DenGraph-HO: A Density-based Hierarchical Graph Clustering Algorithm

Nico Schlitter, Tanja Falkowski and Jörg Lässig

Abstract

DenGraph-HO is an extension of the density-based graph clustering algorithm DenGraph. It is able to detect dense groups of nodes in a given graph and produces a hierarchy of clusters which can be efficiently computed. The generated hierarchy can be used to investigate the structure and the characteristics of social networks. Each hierarchy level provides a different level of detail and can be used as the basis for interactive visual social network analysis. After a short introduction of the original DenGraph algorithm we present DenGraph-HO and its top-down and bottom-up approaches. We describe the data structures and memory requirements and analyse the run time complexity. Finally, we apply the DenGraph-HO algorithm to real-world datasets obtained from the online music platform Last.fm and from the former U.S. company Enron.

1. Introduction

In 2011, we proposed DenGraph-HO in order to fulfill the special needs of social network analysts (Schlitter, Falkowski & Lässig 2011). In most cases, the visual inspection of a network is the first step of the analytical process and helps to determine the basic graph characteristics and further actions. DenGraph-HO supports this early stage by providing a quick visual analysis of the network structure. It provides the ability of zooming into network clusterings and has proven its usefulness for our practical work.

Our approach differs from traditional hierarchical clustering methods in that DenGraph-HO is a non-partional clustering algorithm. We consider the fact that not all nodes are necessarily members of clusters. In addition, the proposed hierarchy is not strictly built up by the classic divisive or agglomerative approach that is known from literature. We generalize these methods and propose a top-down approach and a bottom-up approach by extending the hierarchy paradigms. The proposed hierarchy supports superordinate clusters that contain subclusters.

Each level of the hierarchy represents a clustering that fulfills the original DenGraph paradigms, which will be presented in Section 2.1. The levels, respectively the clusterings, differ in the density that is required to form a cluster. While lower level clusterings aggregate nodes with a lower similarity, higher level clusterings require a higher similarity between nodes. The density-based cluster criteria are controlled by the parameters η and ε which are iteratively in-

or decreased to obtain different levels of the hierarchy. An existing clustering is used to compute the clustering of the next level. The efficiency of our algorithm is based on this iterative sequence of cluster adaptations instead of a complete new clustering.

The remainder of this article is organized as follows. Section 2 discusses related work and introduces the original DenGraph algorithm and its variations DenGraph-O and DenGraph-IO. Section 3 covers the proposed top-down and bottom-up approaches of DenGraph-HO as well as the used data structures, their memory requirements and the algorithm's run time complexity. Its usability is demonstrated in Section 4 by applying DenGraph-HO to two real-world datasets. Finally, a conclusion and an outlook are given in Section 5.

2. Related and Previous Work

Clustering is a data mining method which is used to detect hidden structures in huge data sets. Its purpose is to bundle of single objects into groups in such a way that objects of the same group are more similar to each other than to objects of other groups.

K-means (MacQueen 1967) is a commonly used and well studied clustering algorithm for spatial data. The algorithm strictly groups data all points into clusters. The number of clusters has to be predefined and stays constant during the clustering process. Each data point belongs to exactly one cluster. This can been seen as one major drawback of the algorithm since it does no deal with the concept of outliers.

Density-based approaches like DBSCAN (Ester, Kriegel, Sander & Xu 1996) require no previous knowledge of the number of clusters hidden in the data. Clusters are defined as regions that have a high density of data points and that are surrounded by regions of data points of lower density. Since data points may not belong to any cluster, DBSCAN is a non-partitioning clustering method, which provides the ability to handle outliers. The major drawback of DBSCAN is its inability to find nested or overlapping clusters. Therefore, the authors extended their original approach to overcome this problem. The algorithms OPTICS (Ankerst, Breunig, Kriegel & Sander 1999), HiCS (Achtert, Böhm, Kriegel, Kröger, Müller-Gorman & Zimek 2006), HiCO (Achtert, Böhm, Kröger & Zimek 2006) and DiSH (Achtert, Böhm, Kriegel, Kröger, Müller-Gorman & Zimek 2007) were designed to generate cluster hierarchies.

Since density-based clustering approaches provide significant advantages they have also been used for graph clustering purposes. DenShrink (Huang, Sun, Han & Feng 2011), a parameter-free algorithm for community detection, applies modularity optimization to reveal the embedded hierarchical structures with various densities in large-scale weighted undirected networks.

DenGraph (Falkowski, Barth & Spiliopoulou 2007) and SCAN (Xu, Yuruk, Feng & Schweiger 2007) have been independently proposed by extending DBSCAN. Both methods operate on undirected graphs, use a local cluster criteria and provide the ability to detect communities in social networks. However, a comparison of both methods (Kriegel, Kröger, Ntoutsi & Zimek 2011) pointed out that they significantly differ in the applied similarity function. DenGraph operates on weighted graphs and considers the weight of an edge between two nodes as similarity of these nodes. On the contrary, SCAN uses a similarity function that is based on the topology of the underlying graph. The similarity of two nodes correlates to the number of neighbors they share. Later, SCAN's similarity function was also used for the divisive hierarchical clustering DHSCAN (Yuruk, Mete, Xu & Schweiger 2007) and the agglomerative hierarchical clustering AHSCAN (Yuruk, Mete, Xu & Schweiger 2009).

Since 2007, the development of DenGraph continued as well. We proposed the extensions DenGraph-O that allows overlapping clusters and DenGraph-IO which tracks community evolution over time (Falkowski, Barth & Spiliopoulou 2008). In the following, we briefly introduce the original DenGraph algorithm and its extensions. In section 3 we then propose the density-based hierarchical algorithm DenGraph-HO.

2.1. DenGraph

Given a graph $G = (V,E)$ consisting of a set of nodes V and a set of weighted, undirected edges E, the DenGraph algorithm produces a clustering $\zeta = \{C_1,\ldots,C_k\}$ where each cluster C_i $(i = 1\ldots k)$ consists of nodes $V_{C_i} \subseteq V$. Since DenGraph is a non-partitioning clustering algorithm there can be *noise nodes* $V_N = \{u \in V \mid u \notin C_i\}$ that are not part of any cluster C_i. The remaining non-noise nodes are members of only one cluster and either *core nodes* or *border nodes*.

Definition The ε-neighborhood $N_\varepsilon(u)$ of a node $u \in V$ is defined as set of nodes that are connected to u having a distance less than or equal to ε. $N_\varepsilon(u) = \{v \in V \mid \exists(u,v) \in E \wedge dist(u,v) \leq \varepsilon\}$, where $dist(u,v)$ is the distance between u and v.

