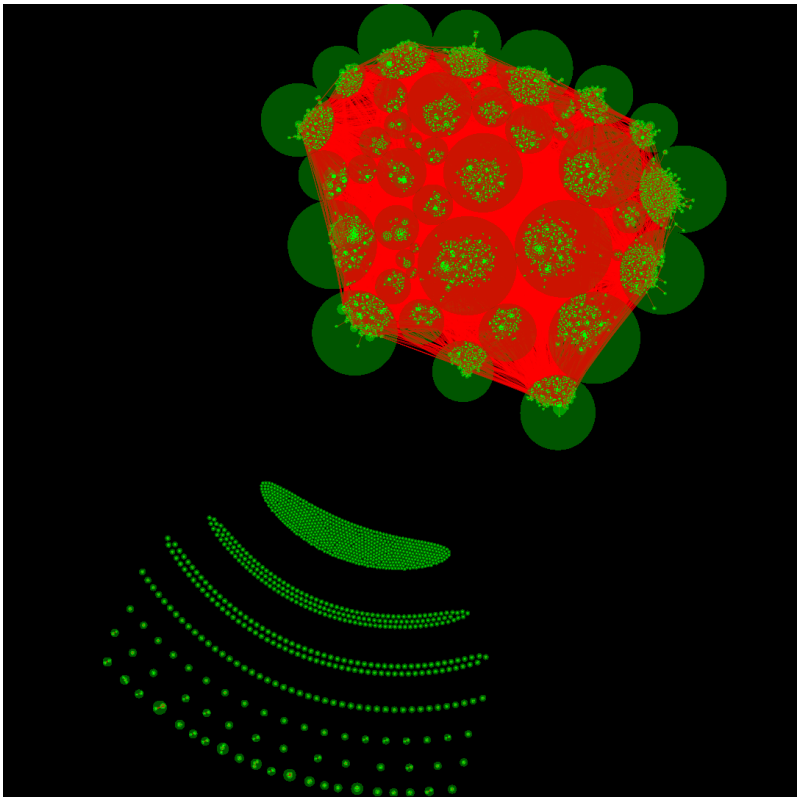


Figure C.5.: The graph *astro-ph* with 2 levels of clustering. This graph is very dense with a lot of inter-cluster-edges. Even though single edges are almost impossible to destinguish, the inter-cluster attraction allows to see a tendency which nodes have strong connections to adjacent clusters.

