Visual Analysis of Large Graphs Using (X,Y)-clustering and Hybrid Visualizations

Vladimir Batagelj, Franz J. Brandenburg, Walter Didimo, Giuseppe Liotta, Pietro Palladino, and Maurizio Patrignani

Abstract—Many different approaches have been proposed for the challenging problem of visually analyzing large networks. Clustering is one of the most promising. In this paper we propose a new clustering technique whose goal is that of producing both intra-cluster graphs and inter-cluster graph with desired topological properties. We formalize this concept in the (X,Y)-clustering framework, where Y is the class that defines the desired topological properties of intra-cluster graphs and X is the class that defines the desired topological properties of the inter-cluster graph. By exploiting this approach hybrid visualization tools can effectively combine different node-link and matrix-based representations, allowing users to interactively explore the graph by expansion/contraction of clusters without loosing their mental map. As a proof of concept, we describe the system VHYXY (**V**isual **Hy**brid (**X**,**Y**)-clustering) that implements our approach and we present the results of case studies to the visual analysis of social networks.

Index Terms—Large Graphs, Graph Clustering, Hybrid Visualization, Visual Analytics.

1 Introduction

Due to the huge amount of relational data handled by modern information systems in many application domains, the need of tools that support humans in the visual inspection of these data is rapidly growing. For this reason, the problem of visually analyzing large networks (also called *graphs*) has received an incredible amount of attention in the last decade, but the design of efficient and effective solutions remains a problem only partially solved. This problem poses two major challenges: (i) The most effective algorithms for automatic graph drawing are conceived to compute aesthetically readable layouts for relatively small and sparse graphs, and often they do not scale on graphs having hundreds or thousands of vertices [18], [36]; (ii) humans have limited capabilities to interpret big amounts of visual data at a whole; this implies that systems for the visual analysis of large graphs must be necessarily equipped with interaction methods that allow for an overview of the graph and for an incremental exploration of the data, while keeping the visual complexity of the diagram under control (see, e.g., [16], [51]).

The research in this paper began at Dagstuhl Seminar No. 08191 "Graph Drawing with Applications to Bioinformatics and Social Sciences", May 2008. A preliminary version of this work has been accepted for publication in the proceedings of the third IEEE Pacific Visualization Symposium, PacificVis 2010. The research in this work is supported in part by the MIUR project AlgoDEEP prot. 2008TFBWL4.

- *Vladimir Batagelj is with the Department of Mathematics, FMF, University of Ljubljana, Slovenia.*
- *Franz J. Brandenburg is with the Fakultät für Mathematik und Informatik, Universität Passau, Germany.*
- *Walter Didimo, Giuseppe Liotta, and Pietro Palladino are with the Dipartimento di Ingegneria Elettronica e dell'Informazione, Università degli Studi di Perugia, Italy.*
- *Maurizio Patrignani is with the Dipartimento di Informatica e Automazione, Università Roma Tre, Italy.*

1.1 Visualization of Large Graphs

Different solutions have been proposed so far to face the challenges above.

From one side, fast graph drawing algorithms have been invented to overcome the scalability limits of classical drawing techniques (see the work of Hachul and Jünger for a survey and an experimental study [32]). These algorithms however compute a flat layout of the whole graph that provides an overview of the entire graph structure but that is typically too complex and confused for detailed views. To enable viewers to see portions of primary interest presented in full detail while at the same time getting an overview of all context, a popular interaction strategy, known as *focus+context*, can be applied (see, e.g., [34] for a survey on focus+context approaches). A technique that works within this strategy is the well-known *fisheye view*, that produces a

distortion resembling the effect of using fisheye lenses on traditional layouts. However, since this distortion is independent of the layout technique, it may destroy the aesthetics governing the specific drawing algorithm (for example more crossings can be introduced during the interaction). An alternative focus+context method that takes into account some layout parameters is the well-known *hyperbolic view* (see, e.g., [38], [43]), but this method is mainly tailored for trees. An alternative zooming method, called *topological fisheye view*, has been proposed by Gansner, Koren and North [29]; it precomputes a hierarchy of coarsened graphs that are combined on-the-fly into renderings, with the level of detail dependent on distance from one or more foci.

From the other side, interactive visualization approaches have been proposed to explore a large graph incrementally. They can be classified into two main categories.

– Bottom-up approaches: The graph is visualized a piece per time. For example, the user can move on the graph with a "topological window" of limited size, and only the portion of the graph inside the topological window is shown at each time [25]. Differently, the user can start from a seminal node of the graph and recursively explore its neighbors, so enhancing the diagram step by step [15]. Bottom-up approaches are intuitive and efficient, but suffer from two main drawbacks: (a) The user does not have an overview of the graph; (b) preserving the user's mental map during the browsing is difficult.

– Top-down approaches: They provide an initial high-level view of the whole graph and then offer to the user the possibility of exploring specific portions of the graph to get more details, while trying to preserve her mental map. Most top-down approaches are based on *clustering*. The nodes of the graph are initially grouped into clusters, according to some clustering technique; then an *inter-cluster graph* is shown, where each cluster is represented as a single node and the relationships between the vertices of the graph are visually summarized as connections between the corresponding cluster nodes. The user can expand each single cluster node to inspect it inside, or she can contract a previously expanded cluster node, so to obtain many views of the graph at different levels of details. Examples of this approach can be found in [2], [3], [5], [6], [19], [26], [47], [50], [54].