A node $u \in V$ is considered as *core node* if it has an ε-neighborhood of at least η neighbor nodes ($|N_\varepsilon(u)| \geq \eta$). Nodes which are in the ε-neighborhood of a core node, but do not have an own ε-neighborhood are called *border nodes*.

According to (Falkowski 2009), the actual cluster criterion is based on the concepts *directly density-reachable*, *density-reachable* and *density-connected* which are defined below and illustrated in Figure 1.

Definition Let $u,v \in V$ be two nodes. Node u is *directly density-reachable* from v within V with respect to ε and η if and only if v is a core node and u is in its ε-neighborhood, i.e. $u \in N_\varepsilon(v)$.

Definition Let $u,v \in V$ be two nodes. Node u is *density-reachable* from v within V with respect to ε and η if there is a chain of nodes $p_1,\ldots,p_n$ such that $p_1 = v, p_n = u$ and for each $i = 2,\ldots,n$ it holds that p_i is directly density-reachable from p_{i-1} within V with respect to ε and η.

Definition Let $u,v \in V$ be two nodes. u is *density-connected* to v within V with respect to ε and η if and only if there is a node $m \in V$ such that u is density-reachable from m and v is density-reachable from m.

Definition Let $G(V,E)$ be an undirected, weighted graph. A non-empty set $C \subseteq V$ is denoted as *Cluster* with respect to ε and η if and only if:

- For all $u,v \in V$, if $u \in C$ and v is density reachable from u, then $v \in C$.
- For all $u,v \in C$ u is density-connected to v within V with respect to ε and η.

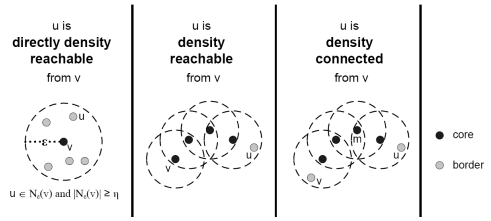


Figure 1: Concepts of Connectivity (Falkowski 2009)

The complete DenGraph procedure is described in Algorithm 1. It uses a stack in order to process the graph nodes. In a first step, all nodes V are marked as noise. Afterwards, each so far unprocessed node v is visited and checked if it has an ε-neighborhood. If the neighborhood contains at least η nodes ($|N(v)| \geq \eta$) the node v is marked as core and a new cluster is founded. Each of v's neighbors within the ε-neighborhood is marked as border, becomes a member of the new cluster and is pushed on the stack. After processing all neighbors, each node u from the stack is checked regarding having an ε-neighborhood and is marked correspondingly. If u became a core node, all of it's neighbors are marked as border and pushed on the stack. This procedure is repeated until all nodes of the graph are processed.

According to Algorithm 1 the time complexity of our procedure mainly depends on the number of nodes and the number of edges. Each node is visited once and each edge is processed up to two times - once from both end-nodes. Consequently, the overall run time complexity of the DenGraph algorithm is $O(|V| + |E|)$, where $|V|$ is the number of nodes and $|E|$ the number of edges. (Falkowski 2009)

Figure 2 shows the clustering of an exemplary interaction graph. The graph was clustered by applying the original DenGraph. Core nodes are blue, border nodes are green and noise nodes are

Algorithm 1: DenGraph

```
input : Graph,Clustering ζ,η,ε
output: Clustering ζ
begin
    foreach (u ∈ V|u.state = noise) do
        if (|N_ε(u)| ≥ η) then
            Cluster=CreateNewCluster();
            Cluster.addNode(u);
            u.state=core;
            foreach n ∈ N_ε(u) do
                Cluster.addNode(n);
                n.state=border;
                stack.push(n);
            repeat
                v=stack.pop();
                if (|N_ε(v)| ≥ η) then
                    v.state=core;
                    foreach
                    n ∈ N_ε(v)|n.state ≠ core do
                        Cluster.addNode(n);
                        n.state=border;
                        stack.push(n);
            until stack is empty;
    return ζ;
```

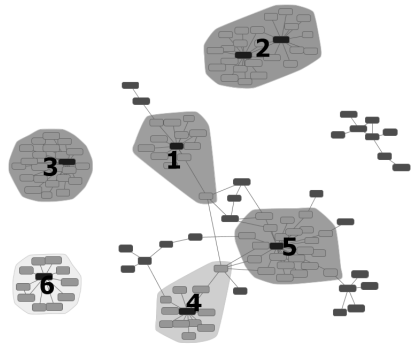


Figure 2: DenGraph Clustering (Falkowski 2009)

drawn in red color. Each cluster contains one or more core nodes which are connected with each other. Following the DenGraph paradigm, the total number of nodes per cluster is greater than η. Nodes that are not member of any cluster are considered as noise nodes.

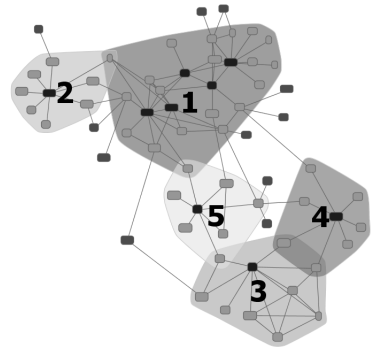


Figure 3: DenGraph-O Clustering (Falkowski 2009)

2.2. *DenGraph-O*

Practical work with DenGraph in the field of Social Network Analysis revealed a minor drawback: While in real world applications nodes - respectively human beings - might be part of more than one community, the original DenGraph algorithm does not allow for clusters to overlap. The affected nodes were exclusively assigned to only one cluster among the cluster candidates.

This issue was addressed in (Falkowski et al. 2008). The extended version, called DenGraph-O, allows border nodes that are members of multiple clusters. As a consequence several clusters of a graph might be overlapped. An example for those overlapping clusters is illustrated in Figure 3. Cluster 1 overlaps with Cluster 2 and Cluster 5 while Cluster 3 overlaps with Cluster 5 and Cluster 4.

2.3. *DenGraph-IO*

In 2007, Falkowski et al. proposed DenGraph-IO (Falkowski & Barth 2007, Falkowski et al. 2008) to analyse the dynamics of communities over time. The authors compared the changes between clusterings that were obtained in different points in time. For this, it is necessary to iteratively compute the graph clusterings that are subsequently observed over time. A huge computational effort would be necessary if the original DenGraph was used to process multiple consecutive snapshots of social networks. However,

as social structures often change slowly over time, the graphs G_t and G_{t+1} differ just slightly. Therefore, a complete re-clustering, as the use of the original DenGraph algorithm would demand, is quite inefficient.