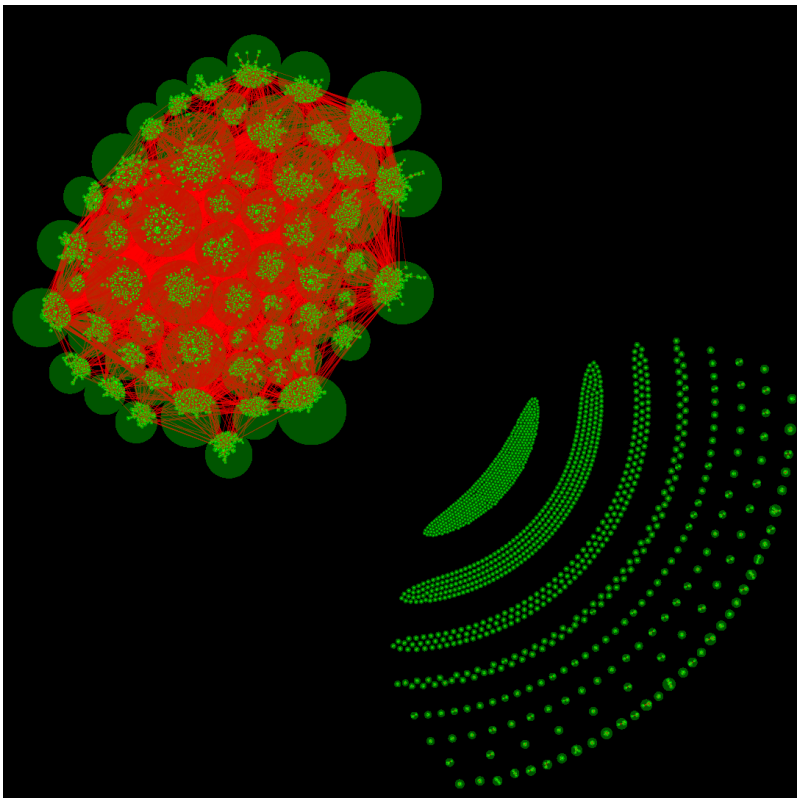


Figure C.6.: The graph *cond-mat* with 2 levels of clustering

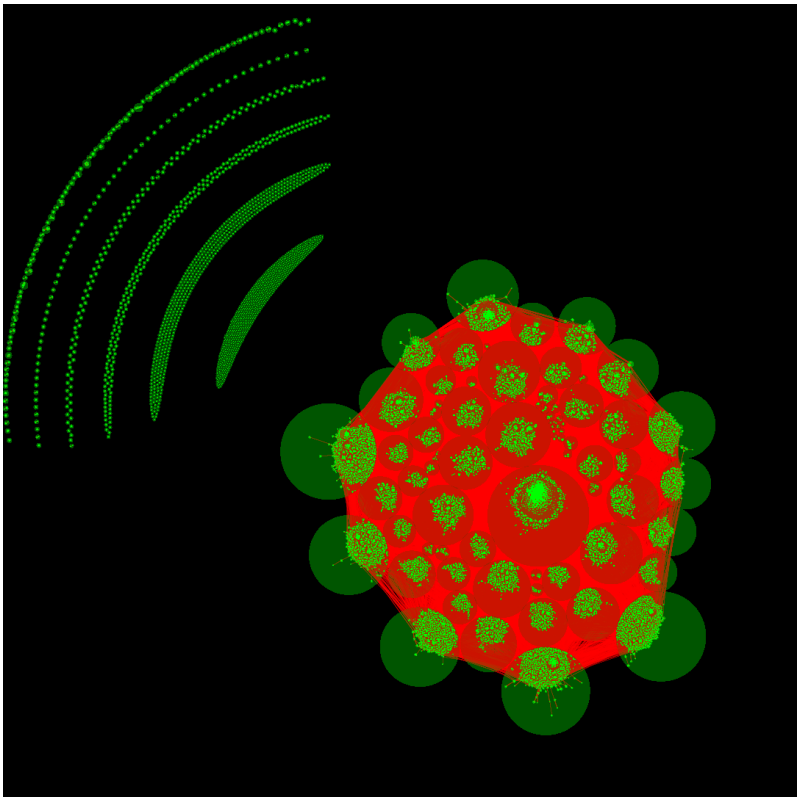


Figure C.7.: The graph *cond-mat-2005* with 2 levels of clustering

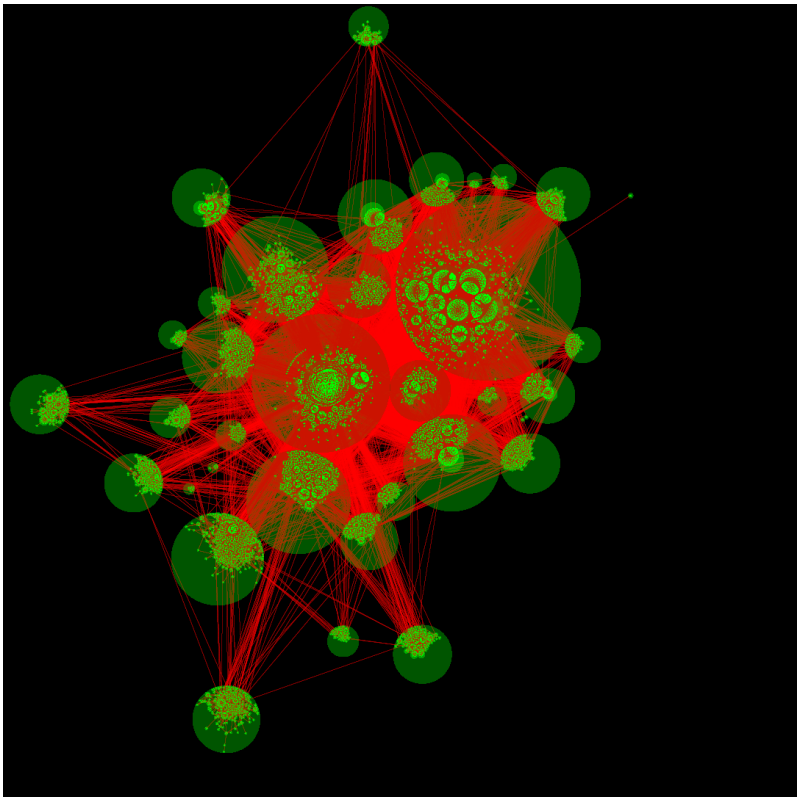
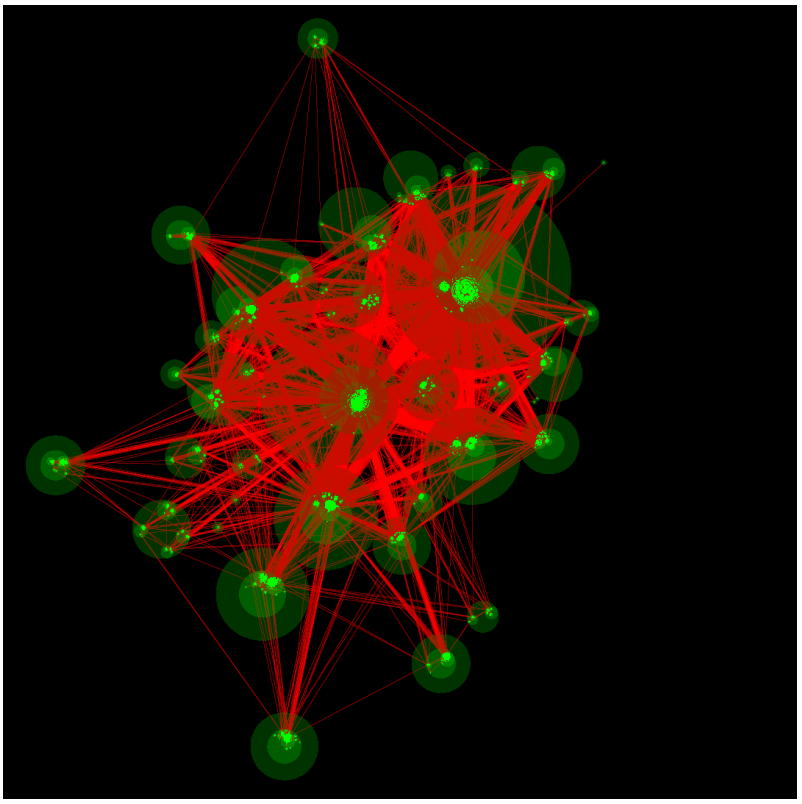
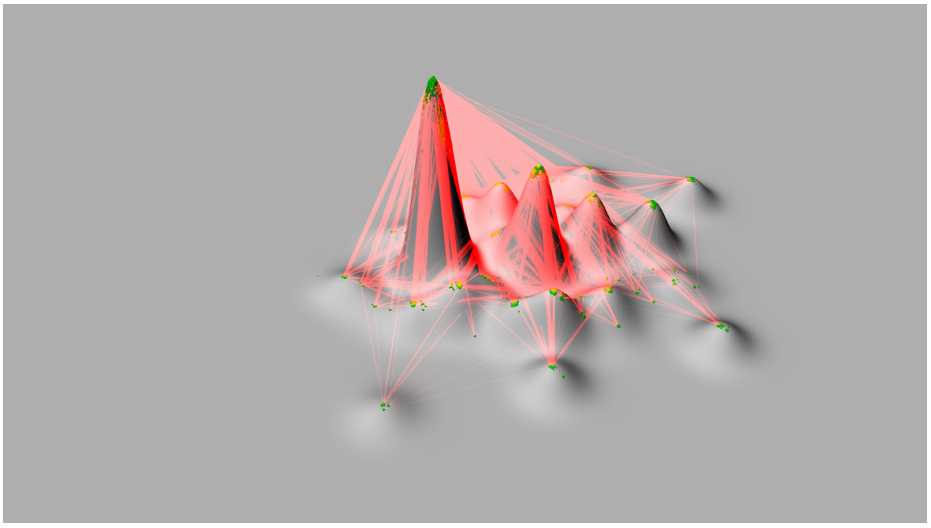


Figure C.8.: The graph *as-22july06* with 2 levels of clustering



(a)



(b)

Figure C.9.: The graph *as-22july06* with 4 levels of clustering and the corresponding landscape representation based on the node density. Note that the node density is very high due to the reduced layout area per cluster compared to the 2-level-layout.