Top-down approaches based on clustering are among the most promising strategies for the visual analysis of large graphs. However, if the input graph is not already clustered, an automatic clustering algorithm is needed. Most graph clustering algorithms look for the highly connected portions of the graph to compute clusters; also, there exist approaches that cluster the graph based on attribute information rather than on graph connectivity (see Section 2 for details). In both cases, they do not guarantee any property on the inter-cluster graph; as a consequence, computing a readable layout of such a high-level structure can be difficult and asking what is the best algorithm to draw it remains in general an unanswered question.

1.2 Contribution of the Paper

In this paper we introduce a novel methodology for the visual analysis of large networks within the top-down approach. The main ingredients of our methodology are: (i) A new framework to design automatic graph clustering algorithms. (ii) Decomposition of the graph into a cluster tree whereby both the clusters and inter-cluster graph have provable properties, and a hybrid visualization system based on this idea. More precisely:

– In Section 3 we define an algorithmic framework for computing two-level clustering algorithms that guarantee desired topological properties both for the intra-cluster graphs and for the inter-cluster graph. We call such a clustering an (X,Y)-*clustering*, where X and Y are the classes that define the topological properties of the inter-cluster graph and of the intra-cluster graphs, respectively. This allows us to exploit and combine different visualization algorithms for the X and the Y classes. (X,Y)-clustering can be recursively applied to generate a hierarchical clustering with many levels.

– In Section 4 we explore several types of (X,Y)-clustering, where the inter-cluster graph (X graph) is sparse and the intra-cluster graphs (Y graphs) are highly connected. We describe an $O((n+m)\log\Delta)$-time graph clustering algorithm such that each intra-cluster graph has vertex-degree at least k (for an appropriate k) and the inter-cluster graph is planar, where n, m, and Δ are the number of vertices, the number of edges, and the maximum vertex degree of the input graph, respectively. Our algorithm finds the maximum k that guarantees these properties. We then discuss a variant of this algorithm that makes it possible to have control on the density of the inter-cluster graph, instead of guaranteeing its planarity. The time complexity of this variant is $O((n+m)\Delta)$. We also prove that recognizing (X,Y)-graphs where X is a planar graph and Y is a k-clique (for a given k) is NP-hard.

– In Section 5 we describe a system called VHYXY (**V**isual **Hy**brid (**X**,**Y**)-clustering) that implements our polynomial-time clustering algorithms and some primitives for exploring the graph in the top-down

fashion. It makes it possible to combine effective node-link graph drawing algorithms to visualize the inter-cluster graph with matrix-based representations for dense intra-cluster graphs. As it will be explained throughout the paper, our system has some major differences with existing systems that adopt hybrid visualizations.

– To show the feasibility and the effectiveness of our approach we present in Section 6 some applications of VHyXY to the analysis of social networks. We discuss the scalability of our approach by illustrating some case studies and experimental results.

Section 2 discusses further works related with our research and Section 7 presents conclusions and open problems.

2 Related Works

The problem of computing clusterings that guarantee topological properties both for intra-cluster graphs and for the inter-cluster graph was previously studied by Brandenburg [13]. His model however requires that the clusters cover all the vertices of the graph and, as it will be clarified in Section 3, this requirement strongly limits its use in real-life applications.

From the point of view of the visualization paradigm, the use of hybrid visualizations has been explored in several works:

(i) Henry, Fekete and McGuffin describe an interesting system, NodeTrix, that combines node-link and matrix-based representations in the same layout [33]; however, differently from our approach, in NodeTrix clusters are not automatically computed, but they are defined manually by the user who interacts with the drawing. Also, as it will be explained later, in our matrix-based representation we compute an ordering of the vertices that reduces the number of crossings between the edges that connect vertices inside the cluster with vertices outside the cluster. This feature is not present in NodeTrix. A comparison between advantages of matrix-based representations versus node-link ones is described by Ghoniem, Fekete and Castagliola [30]. Techniques for searching orderings of vertices that highlight clusters in matrix-based representations of social networks are described by Doreian *et al.* [21].

(ii) Archambault, Munzner and Auber describe a multilevel graph drawing algorithm TopoLayout, which combines different node-link layouts for conveying distinct portions of a network on the basis of their topological properties [4]. Also, the same authors present two systems, Grouse [3] and GrouseFlocks [5]. The first system takes a graph along with a given cluster hierarchy (e.g., pre-computed by the TopoLayout algorithm) and allows users to interactively explore the clusters. To visualize the graph inside a cluster, the user can either choose in a range of possible layout algorithms (e.g., high-dimensional embedders, tree, circular, or force-directed algorithms), or she can ask the system to automatically apply one of these algorithms, according to the topological properties of the graph determined by TopoLayout; also, to save computational time, the drawings are stored for future visualizations. GrouseFlocks extends Grouse and makes it possible to automatically create cluster hierarchies also based on node attributes. The user can interactively explore and modify the hierarchies by means of some primitives.