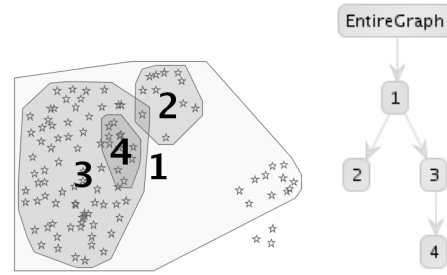
The incremental clustering algorithm DenGraph-IO addresses this issue and updates an existing clustering based on the changes of the underlying graph. The graph changes either by adding/removing of nodes, adding/removing of edges or changing the weight of an existing edge. As a result of the graph updates, new clusters may appear or existing clusters may be removed, merged or split.

Since the DenGraph-IO algorithm deals exclusively with the parts of the graph that changed from one point in time to the other, the computational complexity is dramatically reduced and even huge networks can be processed in reasonable time. Our experiments using a real-world social network obtained from *Last.fm* show that handling 2,500 graph updates using DenGraph-IO is about 400 times faster than the re-clustering with DenGraph-O (Schlitter & Falkowski 2009, Falkowski 2009).

3. DenGraph-HO

One challenge of the DenGraph algorithm is the choice of the parameters ε and η. Several heuristics were investigated (Falkowski 2009), however, the "right" parameter combination mainly depends on the aim of the analysis. If the analyst is for example interested in observing strongly connected nodes rather than in clusterings that show the overall structure, the parameters need to be chosen accordingly. DenGraph-HO addresses this issue by allowing a quick visual inspection of clusterings for a given set of parameter combinations. The process of zooming into or out of the network can be interactively controlled by the analyst according to his needs.

The proposed algorithm returns a hierarchical clustering that describes the structure of the underlying network. The resulting hierarchy provides multiple views of the network structure in different levels of detail. Consequently, the cluster hierarchy is an ideal basis for an efficient zooming implementation. Zooming-in is done by stepping up in the hierarchy. It provides a more detailed view of the current cluster by presenting its subclusters. A higher level of abstraction is reached by zooming-out, which is equivalent to merging similar clusters into superordinate clusters.



(a) Hierarchical Clusterings (b) Hierarchy

Figure 4: Visualization of an Hierarchical DenGraph-HO Clustering

Figure 4 shows an exemplary graph clustering and the corresponding hierarchy as a tree of clusters. For the sake of clarity we removed the graph edges. The root of the hierarchy tree represents the whole graph and children represent subclusters of their superordinate cluster. Following this definition, the leaves of the tree correspond to the smallest clusters that have no subclusters.

The proposed DenGraph-HO hierarchy is based on the concepts of the DenGraph algorithm. Each level of the tree (besides the root) represents a valid clustering that fulfills the DenGraph-O paradigms. The hierarchy can be built by repeatedly applying DenGraph-O while using specific parameter settings for each level of the tree. The choice of the parameters η and ε is limited by constraints in order to ensure that lower level clusters are subclusters of the superordinate cluster.

Let us assume that the clustering ζ_l forms level l of the hierarchy and is computed by applying DenGraph-O with the parameters ε_l and η_l. Level $l+1$ represents a clustering that is based on ε_{l+1} and η_{l+1} and guarantees a higher

similarity of nodes in the cluster. According to the description above, ζ_{l+1} has to contain subclusters of clusters that are element of ζ_l. In order to preserve this parent-child relationship we have to ensure the following constraints:

1) The parameter ε_l that is used to generate the clustering ζ_l has to be bigger than or equal to ε_{l+1} which is used to compute the clustering ζ_{l+1} : $\varepsilon_l \geq \varepsilon_{l+1}$.
2) The parameter η_l that is used to generate the clustering ζ_l has to be lower than or equal to η_{l+1} which is used to compute the clustering ζ_{l+1} : $\eta_l \leq \eta_{l+1}$.

Increasing ε might lead to a transition of a node state from noise or border to core or from noise to border. By increasing ε a core node cannot lose its state. This explains why increasing ε might create a new cluster or expand an existing one and why it surely avoids cluster reductions or removals. The same argument holds for decreasing η and shows why the demanded cluster-subcluster relation can be guaranteed by the given constraints.

In the following, we discuss how the proposed cluster hierarchy can be efficiently generated based on a list of parameter settings that fulfill the discussed constraints. An obvious approach would be to perform multiple re-clusterings using DenGraph-O until each parameter setting is processed. However, this is very inefficient and would be a huge computational effort.

The proposed DenGraph-HO algorithm addresses this issue and uses incremental parameter changes to generate the cluster hierarchy. Instead of computing a complete new clustering for each level, an existing clustering is used and adapted. In the following, we discuss how an existing clustering of level l can be used to compute the clusterings of level $l+1$ and $l-1$. We propose a bottom-up and top-down approach and analyse their efficiency depending on the graph structure. The input for both approaches is a graph $G = (V,E)$ and an existing clustering $\zeta_l = \{C_1^l, \ldots, C_k^l\}$ that fulfills the DenGraph paradigms.

3.1. Top-down Approach: Cluster Reduction, Split or Removal

The top-down approach performs a zoom-in operation and generates a new clustering for level $l+1$ of the hierarchy. Clusters of level l might be reduced, split or removed. By decreasing ε and increasing η the state of nodes within a cluster might change. A former border node might become noise (Cluster Reduction). Former core nodes might get border state (possible Cluster Splitting) or noise state (Cluster Reduction, possible Cluster Splitting or Removal).

Due to the DenGraph paradigms, it is guaranteed that noise nodes can not reach border or core state by decreasing ε or increasing η. Thus, noise nodes will not change their state and do not need to be processed. Consequently, the top-down approach traverses just border and core nodes and performs a re-clustering for each existing cluster. For this purpose, we use the modified DenGraph-O procedure shown in Algorithm 3. Each cluster C of level l is re-clustered by applying the parameters of level $l+1$.

Algorithm 2 shows how the modified DenGraph-O algorithm is used to create a complete hierarchy. After generating a parameter list of (ε,η)-pairs an initial DenGraph-O clustering for level l=1 is iteratively adapted to generate the subsequent clusterings for the levels l+i.

Algorithm 2: Top-down_Approach

input : Graph,l_{max}
output: Clustering ζ

PL=CreateParameterSettingList(l_{max});
ζ_1=DenGraph(Graph,PL[1].η,PL[1].ε);
for $l = 2$ **to** l_{max} **do**
  ζ_l=Top-down_Step(ζ_{l-1},PL[l].η,PL[l].ε);
return ζ;

Obviously, the characteristic of the cluster hierarchy depends on the values specified in the parameter list. For a given social network analysis task, it is quite hard to find appropriate parameters, that lead to a useful hierarchy - especially if there is no prior knowledge about

the given social network. We are approaching this problem by choosing parameter values which are equally distributed over the parameter search space. On this basis, parameter ranges that lead to useful hierarchal structures can be discovered. Later on, they can be explored in more detail by re-applying the clustering method with new parameter settings.