Bibliography

[BBGW05] Michael Baur, Ulrik Brandes, Marco Gaertler, and Dorothea Wagner: *Drawing the as graph in 2.5 dimensions.* In János Pach (editor): *Graph Drawing*, volume 3383 of *Lecture Notes in Computer Science*, pages 43–48. Springer Berlin / Heidelberg, 2005, ISBN 978-3-540-24528-5. http://dx.doi.org/10.1007/978-3-540-31843-9_6.

[BGLL08] Vincent D. Blondel, Jean Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre: *Fast unfolding of communities in large networks.* Journal of Statistical Mechanics: Theory and Experiment, 2008(10):P10008+, July 2008, ISSN 1742-5468. http://dx.doi.org/10.1088/1742-5468/2008/10/P10008.

[BH86] Josh Barnes and Piet Hut: *A hierarchical O(N log N) force-calculation algorithm.* Nature, 324(6096):446–449, December 1986. http://dx.doi.org/10.1038/324446a0.

[BM06] Vladimir Batagelj and Andrej Mrvar: *Pajek datasets*, 2006. URL: http://vlado.fmf.uni-lj.si/pub/networks/data/, [Online; accessed 27-November-2011].

[BNDL04] Michael Balzer, Andreas Noack, Oliver Deussen, and Claus Lewerentz: *Software Landscapes: Visualizing the Structure of Large Software Systems.* In Oliver Deussen, C. Hansen, D. A. Keim, and D. Saupe (editors): *Proceedings of the Joint Eurographics - IEEE TCVG Symposium on Visualization*, pages 261–266, 2004. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.9.8069.

[BnPSDGA04] Marián Boguñá, Romualdo Pastor-Satorras, Albert Díaz-Guilera, and Alex Arenas: *Models of social networks based on social distance attachment.* Phys. Rev. E, 70:056122, Nov 2004. http://link.aps.org/doi/10.1103/PhysRevE.70.056122.

[BW02] Ulrik Brandes and Thomas Willhalm: *Visualization of Bibliographic Networks with a Reshaped Landscape Metaphor.* In *Proc. 4th Joint Eurographics - IEEE TVCG Symp. Visualization (VisSym '02)*, pages 159–164. ACM Press, 2002. http://www.inf.uni-konstanz.de/algo/publications/bw-vbnrl-02.pdf.

[Cha93] Matthew Chalmers: *Using a Landscape Metaphor to Represent a Corpus of Documents.* In *Proc. European Conference on Spatial Information Theory*, volume 716 of *LNCS*, pages 377–390, 1993. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.48.471.

[CK95] Jeromy Carriere and Rick Kazman: *Interacting with huge hierarchies: Beyond cone trees.* In *Proc. IEEE Information Visualization '95, IEEE Computer Press, Los Alamitos, CA*, pages 74–81. IEEE, 1995.

[DM11] Walter Didimo and Fabrizio Montecchiani: *On the visualization of large hierarchically clustered graphs.* Tech. report RT-003-11, Dipartimento di Ingegneria Elettronica e dell'Informazione, Università di Perugia, 2011.

[DMM97] Ugur Dogrusöz, Brendan Madden, and Patrick Madden: *Circular layout in the graph layout toolkit.* In Stephen North (editor): *Graph Drawing*, volume 1190 of *Lecture Notes in Computer Science*, pages 92–100. Springer Berlin / Heidelberg, 1997, ISBN 978-3-540-62495-0. http://dx.doi.org/10.1007/3-540-62495-3_40.

[Dür05] Juan C. Dürsteler: *The landscape metaphor.* Inf@Vis! - The digital magazine of InfoVis.net, 168, 2005. http://www.infovis.net/printMag.php?lang=2&num=168, [Online; accessed 01-November-2011].

[Ead84] Peter Eades: *A Heuristic for Graph Drawing.* Congressus Numerantium, 42:149–160, 1984.

[EF97] Peter Eades and Qing Wen Feng: *Multilevel visualization of clustered graphs.* In Stephen North (editor): *Graph Drawing*, volume 1190 of *Lecture Notes in Computer Science*, pages 101–112. Springer Berlin / Heidelberg, 1997, ISBN 978-3-540-62495-0. http://dx.doi.org/10.1007/3-540-62495-3_41.