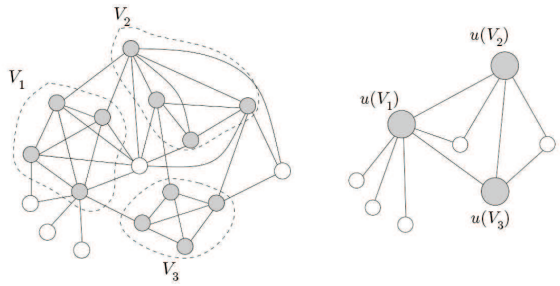
Differently from TopoLayout and GrouseFlocks, our clustering framework is not driven by some topological properties and node attributes of the graph, but it aims at guaranteeing desired structural properties of inter- and intra-cluster graphs. Also, our hybrid visualizations allows for a combination of node-link and matrix-based representations, providing some additional features with respect to existing systems, as previously observed. Finally, differently from the majority of the systems for the visual analysis of large graphs, we make primarily use of an engineered version of the topology-shape-metrics approach for orthogonal layouts to represent the graphs at a high-level of abstraction. The resulting drawing is crossing-free if the graph is planar or it contains a few edge crossings if the graph is relatively sparse.

(iii) Other interesting examples of hybrid visualizations combine node-link layouts with space-filling approaches [35], [52], [56]. Abello, Kobourov, and Yusufov describe a visualization approach based on fisheye views and treemaps [1]. Pattison, Vernik, and Phillips present a flexible system that combines treemaps and node-link layouts based on vertex and edge attributes [44]. Also, a hybrid visualization that combines orthogonal drawings with circular drawings for a graph with a given two-level clustering is described by Di Giacomo *et al.* [20].

3 (X,Y)-graphs and Clustering

Let $G=(V,E)$ be a graph, with vertex set V and edge set E. A *cluster* of G is a subset $V' \subseteq V$ of vertices of G. In this paper we always assume that a cluster consists of at least two vertices (i.e., we do not consider trivial clusters consisting of a single vertex). Given a cluster V' of G, we denote by $G[V']$ the subgraph of G induced by the vertices of V'.

Let $\mathscr{C}=\{V_1,V_2,\ldots,V_h\}$ be a set of disjoint clusters of G. The *graph of clusters of G with respect to* $\mathscr{C}$, denoted by $H(G,\mathscr{C})$, is the graph obtained from G by collapsing



(a) (b)

Fig. 1. (a) A graph G and a set of three clusters $\mathscr{C} = \{V_1, V_2, V_3\}$. (b) The graph of clusters $H(G, \mathscr{C})$.

each cluster of $\mathscr{C}$ into a single vertex and by replacing multiple edges with a single one. More formally, let $V_{\mathscr{C}} = V_1 \cup V_2 \cup \cdots \cup V_h$. Graph $H(G, \mathscr{C})$ consists of a vertex set V_H and of an edge set E_H defined as follows:

– For each vertex $v \in V \setminus V_{\mathscr{C}}$ there is a distinct vertex $u(v) \in V_H$ associated with v. For each cluster $V_i \in \mathscr{C}$ there is a distinct vertex $u(V_i) \in V_H$ associated with V_i $(1 \leq i \leq h)$.

– Let $(v, v') \in E$ be an edge of G. If $v, v' \in V \setminus V_{\mathscr{C}}$, there is the edge $(u(v), u(v')) \in E_H$. If $v \in V \setminus V_{\mathscr{C}}$ and $v' \in V_i$ $(1 \leq i \leq h)$, then there is the edge $(u(v), u(V_i)) \in E_H$. If $v \in V_i$, $v' \in V_j$, and $i \neq j$ $(1 \leq i, j \leq h)$, then there is the edge $(u(V_i), u(V_j)) \in E_H$.

Notice that, by definition, $H(G, \mathscr{C})$ does not contain self-loops or multiple edges. Some of the vertices of $H(G, \mathscr{C})$ correspond to clusters in $\mathscr{C}$ and others correspond to single vertices of G. For example, Fig. 1(a) depicts a graph G and a set $\mathscr{C}$ of three clusters (the vertices of each cluster are the gray vertices inside a closed dashed region); Fig. 1(b) shows that graph of clusters $H(G, \mathscr{C})$.

Let X and Y be two classes of graphs with certain properties and let $G = (V, E)$ be a graph. We say that G is an (X, Y)-*graph* if there exists a set $\mathscr{C} = \{V_1, V_2, \ldots, V_h\}$ of disjoint clusters of G such that: (i) Graph $H(G, \mathscr{C})$ belongs to X; (ii) subgraph $G[V_i]$ belongs to Y, for each $i = 1, \ldots, h$. In this case we also say that $\mathscr{C}$ is an (X, Y)-*clustering* of G. For example, the graph in Fig. 1(a) is a (planar, 4-clique)-graph, because each cluster of $\mathscr{C}$ induces a 4-clique and the graph of clusters is planar.

Observe that, although (X, Y)-graphs defines a two-level clustering, one can recursively apply this clustering technique to the graphs inside clusters in order to get a hierarchy with the desired number of levels.

We remark that the definition of (X, Y)-graph given in this paper generalizes that of *X-graph of Y-graphs* introduced by Brandenburg [13], in which it is required that $\mathscr{C}$ is a partition of V, i.e., $V = \bigsqcup_{i=1,\ldots,h} V_i$. The original model proposed by Brandenburg is interesting, but it turns out to be too restrictive for the design of efficient algorithms in several cases. It becomes equivalent to our approach if trivial clusters of a single vertex are allowed, which are not constrained to be Y-graphs. In the next section we describe polynomial-time algorithms for some types of (X, Y)-clustering that can be effectively used for a top-down visual analysis of large graphs.