Algorithm 3: Top-down_Step

```
input : Graph,ζ,ε,η
output: Clustering ζ
foreach (C ∈ ζ) do
    foreach r ∈ C do r.state=noise;
    foreach (u ∈ C|u.state = noise) do
        if (|N_ε(u)| ≥ η) then
            Cluster=CreateNewCluster();
            C.addCluster(Cluster);
            Cluster.addNode(u);
            u.state=core;
            foreach n ∈ N_ε(u) do
                Cluster.addNode(n);
                n.state=border;
                stack.push(n);
        while stack is empty do
            v=stack.pop();
            if (|N_ε(v)| ≥ η) then
                v.state=core;
                foreach n ∈ N_ε(v)|n.state ≠ core do
                    Cluster.addNode(n);
                    n.state=border;
                    stack.push(n);
return ζ;
```

3.2. *Bottom-up Approach: Cluster Creation, Absorption and Merging*

The bottom-up approach performs a zoom-out operation and generates a new clustering on level $l-1$ of the cluster hierarchy. Existing clusters of level l might grow through the absorption of nodes or through the merge with other clusters.

Our approach deals with changes of η only, because changing ε as well would make it necessary to process all graph nodes. However, if we process all graph nodes our approach would not be more efficient than the original DenGraph algorithm.

Algorithm 4 shows how we expand the existing clusters by the iterative increase of η. Doing so, the state of a core node remains unchanged. A former noise node may become a core node (cluster creation) or a border node (absorption). A former border node could become core node (absorption). In case a former border node is member of multiple overlapping clusters, its transition to core state leads to a merge of those clusters.

Algorithm 4: Bottom-up_Approach

```
input : Graph,l_max,eta_min,ε
output: Clustering ζ
η=η_min;
ζ_{l_max}=DenGraph(Graph,η,ε);
for l = l_max − 1 to 1 do
    η=η+1;
    ζ_l=Bottom-up_Step(ζ_{l+1},η_l,ε);
return ζ;
```

Algorithm 5 describes the procedure that performs the bottom-up step by processing the changes of η. Since core nodes keep their state, there is no need to consider them in the bottom-up approach. Consequently, the proposed procedure traverses only noise and border nodes in order to determine their new state and to adapt the clustering accordingly. Following this argumentation, the procedure's efficiency is based on the saved time that the original DenGraph-O would have spent for processing core nodes.

First, the algorithm iterates over all existing clusters in order to expand them. For each cluster it traverses all border nodes and updates their state according to the parameters η and ε. In case a former border node becomes core, this new core node is pushed on the stack for further processing. After dealing with all border nodes, the procedure *Expand* checks if the new core nodes absorb their neighbors into the cluster. In case an absorbed node has less than η neighbors in its ε-neighborhood it gets border state,

Algorithm 5: Bottom-up_Step

```
input : Graph,Clustering ζ,ε,η
output: Clustering ζ
foreach (C ∈ ζ) do
    Cluster=CreateNewCluster();
    Cluster.addSubCluster(C);
    foreach (u ∈ C|u.state = border) do
        if (|N_ε(u)| ≥ η) then
            u.state=core;
            stack.push(u);
    Expand(stack,Cluster,ε,η);
ζ=DenGraph(Graph,ζ,ε,η);
return ζ;
```

Algorithm 6: Expand

```
input : Graph,Clustering,Cluster,ε,η
output: Clustering
while Stack is not empty do
    u=stack.pop();
    if (u is member of multiple clusters) then
        foreach (C ∈ ζ|u ∈ C) do
            Cluster.addSubCluster(C);
            foreach (p ∈ C|p.state = border) do
                if (|N_ε(p)| ≥ η) then
                    p.state=core;
                    stack.push(p);
                else
                    p.state=border;
    foreach (n ∈ N_ε(u)|n.state ∈ {noise,border}) do
        Cluster.addNode(n);
        if (|N_ε(n)| ≥ η) then
            n.state=core;
            stack.push(n);
        else n.state=border;
return Clustering;
```

otherwise it becomes a core node. These newly discovered core nodes are pushed on the stack and the procedure is repeated until no further nodes are absorbed into the cluster.

Due to the possibility of overlapping clusters, a new core node might have been a member of multiple clusters before. Consequently, its new core state leads to a merge of the affected clusters into a superordinate cluster.

After dealing with all border nodes the existing clusters are maximal expanded with respect to the changed η and the constant ε. The remaining noise nodes are processed to check whether their state has changed. In case a former noise node becomes core, a new cluster is created. To search for new clusters we use the original DenGraph algorithm, which processes exclusively noise nodes. Please note, that during the handling of these noise nodes a newly created cluster will not merge with an existing one. If the new cluster were in ε-distance to another cluster, the nodes of the new cluster would have been already absorbed by this existing cluster during its expansion phase.

3.3. Data Structures

In the last decade, the number of social networks and their participating users has rapidly increased. Following this trend, DenGraph-HO was developed to process efficiently even huge graphs like facebook or twitter that have millions of nodes and edges. To achieve this challenging aim, the used data structures need to be optimized for the most frequent operations of the proposed clustering algorithm. In the following, we briefly describe the used data structures and their memory requirements.

The DenGraph-HO algorithm is implemented using Java. All data structures including the graph $G = (V,E)$, the nodes V and the edges E are modeled as single classes. The corresponding objects are instantiated during the graph built-up phase and each object is singularly materialized in memory even if there are several references to this object.

According to Algorithm 2 and 4, the top-down approach as well as the bottom-up approach need a valid clustering as starting point which is generated by the original DenGraph algorithm. During this initial application of DenGraph procedure, the algorithm traverses all noise nodes in order to update the node states based on the cardinality of their ε-neighborhood. For an efficient implementation, a list structure is needed that contains references to the noise nodes, allows rapid node

traversing and also provides efficient *insert* and *remove* operations for the current item. The class *LinkedList* provided by Java fulfills these requirements. Traversing the list takes linear time and works in $O(n)$, the *insert* and *remove* operations on the current item are handled in constant time $O(1)$.

Because it is basically a modified DenGraph which performs the top-down step, Algorithm 3 benefits from the list of noise nodes as well.

We implemented a similar list of border nodes for each cluster because the bottom-up step, implemented in Algorithm 5, needs to traverse all border nodes. Therefore, traversing nodes is efficiently done for all procedures of the algorithm.

In order to update the state of a node, the cardinality of a node's ε-neighborhood must be determined by counting edges that have weight values less than ε. Due to the fact that this operation is frequently used, we decided to implement a binning mechanism for each node which assigns, depending on the weight, the edges of each node to a specific bin. In case the parameter ε_l for each level l is specified in advance, the ranges of the binning mechanism can be set accordingly. This can be done for example during initializing the graph and takes linear time $O(n)$ where n denotes the number of nodes in the graph.