[FCE95] Qing Feng, Robert Cohen, and Peter Eades: *How to draw a planar clustered graph.* In Ding Zhu Du and Ming Li (editors): *Computing and Combinatorics*, volume 959 of *Lecture Notes in Computer Science*, pages 21–30. Springer Berlin / Heidelberg, 1995, ISBN 978-3-540-60216-3. http://dx.doi.org/10.1007/BFb0030816.

[FMM10] Sara Irina Fabrikant, Daniel R. Montello, and David M. Mark: *The natural landscape metaphor in information visualization: The role of commonsense geomorphology.* J. Am. Soc. Inf. Sci. Technol., 61:253–270, February 2010, ISSN 1532-2882. http://dx.doi.org/10.1002/asi.v61:2.

[FR91] Thomas M. J. Fruchterman and Edward M. Reingold: *Graph drawing by force-directed placement.* Software: Practice and Experience, 21(11):1129–1164, 1991, ISSN 1097-024X. http://dx.doi.org/10.1002/spe.4380211102.

[Gör10] Robert Görke: *An Algorithmic Walk from Static to Dynamic Graph Clustering.* PhD thesis, Karlsruhe Institute of Technology (KIT), Department of Informatics, February 2010. http://digbib.ubka.uni-karlsruhe.de/volltexte/1000018288.

[HH06] Joshua Ho and Seok Hee Hong: *Drawing clustered graphs in three dimensions.* In Patrick Healy and Nikola Nikolov (editors): *Graph Drawing*, volume 3843 of *Lecture Notes in Computer Science*, pages 492–502. Springer Berlin / Heidelberg, 2006, ISBN 978-3-540-31425-7. http://dx.doi.org/10.1007/11618058_44.

[LK07] Katharina Lehmann and Stephan Kottler: *Visualizing large and clustered networks.* In Michael Kaufmann and Dorothea Wagner (editors): *Graph Drawing*, volume 4372 of *Lecture Notes in Computer Science*, pages 240–251. Springer Berlin / Heidelberg, 2007, ISBN 978-3-540-70903-9. http://dx.doi.org/10.1007/978-3-540-70904-6_24.

[New01] M. E. J. Newman: *The structure of scientific collaboration networks.* Proceedings of the National Academy of Sciences of the United States of Amer-

ica, 98(2):404–409, January 2001, ISSN 0027-8424. http://dx.doi.org/10.1073/pnas.021544898.

[New04] M. E. J. Newman: *Detecting community structure in networks.* The European Physical Journal B - Condensed Matter and Complex Systems, 38:321–330, 2004, ISSN 1434-6028. http://dx.doi.org/10.1140/epjb/e2004-00124-y.

[RMC91] George G. Robertson, Jock D. Mackinlay, and Stuart K. Card: *Cone Trees: animated 3D visualizations of hierarchical information.* In *Proceedings of the SIGCHI conference on Human factors in computing systems: Reaching through technology*, CHI '91, pages 189–194, New York, NY, USA, 1991. ACM, ISBN 0-89791-383-3. http://dx.doi.org/10.1145/108844.108883.

[Shn96] B. Shneiderman: *The eyes have it: a task by data type taxonomy for information visualizations.* Visual Languages, IEEE Symposium on, 0:336–343, September 1996, ISSN 1049-2615. http://dx.doi.org/10.1109/VL.1996.545307.

[Tam87] Roberto Tamassia: *On embedding a graph in the grid with the minimum number of bends.* SIAM J. Comput., 16:421–444, June 1987, ISSN 0097-5397. http://dx.doi.org/10.1137/0216030.

[Wal01] C. Walshaw: *A multilevel algorithm for force-directed graph drawing.* In Joe Marks (editor): *Graph Drawing*, volume 1984 of *Lecture Notes in Computer Science*, pages 31–55. Springer Berlin / Heidelberg, 2001, ISBN 978-3-540-41554-1. http://dx.doi.org/10.1007/3-540-44541-2_17.

[XCHT07] K. Xu, A. Cunningham, S. Hong, and B. Thomas: *GraphScape: integrated multivariate network visualization.* In *Visualization, 2007. APVIS '07. 2007 6th International Asia-Pacific Symposium on*, pages 33–40, 2007. http://dx.doi.org/10.1109/APVIS.2007.329306.

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Karlsruhe, den 14. Dezember 2011

(Jan Christoph Athenstädt)