4 Complexity and Algorithms for (X, Y)-clustering

Following the intuitive notion of cluster, the majority of the automatic graph clustering algorithms attempts to detect groups of vertices whose induced subgraphs are highly connected. However, these algorithms do not guarantee any property on the structure of the graph of clusters, and therefore a readable visualization of this graph could remain a difficult task. Looking for a specific (X, Y)-clustering makes it possible to know in advance the properties of the graph of clusters, other than the properties of the subgraphs induced by each cluster.

Several empirical studies compare different readability metrics for a graph drawing and show that the number of edges crossings often has the strongest impact on the readability of a diagram (see, e.g., [45], [46]). As a consequence, it has been widely accepted that one of the primary optimization tasks of a good graph drawing algorithm is minimizing crossings; a large body of literature has been devoted to this topic and many graph drawing algorithms have been designed to work effectively on sparse graphs and especially on planar graphs [18]. Furthermore, communities in a graph usually correspond to subsets of highly connected vertices.

Our idea is therefore to look for (X, Y)-clustering algorithms that guarantee highly connected graphs inside clusters and few edge crossings (possibly none) for the graph of clusters. With this spirit, in the next subsections we explore different combinations of (X, Y)-clustering that appear to be natural choices for the visualization of a large graph in the top-down approach. While for the first of these combinations we have a negative result in terms of tractability, the others can be algorithmically exploited in polynomial-time and can be effectively adopted for application purposes.

4.1 (Planar, k-clique)-clustering

A natural question is asking whether a large graph is a (planar, k-clique)-graph, i.e., whether it admits an (X, Y)-clustering such that the subgraph induced by each cluster

is a k-clique (for some fixed k) and the graph of clusters is planar.

Deciding whether a graph G is a planar graph of cliques is NP-hard, both in this and in Brandenburg's model, and it is NP-hard deciding whether a graph is a path (of any length $k \geq 2$) or a tree of paths [13], [37], [39], [48]. However, these results consider infinite types of Y-graphs, such as all cliques or all paths. Next we complement these results by considering a single type of graph for all clusters. Due to space limitations, we report here a sketch of the proof of Theorem 1. For a full proof refer to [7].

Theorem 1. *Let G be a graph and let $k \geq 5$ be a positive integer number. Deciding whether G is a (planar, k-clique)-graph is NP-hard.*

Sketch of Proof: We use a reduction from the NP-complete problem Planar-3SAT [41]. In such a problem you are asked to assign truth values to some Boolean variables in order to satisfy a 3SAT formula. In addition, each Boolean variable or clause of the 3SAT formula is mapped to a vertex of a planar graph, whose edges connect each clause-vertex with the three variable-vertices corresponding to the three literals of the clause.

The construction of the corresponding instance of the (planar, k-clique)-graph recognition problem is based on the fact that whenever two k-cliques share some vertices, they can not be simultaneously collapsed, but, since $k \geq 5$, one of the two has to be collapsed in order to obtain a planar graph. Hence, each variable-gadget is composed by a circular sequence of k-cliques, each sharing some vertices with the previous and the next k-clique, in such a way that either the k-cliques in odd positions or those in even positions have to be collapsed in order to obtain a planar graph, thus representing the two truth values of the corresponding variable.

The clause-gadgets are $(k+2)$-cliques sharing three nodes with the variable-gadgets of the corresponding literals, so that if the literal is true, the shared vertex is not involved in any contraction. Only if at least one of the literals is true, it is possible to find in the clause-gadget a k-clique that can be collapsed obtaining a planar graph, while if all literals are false, a $(k+2)$-clique that can not be collapsed yields a non-planar graph.

The graph composed by variable-gadgets and clause-gadgets can be constructed in polynomial time and is a (planar, k-clique)-graph if and only if the Boolean variables of the original Planar-3SAT instance can be assigned a truth value such that each clause has at least a true literal. □

Theorem 1 strongly motivates us to look for some other family Y of highly connected graphs. In the next subsection we exploit the notion of *k-core* of a graph and present a polynomial-time (X,Y)-clustering algorithm that is suitable for application purposes. We recall that the notion of k-core was introduced by Seidman in 1983 [49], and used in several application areas for the decomposition and the analysis of large graphs, especially in the field of social networks and biology (see, e.g., [8], [12], [22], [28], [31], [42], [55]).

4.2 (Planar, k-core component)-clustering

We investigate (X,Y)-graphs where X is the class of planar graphs and Y is the class of "k-core connected components" of a given graph. Before giving the formal definition of such (X,Y)-graphs, we recall basic definitions about k-cores (see also Fig. 2(a)).

Let $G=(V,E)$ be a graph and k be an integer number such that $0 \leq k < |V|$. Let W be a non-empty subset of V. The induced subgraph $G[W]$ of G is a *k-core* (or a *core of order k*) of G if all vertices of W have degree at least k in $G[W]$ and $G[W]$ is a maximal subgraph of G with this property. The *core number* of a vertex of G is the highest order of a core that contains this vertex. Due to the maximality property, if G contains a k-core, then it is unique. Also, the k-core of G is not necessarily connected; a connected component of a k-core of G will be simply called a *k-core component* of G.