For example, choosing $\varepsilon_{l=1} = 0.7$ and $\varepsilon_{l=2} = 0.25$ to generate a hierarchy with two levels, the ranges for the binning mechanism should be set to the same values. By this, all edges with a weight less than 0.25 are put in bin 1, edges with a weight between 0.25 and 0.75 are stored in bin 2. The remaining edges are put into bin 3. According to Algorithm 2, for each non-noise node u the cardinality of $N_{0.25}(u)$, and $N_{0.7}(u)$ must be determined. Due to the binning mechanism, which counts the number of edges in each bin, this task can be performed in $O(l)$ where l denotes the number of bins.

3.4. Run Time Complexity

The run time complexity of the original DenGraph applied on a graph G(V,E) is $O(|V|+|E|)$, where $|V|$ is the number of nodes and $|E|$ the number of edges. (Falkowski 2009).

By applying DenGraph k times with k different (ε,η)-pairs, we would be able to produce a hierarchy with k levels similar to the ones generated by our DenGraph-HO algorithm. However, this would be quite inefficient since for each iteration each node is traversed even if the node's state definitely will not change. As described before, DenGraph-HO overcomes this problem and traverses only relevant nodes. In fact, for each level of the hierarchy, DenGraph-HO deals only with the subgraph $G'(V',E')$ where $V' \subset V$ and $E' \subset E$. In the worst case scenario, if the original graph G equals G', DenGraph-HO has the same run time like applying the original DenGraph algorithm k times. However, our practical work has shown that depending on the parameters ε and η the subgraph G' is up to 50% smaller than the original graph G. This leads to a significant run time reduction, which has a huge impact on practical work. However, since this improvement reduces the run time just by a constant factor, the complexity according to the O-notation does not change and is still $O(|V|+|E|)$.

3.5. Memory Requirements

In the following, the memory requirement of the data structures is briefly described. It depends on the number of nodes $|V|$, the number of edges $|E|$ and the number of hierarchy levels l_{max}. Assuming a 32bit environment, each pointer to a single object requires 4 byte.

Each edge contains references to the two nodes that are connected by this edge. In addition, the weight of this connection is stored. Consequently, each edge needs $2 \times 4 + 4 = 12$ byte. The required memory to store all edges is $|E| \times 12$ byte.

For each node its identifier, the current state (noise, border or core), the cluster(s) to which the node belongs and a reference to the edge binning structure that holds information about the node's edges are stored. Due to memory requirements of $4 + 4 + (4 \times l_{max}) + 4 = 12 + 4 \times l_{max}$ byte per node, the whole set of nodes needs $(12 + 4 \times l_{max}) \times |V|$ byte.

The graph structure stores a list of references to all edges, a list containing the border and core nodes and a list of all noise nodes. Since we use a LinkedList there is a need for a forward pointer, a backward pointer and the reference to the actual node. In total the requirement of these lists is $12 \times |E| + 12 \times |V|$ byte.

The binning structure for the edges contains l_{max} bins that are implemented as a LinkedList of edge references. Since each edge is listed in two binning structures, we calculate a total memory need of $2 \times |E| \times 12$ byte for all binning structures of the graph. This already includes the for- and backward pointers of our list implementation. The addition storage of the binning ranges takes $|V| \times l_{max} \times 4$ byte.

Each cluster contains references to the nodes of this cluster. For each hierarchy level, we assume that each node is member of one cluster. As a result, the clustering demands for a total of $|V| \times l_{max} \times 12$ byte.

During processing, additional memory for the stack structure is required. In the worst case scenario, this stack holds all nodes and is then limited to $|V| \times 4$ bytes.

In total, we calculate a memory requirement of $|E| \times 48 + |V| \times (28 + 20 \times l_{max})$ byte to store all data structures of the DenGraph-HO algorithm. Consequently, a 3-level-clustering of a graph that has 1200 nodes and 12600 edges requires about 710 KB.

4. Applications

In the following, we apply the DenGraph-HO algorithm on different datasets and show its ability for explorative visual social network analyses. The presented algorithm was implemented as a plugin for the open-source data mining framework RapidMiner (Mierswa, Wurst, Klinkenberg, Scholz & Euler 2006) which was formerly known as YALE. For graph visualization we used the prefuse toolkit developed by Jeffrey Heer (Heer, Card & Landay 2005).

The first case study demonstrates the algorithm's usefulness and analyses the email communication of the former U.S. company *Enron*. This data was published by the Federal Energy Regulatory Commission during the investigation of the biggest bankruptcy case in US-history. For our second analysis, we use information about user's music listening behavior provided by the online music platform *Last.fm*.

4.1. Enron Case Study

The collapse of *Enron*, a U.S. company honored in six consecutive years by "Fortune" as "America's Most Innovative Company", caused one of the biggest bankruptcy cases in US-history. To investigate the case, a data set of approximately 1.5 million e-mails sent or received by *Enron* employees was published by the Federal Energy Regulatory Commission.

Because of the public interest and the rareness of available real world e-mail data many studies has been carried out on the *Enron* e-mail dataset. The original dataset had some major integrity problems and was cleaned up by Melinda Gervasio at the independent, nonprofit research institute SRI. The revised dataset was used by Klimt and Yang when they presented their introduction of then *Enron* corpus (Klimt & Yang 2004). Bekkerman et al. use the dataset for experiments in e-mail foldering and classification (Bekkerman, McCallum & Huang 2004). Diesner et al. analysed the evolution of structure and communication behavior of the employees on different organizational levels (Diesner, Frantz & Carley 2005). Shetty and Adibi published a subset of the original data containing approximately 250,000 e-mails from/to 151 *Enron* employees which were sent during 1998 and 2002 (Shetty & Adibi 2004). In 2011, we used this dataset to analyse the evolution of communities over time (Falkowski et al. 2008, Falkowski 2009).

For our experiments with the DenGraph-HO clustering algorithm, we use a subset of this dataset containing only messages sent from *Enron* employees to *Enron* employees.

Traditionally, clustering is based on the *distance* between the objects to be clustered. On a graph of interactions, we model distance between two actors based on the number of their interactions. To deal with outliers we chose a value z so that about 1.5 percent of all edges have an edge

weight larger than z. Afterwards, we ensure that the weight of these edges is bounded to z.

Definition The distance between two actors u and v is defined as

$$distance(u,v) = \frac{min\{u \to v, v \to u, z\} - 1}{z - 1}, \quad (1)$$

where $u \to v$ is the number of messages sent from u to v, $v \to u$ the number of messages sent from v to u and z is a value specified for handling outliers.