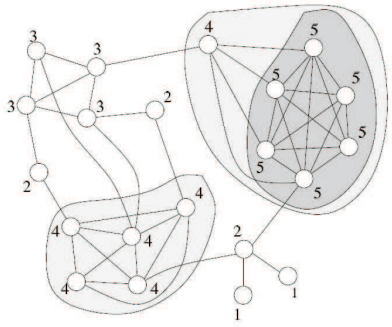
For any fixed k, denote by G_k the k-core of G (if it exists). The following property is an immediate consequence of the definition.

Property 1. *If G contains a k-core G_k, for $k \geq 1$, then G contains a $(k-1)$-core $G_{(k-1)}$ and $G_k \subseteq G_{(k-1)}$.*

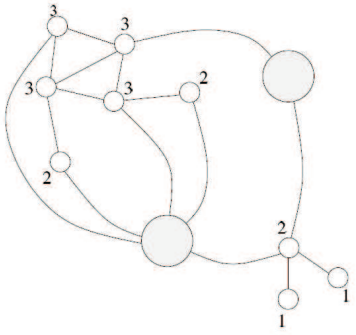
A graph G is a *(planar, k-core component)-graph* if it is an (X,Y)-graph where X is the class of planar graphs and Y is the class of connected k-cores of G. It is interesting to observe that every non-planar graph G is a (planar, 2-core component)-graph, as an immediate consequence of the following property.

Property 2. *If G is a non-planar graph then G has a 2-core. Also, if $\mathscr{C}$ is a set of clusters such that each cluster coincides with the vertex-set of a distinct 2-core component of G, then the graph of clusters $H(G,\mathscr{C})$ is a forest.*

Proof: If G is not planar, then G contains a cycle. Since all the vertices of this cycle have degree at least two, they all belong to a 2-core. Also, two distinct 2-core components cannot be joined by an edge or by a path in G, because otherwise they would form a unique 2-core component. Hence, they belong to two



(a)



(b)

Fig. 2. (a) A graph G where the vertices are labelled with their core numbers. The dark grey region indicates the 5-core of G; each of the two light grey regions indicates a 4-core component of G (their union is the 4-core of G). (b) The (planar) graph of clusters obtained by collapsing the 4-core component. Note that the graph of clusters is not planar if we collapse the 5-core; this implies that the (planar, 4-core component)-clustering of G depicted in the figure is a main planar core-clustering of G.

distinct connected components of G, and each connected component of $H(G,\mathscr{C})$ is a tree that contains at most one vertex representing a 2-core component of G. □

The next lemmas will be used as basic tools for designing an (X,Y)-clustering algorithm based on the concept of k-core components.

Lemma 1. *Let k be a positive integer number. If G is a (planar, k-core component)-graph then it is a (planar, $(k-1)$-core component)-graph.*

Proof: Let $G=(V,E)$ be a (planar, k-core component)-graph (for $k\geq 1$). This implies that there exists a clustering $\mathscr{C}=\{V_1,V_2,\ldots,V_h\}$ such that $H(G,\mathscr{C})$ is planar and $G[V_i]$ is a k-core component of G. By Property 1, each k-core component $G[V_i]$ is a subgraph of a $(k-1)$-core component $G[V_i']$, where $V_i\subseteq V_i'$. Denote by $\mathscr{C}'$ the set of $(k-1)$-core components containing all elements of $\mathscr{C}$. A graph isomorphic to $H(G,\mathscr{C}')$ can be obtained from $H(G,\mathscr{C})$ by performing an edge contraction for each edge that connects either two vertices of $V_i'\setminus V_i$ or a vertex of $V_i'\setminus V_i$ to $u(V_i)$ $(1\leq i\leq h)$, and then removing multiple edges and self-loops. Since edge-contractions do not increase the genus of a graph, it follows that graph $H(G,\mathscr{C}')$ is also planar. □

Lemma 2. *Let G be a (planar, k-core component)-graph and let $\mathscr{C}$ be the set of all k-core components of G. Then $H(G,\mathscr{C})$ is planar.*

Proof: By hypothesis, there exists a subset $\mathscr{C}'$ of $\mathscr{C}$ such that $H(G,\mathscr{C}')$ is planar. Denote by $V_1\ldots V_r$ the k-core components that belong to $\mathscr{C}\setminus\mathscr{C}'$. A graph isomorphic to $H(G,\mathscr{C})$ can be obtained from $H(G,\mathscr{C}')$ by performing an edge contraction for each edge that connects two vertices of V_i $(1\leq i\leq r)$, and then by removing multiple edges and self-loops. Since edge-contractions do not increase the genus of a graph, it follows that graph $H(G,\mathscr{C})$ is also planar. □

Observe that the higher values of k correspond to smaller clusters. Hence, it is interesting for applications to look for the maximum k such that $G=(V,E)$ is a (planar, k-core component)-graph, which corresponds to looking for the smallest clusters such that the graph of clusters is planar. A (planar, k-core component)-clustering of G is called a *main planar core-clustering* of G, if k is the maximum possible value for which G is a (planar, k-core component)-graph. Property 2 implies that if a (planar, k-core component)-clustering is a main planar core-clustering of G, then $k\geq 2$. For example, the (planar, 4-core component)-clustering shown in Fig. 2(a) is a main planar core-clustering of G, and Fig. 2(b) shows the corresponding graph of clusters. Using Lemmas 1 and 2, the following result can be proven.

Theorem 2. *Let G be a graph with n vertices, m edges, and maximum vertex degree Δ. There exists an algorithm that computes a main planar core-clustering of G in time $O((n+m)\log\Delta)$.*

Proof: An algorithm that finds a main planar core-clustering of G works in three main steps (refer to Algorithm 1):

Step 1. It computes the core number of every vertex of G and stores these numbers into an array (called *cores* in the algorithm's code) where each cell is associated with a distinct vertex of G (line 1). The computation of all vertex core numbers can be performed in $O(n+m)$ time by using an algorithm described by Batagelj and Zaveršnik [9].

Step 2. The algorithm uses the procedure Algorithm 2 to find the maximum k for which the input graph is a (planar, k-core component)-graph. This procedure per-

forms a binary search for finding the maximum possible value for k. If $first$ and $last$ denote the minimum and the maximum core number of a vertex of G, the binary search is performed on the interval $[first, last]$. At the generic step of the search, for a fixed value of k, we can apply the following strategy based on Lemmas 1 and 2: If G is a (planar, k-core component)-graph, then we can try to increase k by continuing the search on the half interval to the right of k, otherwise we have to decrease k and then concentrate on the half interval to the left of k. To decide whether or not G is a (planar, k-core component)-graph, we compute the set $\mathscr{C}$ of all k-core components of G (line 7) and the graph of clusters $H(G,\mathscr{C})$ (line 8), and then we run on $H(G,\mathscr{C})$ a planarity testing algorithm (line 9). Computing $\mathscr{C}$ and $H(G,C)$ from the vertex core numbers can be done in $O(n+m)$ time, and also the planarity testing can be executed in linear time (see, e.g., [18] for many references on planarity testing algorithms). Since the binary search consists of $O(\log\Delta)$ steps (the maximum vertex core number is at most Δ), we conclude that this step is performed in $O((n+m)\log\Delta)$ time.

Step 3. Once the maximum possible value for k has been found, the algorithm computes again the set $\mathscr{C}$ of all k-core components of G in linear time, and returns it[1].

It follows that the described algorithm correctly runs in $O((n+m)\log\Delta)$ time. □

Algorithm 1 MAXIMUMCORE

Input: A graph G
Output: A main planar core-clustering of G

```
1: cores[] ← COMPUTECORENUMBERS(G)
2: k ← SEARCHMAXIMUMK(G, cores)
3: 𝒞 ← COMPUTECORECOMPONENTS(G, k)
4: return 𝒞
```

4.3 Relaxing Planarity

Sometimes, looking for a main planar core-clustering can lead to few cores that are too big and/or too dense to be effectively explored. To overcome this problem, we suggest a variant of the (planar, k-core component)-clustering, which can be regarded as a relaxation of planarity. Namely, we suggest to keep under control the density of the graph of clusters, that is, the ratio between the number of edges and the number of vertices. Indeed, if the density of a graph is relatively small, one can expect that the graph admits a drawing that contains not so many crossings. We say that a graph is a *d-density graph* if its density is at most d. With the same strategy as Algorithm 1, we can find the maximum k for which a graph is a (d-density, k-core component) graph and to compute a corresponding (d-density, k-core component)-clustering; we call such a clustering a *main d-density core-clustering*. Note however that the density of the graph of clusters does not necessarily decrease when k decreases (see, e.g., Fig. 3). This implies that, in general, we cannot apply a binary search to find k as done by Algorithm 2, but we need to sequentially explore all possible values for k (which are $O(\Delta)$ in the worst case). For each fixed k, the graph of clusters is constructed in $O(n+m)$ time, and its density is computed in constant time. Hence, this theorem follows:

Algorithm 2 SEARCHMAXIMUMK(G, $cores[]$)

Input: A graph G and the vertex core numbers
Output: The maximum value of k for which G is a (planar, k-core component)-graph

```
1: first ← MIN(cores)
2: last ← MAX(cores)
3: k ← 0
4: maxK ← 0
5: while (first ≤ last) do
6:    k ← (first + last)/2
7:    𝒞 ← COMPUTECORECOMPONENTS(G, k)
8:    H ← COMPUTEGRAPHOFCLUSTERS(G, 𝒞)
9:    if (H is planar) then
10:      maxK ← k
11:      first ← k + 1 /* H is planar for all i < k */
12:      if (first ≥ last) then
13:         break /* maximum achieved */
14:      end if
15:   else
16:      last ← k − 1
17:   end if
18: end while
19: return maxK
```

Theorem 3. *Let G be a graph with n vertices, m edges, and maximum vertex degree Δ. Also, let $d > 0$ be a real number. There exists an algorithm that computes a main d-density core-clustering in $O((n+m)\Delta)$ time if such a clustering exists.*

From a theoretical point of view, computing a main d-density core-clustering is computationally more expensive than computing a main planar core-clustering. However, from a practical point of view, Δ is usually

1. We observe that the second step and the third step can be slightly improved in order to compute the final set $\mathscr{C}$ only once.