The distance function ranges in $[0,1]$ and is symmetric. If only one reciprocated interaction exists between u and v, then their distance is one.

For our experiments we used a graph consisting of 57 nodes and 81 edges that represents the internal *Enron* communication between 2000/12/04 and 2000/12/10. We applied the presented top-down approach and retrieved the cluster hierarchy as shown in Figure 6. For each hierarchy level the used parameters η and ε, the number of resulting clusters and their sizes are listed in Table 1. Figure 5 shows the graphical representation of the interaction graph and the computed clusters.

Due to the fact that the four members of Cluster 2 did not communicate with the other employees there is no connection between Cluster 2 and the bigger Cluster 1. Within Cluster1 the three subclusters Cluster 3, Cluster 4 and Cluster 5 appear. A reason might be, that the respective cluster members work together in a team or in a project. At level three, Cluster 3 splits into Cluster 6 and Cluster 7. Since the

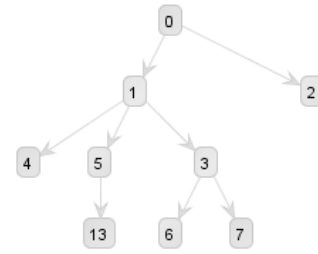


Figure 6: Hierarchy of the Enron Clustering

Table 1: Hierarchy of the Enron Clustering

Level	ε	η	**Labels**	**# Nodes**
0	-	-	Graph	57
1	1	2	Cluster 1	53
			Cluster 2	4
2	0.988	3	Cluster 3	15
			Cluster 4	11
			Cluster 5	5
3	0.975	4	Cluster 6	8
			Cluster 7	5
4	0.963	5	No changes	
5	0.95	6	Cluster 13	9

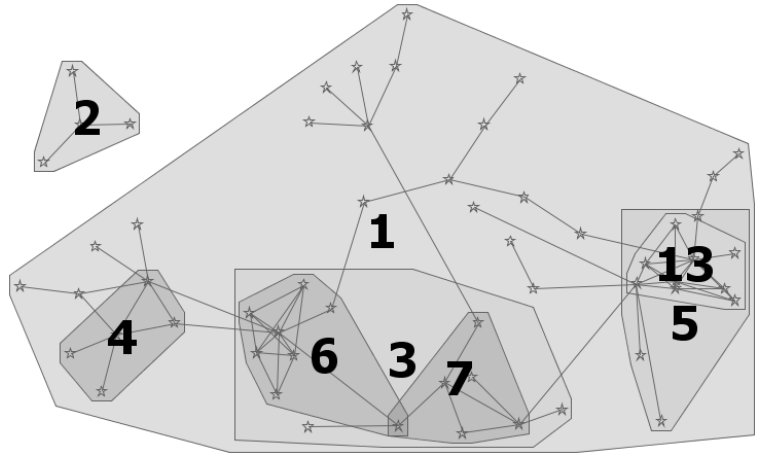


Figure 5: Hierarchical Clustering of the Enron Graph

Table 2: Hierarchy of the Last.fm Clustering

Level	ε	η	Cluster Labels	# Nodes
0	-	-	Entire Graph	1209
1	0.05	22	hip-hop	35
			indie, rock, alternative, punk	631
			death metal, metal, heavy metal	85
2	0.037	24	hip-hop	31
			punk, rock, indie	86
			punk, rock	33
			indie, rock, alternative	316
			death metal, metal, heavy metal	37
3	0.025	26	indie, indie rock, rock	120

parameter combination of η and ε at level four leads to no cluster changes, Cluster 13 is finally formed within Cluster 5 at level five.

4.2. *Last.fm Case Study*

Last.fm is a music community with over 20 million active users based in more than 200 countries. After a user signs up, a media player plugin is installed and all tracks a user listens to are submitted to a database. *Last.fm* records - among others - all artists a user listens to and provides lists of the most frequently listened artists for each week over the lifetime of a user.

Last.fm provides for each user a list of the most frequently listened artists and the number of times the artist was played. We obtained the user listening behavior of 1,209 users over a periode of 130 weeks (from March 2005 to May 2008) and use this information to build user profiles by extracting the genres of the most listened artists.

For each artist, *Last.fm* provides the tags that are used by users to describe the artist. For example, the band "ABBA" has 41 tags. The most often used tags are "pop", "disco", "swedish" and "70s". The singer "Amy Winehouse" has 34 tags, the most often used tags are "soul", "jazz", "female vocalists" and "british".

Much work has been done using the collaborative music tagging data from *Last.fm*. Chen et al. studied the automatic classification of music genres (Chen, Wright & Nejdl 2009). The authors demonstrate the benefits of a classification technique that uses the tags supplied by the users for accurate music genre classification. Konstas et al. created a collaborative recommendation system for Last.fm based on both the social annotation and friendships between users (Konstas, Stathopoulos & Jose 2009). In 2009, we ap-

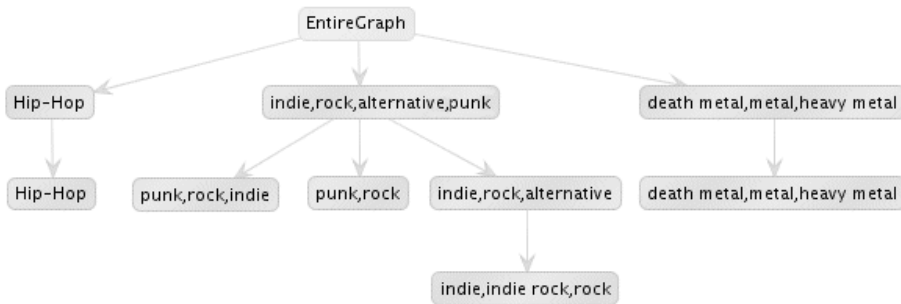


Figure 7: Hierarchy of the Last.fm Clustering

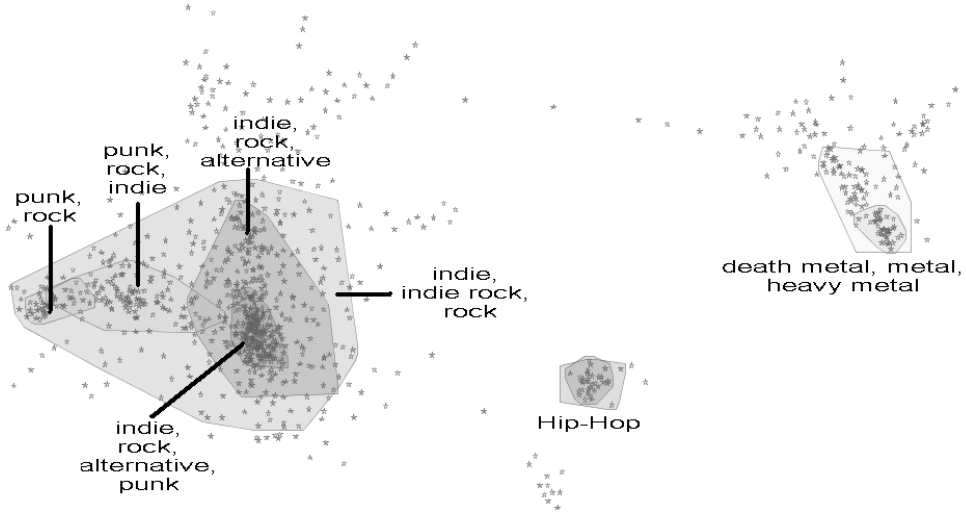


Figure 8: Hierarchical Clustering of the Last.fm Graph

plied DenGraph-IO on the dataset to analyse the dynamics of music communities (Schlitter & Falkowski 2009, Falkowski 2009). Geleijnse et al. used the data provided by *Last.fm* to investigate whether the tagging of artists is consistent with the artist similarities found with collaborative filtering techniques (Geleijnse, Schedl & Knees 2007). Since the authors found the data both consistent and descriptive for creating a ground truth for artist tagging and artist similarity we are following their approach. We use the tags provided by Last.fm for each artist to describe the artist in a genre vector $\vec{a}_i$.

Definition An *artist* a_i is defined as a *genre vector* $\vec{a}_i = (w_1, w_2, \ldots, w_k)$ of the k-most used genre tags $w_i \in W$

Definition For each user u a *user profile* is defined as

$$\vec{u} = \sum_{i=1}^{m} \vec{a}_i \cdot c_i, \tag{2}$$

while m is the number of artists listened to. The value c_i is provided by Last.fm and describes how often the artist a_i was listen to by user u.

Based on the user profiles we calculated the pairwise similarity of user music preferences and generated a graph in which the nodes represent the users and the edges encode the distance between users based on the similarity of their music listening behavior (Schlitter & Falkowski 2009).

Definition The *similarity* of music preferences between two users u and v is defined as

$$sim(u,v) = \frac{\sum_{i=1}^{m} u_i \cdot v_i}{\sqrt{\sum_{i=1}^{m} u_i^2 \cdot \sum_{i=1}^{m} v_i^2}} \tag{3}$$

For our analysis we used a *Last.fm* graph consisting of 1,209 nodes and 12,612 edges. We applied DenGraph-HO and obtained the cluster hierarchy shown in Figure 7. The calculated cluster labels are based on the user profiles of the cluster members.

Table 2 gives more details about the clustering of each hierarchy level. Shown are the parameters ε and η, the cluster label and the number of nodes for each cluster.

The semantical plausibility of the parent-child-relationship between a superordinate cluster and its subclusters can be demonstrated using the *(indie, rock, alternative, punk)*-cluster. This cluster is formed at level $l = 1$ and splits in the subclusters *(punk, rock, indie)*, *(punk, rock)* and *(indie, rock, alternative)* at level $l = 2$. While l increases, the size of the observable clusters decreases and their semantic meaning becomes more specific.

While traversing the hierarchy tree from the leaves to the root, the number of nodes per cluster increases. Due to the parameter setting of ε and η the clusters grow either through merging of clusters or through absorption of nodes. These properties of DenGraph-HO and the efficient calculation of the cluster hierarchy are the basis for the proposed zooming purpose.

Figure 8 shows the entire graph and the clusters found by DenGraph-HO. For the sake of clarity graph edges are not drawn. Since we limited the number of hierarchy levels in our example and due to the small number of nodes, the graph can easily be understood. However, graphs with millions of nodes and a hierarchy depth greater than ten ask for appropriate tools. As a result, the ability of zooming through the graph enriches our tool set and is an important step for studying the structure of huge graphs.

5. Conclusion and Outlook

In this article we proposed the hierarchical density-based graph clustering algorithm DenGraph-HO. The algorithm computes a hierarchy of clusters where each level of the hierarchy fulfills the DenGraph paradigms. The advantage of our method is based on the iterative change of clustering parameters which allows an efficient extension of a given clustering.

We demonstrated the algorithm's practical use for explorative visual network analysis by applying the algorithm both to a communication graph based on the e-mail correspondence within the former U.S. company *Enron* and to a social network obtained from the online music platform *Last.fm*.

Applied to the *Last.fm* data our algorithm forms clusters by grouping users that have similar music listening preferences. For each cluster we determined a cluster label that represents music genres and which is based on the music listening behavior of the cluster members. The resulting cluster hierarchy shows a plausible semantical relationship between superordinate clusters and subclusters. Furthermore, clusters that represent similar music genres are located closely both in the graphical representation and in the cluster hierarchy.

Since DenGraph-HO has proven its usefulness for our practical work, our next step is to extend our algorithm towards a scalable version that is suitable for a high performance computing environment. We will look into MapReduce and grid computing technologies in order to parallelize the algorithm execution. Thus, we will be able to apply our algorithm to huge social networks like Twitter or Facebook and to retrieve the corresponding cluster hierarchy in acceptable time.

In addition, we will investigate how to integrate the incremental approach known from the DenGraph-IO algorithm. The result will be an hierarchical incremental density-based graph clustering algorithm which is able to track the evolution of cluster hierarchies over time.

6. Acknowledgment

This study was supported by the members of the grid computing project *distributedDataMining* (http://www.distributedDataMining.org) which provided the necessary computational power for our graph clustering experiments.

References

Achtert, E., Böhm, C., Kriegel, H.-P., Kröger, P., Müller-Gorman, I. & Zimek, A. (2006), Finding hierarchies of subspace clusters, *in* J. Fürnkranz, T. Scheffer & M. Spiliopoulou, eds, 'PKDD', Vol. 4213 of *Lecture Notes in Computer Science*, Springer, pp. 446–453.

Achtert, E., Böhm, C., Kriegel, H.-P., Kröger, P., Müller-Gorman, I. & Zimek, A. (2007), Detection and visualization of subspace cluster hierarchies, *in* K. Ramamohanarao, P. R. Krishna, M. K. Mohania & E. Nantajeewarawat, eds, 'DASFAA', Vol. 4443 of *Lecture Notes in Computer Science*, Springer, pp. 152–163.

Achtert, E., Böhm, C., Kröger, P. & Zimek, A. (2006), Mining hierarchies of correlation clusters, *in* 'SSDBM', IEEE Computer Society, pp. 119–128.

Ankerst, M., Breunig, M. M., Kriegel, H.-P. & Sander, J. (1999), 'Optics: ordering points to identify the clustering structure', *SIGMOD Rec.* **28**(2), 49–60.