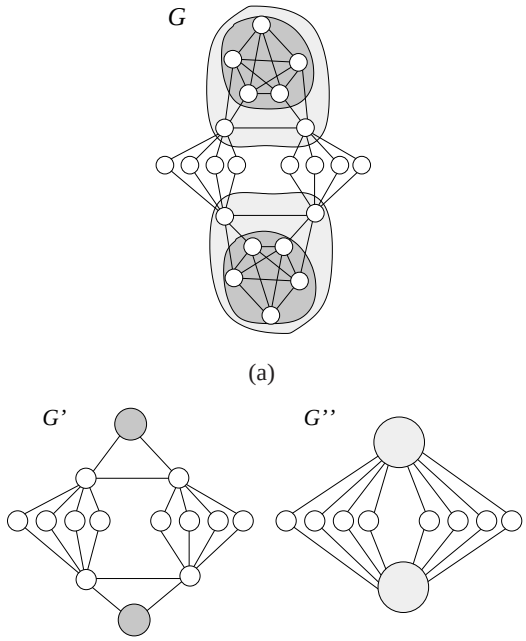


Fig. 3. (a) A graph G. The dark grey regions denote the 4-core components; the ligh grey regions denote the 3-core components. The graph of clusters G' and G'' obtained by collapsing the 4-core components and the 3-core components, respectively. G' has density 1.57, while G'' has density 1.6. The difference between the two densities can be arbitrarily augmented by increasing the number of vertices of degree two.

much smaller than n. Also, for any fixed k, once the graph of clusters has been created, the computation of its density is less expensive than running a planarity testing algorithm.

5 The System VHyXY: (X,Y)-clustering and Hybrid Visualizations

As a proof of concept of our approach, we developed a system, called VHyXY, which implements the algorithms described in Subsections 4.2 and 4.3. It allows users to visually analyze a graph by automatic clustering and interactive exploration of the drawing. Similar to other existing systems (see, e.g., [2], [3], [5], [6], [19], [26], [47], [50], [54]), VHyXY adopts a top-down strategy: (i) Initially the user is presented with a drawing of the graph of clusters, where all clusters are collapsed (see, e.g., Fig. 4(a)); we remark that, unlike other existing systems, our approach guarantees desired properties of the graph of clusters. (ii) then, the user can interactively explore the drawing by expanding/collapsing clusters and navigating through them. At each exploration action the drawing is updated accordingly. Fig. 4(b) shows how the

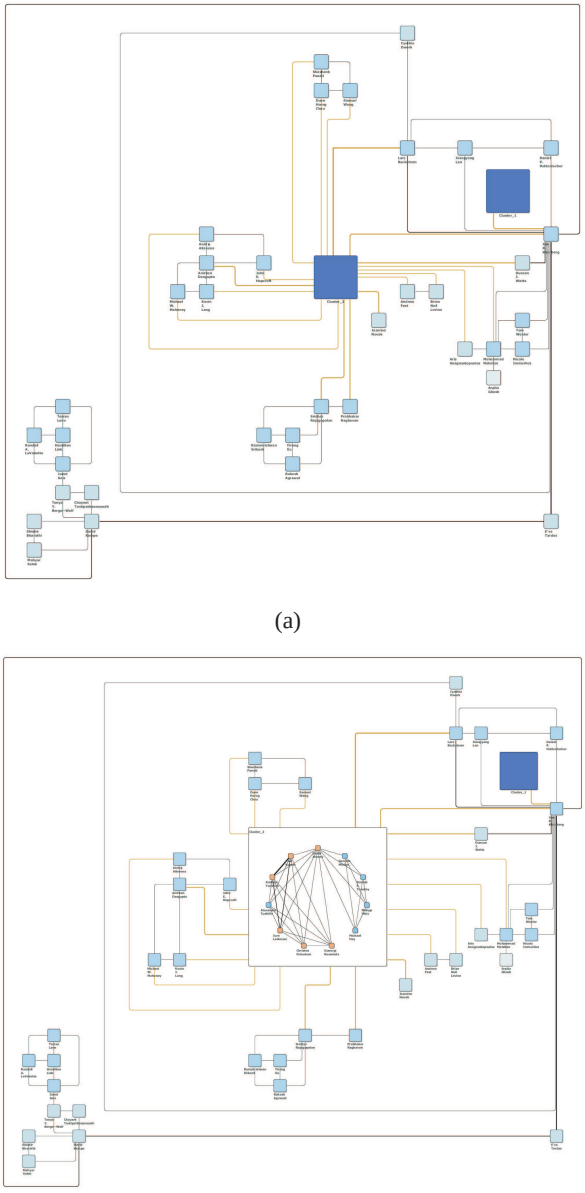


Fig. 4. (a) A view of the graph of clusters, where all clusters are collapsed. Bigger vertices represent clusters. The different scale of color for each vertex is used to reflect its core number: Darker colors correspond to higher core numbers. (b) A new drawing of the graph where one cluster is expanded; the graph induced by the cluster is drawn with a circular layout algorithm.

drawing of Fig. 4(a) is modified when the user expands one of its clusters.