Bekkerman, R., McCallum, A. & Huang, G. (2004), 'Automatic categorization of email into folders: Benchmark experiments on enron and sri corpora', *Center for Intelligent Information Retrieval, Technical Report IR* **418**.

Chen, L., Wright, P. & Nejdl, W. (2009), Improving music genre classification using collaborative tagging data, *in* 'Proceedings of the Second ACM International Conference on Web Search and Data Mining', WSDM '09, ACM, New York, NY, USA, pp. 84–93.

Diesner, J., Frantz, T. L. & Carley, K. M. (2005), 'Communication networks from the enron email corpus "it's always about the people. enron is no different"', *Comput. Math. Organ. Theory* **11**(3), 201–228.

Ester, M., Kriegel, H.-P., Sander, J. & Xu, X. (1996), A density-based algorithm for discovering clusters in large spatial databases with noise, *in* 'Proc. of 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96)', pp. 226–231.

Falkowski, T. (2009), *Community Analysis in Dynamic Social Networks*, Sierke Verlag, Göttingen.

Falkowski, T. & Barth, A. (2007), 'Density-based temporal graph clustering for subgroup detection in social networks', Presented at Conference on Applications of Social Network Analysis.

Falkowski, T., Barth, A. & Spiliopoulou, M. (2007), Dengraph: A density-based community detection algorithm, *in* 'Proc. of the 2007 IEEE / WIC / ACM International Conference on Web Intelligence', IEEE Computer Society, Washington, DC, USA, pp. 112–115.

Falkowski, T., Barth, A. & Spiliopoulou, M. (2008), Studying community dynamics with an incremental graph mining algorithm, *in* 'Proc. of the 14 th Americas Conference on Information Systems (AMCIS 2008)', Toronto, Canada.

Geleijnse, G., Schedl, M. & Knees, P. (2007), The quest for ground truth in musical artist tagging in the social web era, *in* S. Dixon, D. Bainbridge & R. Typke, eds, 'ISMIR', Austrian Computer Society, pp. 525–530.

Heer, J., Card, S. K. & Landay, J. A. (2005), prefuse: a toolkit for interactive information visualization, *in* 'Proceedings of the SIGCHI conference on Human factors in computing systems', CHI '05, ACM, New York, NY, USA, pp. 421–430.

Huang, J., Sun, H., Han, J. & Feng, B. (2011), 'Density-based shrinkage for revealing hierarchical and overlapping community structure in networks', *Physica A Statistical Mechanics and its Applications* **390**, 2160–2171.

Klimt, B. & Yang, Y. (2004), Introducing the enron corpus, *in* 'First Conference on Email and Anti-Spam (CEAS)', Mountain View, CA.

Konstas, I., Stathopoulos, V. & Jose, J. M. (2009), On social networks and collaborative recommendation, *in* 'Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval', SIGIR '09, ACM, New York, NY, USA, pp. 195–202.

Kriegel, H.-P., Kröger, P., Ntoutsi, I. & Zimek, A. (2011), Density based subspace clustering over dynamic data, *in* J. B. Cushing, J. C. French & S. Bowers, eds, 'SSDBM', Vol. 6809 of *Lecture Notes in Computer Science*, Springer, pp. 387–404.

MacQueen, J. B. (1967), Some methods for classification and analysis of multivariate observations, *in* L. M. L. Cam & J. Neyman, eds, 'Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability', Vol. 1, University of California Press, pp. 281–297.

Mierswa, I., Wurst, M., Klinkenberg, R., Scholz, M. & Euler, T. (2006), Yale: Rapid prototyping for complex data mining tasks, *in* 'KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining', ACM, New York, NY, USA, pp. 935–940.

Schlitter, N. & Falkowski, T. (2009), Mining the dynamics of music preferences from a social networking site, *in* 'Proceedings of the 2009 International Conference on Advances in Social Network Analysis and Mining', IEEE Computer Society, Washington, DC, USA, pp. 243–248.

Schlitter, N., Falkowski, T. & Lässig, J. (2011), Dengraph-HO: Density-based hierarchical community detection for explorative visual network analysis, *in* 'Research and Development in Intelligent Systems XXVIII Incorporating Applications and Innovations in Intelligent Systems XIX', Springer, London, pp. 283–296.

Shetty, J. & Adibi, J. (2004), 'The enron email dataset database schema and brief statistical report'. URL: http://www.cs.cmu.edu/~enron/

Xu, X., Yuruk, N., Feng, Z. & Schweiger, T. A. J. (2007), Scan: a structural clustering algorithm for networks, *in* P. Berkhin, R. Caruana & X. Wu, eds, 'KDD', ACM, pp. 824–833.

Yuruk, N., Mete, M., Xu, X. & Schweiger, T. A. J. (2007), A divisive hierarchical structural clustering algorithm for networks, *in* 'ICDM Workshops', IEEE Computer Society, pp. 441–448.

Yuruk, N., Mete, M., Xu, X. & Schweiger, T. A. J. (2009), Ahscan: Agglomerative hierarchical structural clustering algorithm for networks., *in* N. Memon & R. Alhajj, eds, 'ASONAM', IEEE Computer Society, pp. 72–77.

7. The authors

7.1. Nico Schlitter

Nico Schlitter is currently working as head of the bwLSDF project at the Steinbuch Centre for Computing, Karlsruhe Institute of Technology. Much of his current work is related to distributed storage solutions for the state of Baden-Wuerttemberg. He is also head of the multidisciplinary grid computing project www.distributeddatamining.org where he works in the fields of social network analysis and time series analysis. After receiving his degree in computer science from Chemnitz University of Technology in 2006, he has been working at University of Magdeburg in the field of RFID-based data analysis and at University of Applied Sciences Zittau/Görlitz in the area of simulation and optimization.

7.2. Tanja Falkowski

Tanja Falkowski is currently working as head of international relations at University of Göttingen. In 2009, she received her Ph.D. in computer science for her research on the analysis of community dynamics in social networks. She developed algorithms and methods to efficiently analyse temporal dynamics of group structures in social networks. Tanja Falkowski studied information systems at Technical University Braunschweig and wrote her diploma thesis at Haas School of Business, University of California at Berkeley.

7.3. Jörg Lässig

Jörg Lässig is a Full Professor at the Faculty of Electrical Engineering and Computer Science at the University of Applied Sciences Zittau/Görlitz. He holds degrees in computer science and computational physics and received a Ph.D. in computer science for his research on efficient algorithms and models for the generation and control of competence networks at Chemnitz University of Technology. He has been working in various research and industrial projects and is currently focusing on sustainable IT technologies. His research interests include efficient algorithms, bio-inspired methods, and green information systems.