The graph of clusters computed by Algorithm 1 is always planar; hence, to visually convey this graph we can potentially apply any of the drawing algorithms specifically described in the literature for planar graphs (see, e.g., [18]). Nevertheless, the drawing algorithm

should also support constraints that facilitate in the preservation of the mental map during the browsing of the user. Namely, according to our convention, the drawing algorithm must deal with vertices of different sizes and the drawing of the graph of clusters should not change too much when a cluster is collapsed/expanded (that is, when the size of a vertex is modified). For this reason we draw the graph of clusters within the *topology-shape-metrics* approach for orthogonal drawings (that is, drawings in which all edges are chains of horizontal and vertical segments). This approach, originally introduced by Tamassia [53], has received considerable attention in the graph drawing literature, and several variants have been studied to deal with vertices of any degree and to handle different drawing constraints using network flow algorithms. VHYXY implements an engineered version of the algorithm described in [17], which makes it possible to fix the dimensions of each single vertex. In our implementation, we represent each cluster as a box whose dimensions are proportional to the number of vertices inside it. To improve the aspect ratio of the drawing, our algorithm equally distributes the edges incident to a vertex of high degree on the four sides of the vertex. Also, if a vertex v is adjacent to many one-degree vertices, the drawing algorithm may decide to place these vertices inside a dummy invisible box close to v, in order to reduce the drawing area. When the user expands a cluster, the algorithm creates extra room in the drawing to show the subgraph inside the cluster, and the shape of the drawing remains the same for user's mental map preservation (the shape defines the sequence of bends along the edges and the angles formed by the edges around each vertex). See Fig. 4(a) and Fig. 4(b).

VHYXY also allows the user to visualize the graph of clusters by running a variant of the spring embedder algorithm of Eades [24], where vertices are modelled as electrically charged particles and edges are modelled as springs (each edge is drawn as a straight-line segment). We assign to each vertex representing a cluster region a charge that is proportional to its desired size, so that a big vertex has a large repulsive force and staves neighbors off. Note however that this algorithm may lead to drawings that contain edge crossings even if the graph is planar. More sophisticated force-directed techniques that make it possible to start from a given topology and compute a drawing that preserves the number of edge-crossings are described in [11], [23].

To visualize the graph inside a cluster region VHYXY can still apply either an orthogonal drawing algorithm or a spring embedder one, when the cluster contains other subclusters. Otherwise, the graph in the cluster region is a k-core component that cannot

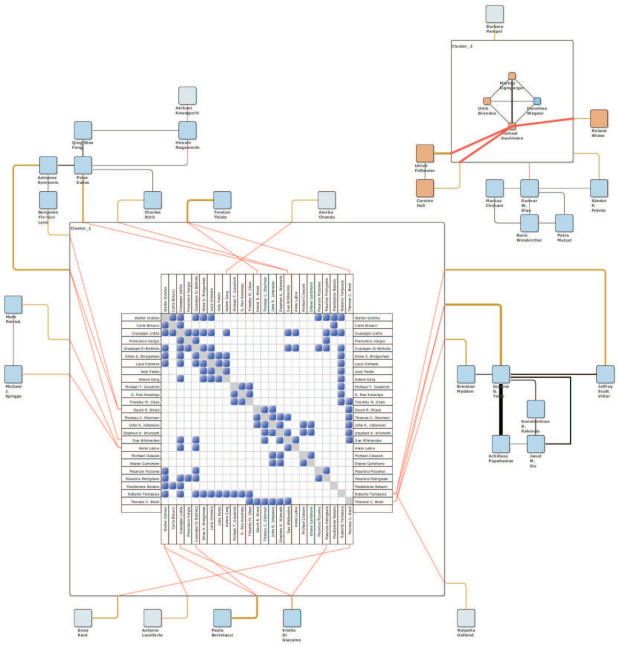


Fig. 5. A network with two expanded clusters; the graphs inside cluster regions are shown with different layouts.

further decomposed. In this case the user can choose the preferred drawing algorithm to represent the graph. As node-link representations, VHYXY currently implements the algorithms described in [10], [17], [24], [27] for the circular, orthogonal, force-directed, and layered drawing conventions, respectively. If the density of the graph is high, a matrix-based representation algorithm could be more effective and efficient than a node-link one, as also observed in [30]. Fig. 5 shows a network where two clusters are expanded and are visually conveyed with different drawing conventions.

To avoid confusion and to limit the visual complexity of the drawing, when the graph inside a cluster region is drawn with a node-link representation edges connecting vertices inside the clusters with vertices outside the cluster are shown only on demand; we call these edges *inside-outside links*. The system uses a (light) red color to highlight those vertices that are incident to some inside-outside links: If the user selects a red vertex its inside-outside links are displayed by straight-lines.

In the matrix-based representation the graph is represented as a square matrix where for each node $v_i \in G$ there is a row and a column with index i and the position of the matrix with indices (i, j) is filled if and only if there exists the (undirected) edge (v_i, v_j) in G (see Fig. 6(a)). Note that, the position (i, i) represents a self-loop. VHYXY makes it possible to show all the inside-outside links in such a way that the number of crossings

between these links is automatically minimized using a heuristic algorithm. Namely, the ordering of the vertices in each column is computed by applying a two-layer-crossings minimization heuristic to the connections on the upper and lower sides of the matrix, and similarly for the rows on the left and right sides. Then the system chooses the ordering that minimizes the sum of crossings (see for example Fig. 6(b)). As far as we know, there is no previous systems that apply this approach for hybrid visualizations.

In addition to the main functionalities described above, VHYXY offers to the user several additional operations that can be performed to make the visual analysis more effective. Weights can be assigned to vertices and edges depending on the specific application, and the user can filter some graphical objects based on these weights. Filtering helps to reduce the visual complexity of the layout when there is an overflow of information. Also, the user can edit the drawing at any time, by dragging vertices or entire portions of it, or by removing specific objects in which she is no longer interested. Graphs and drawings can be imported/exported in the GraphML standard file format [14], which guarantees the possibility of analyzing graphs coming from a large range of sources.

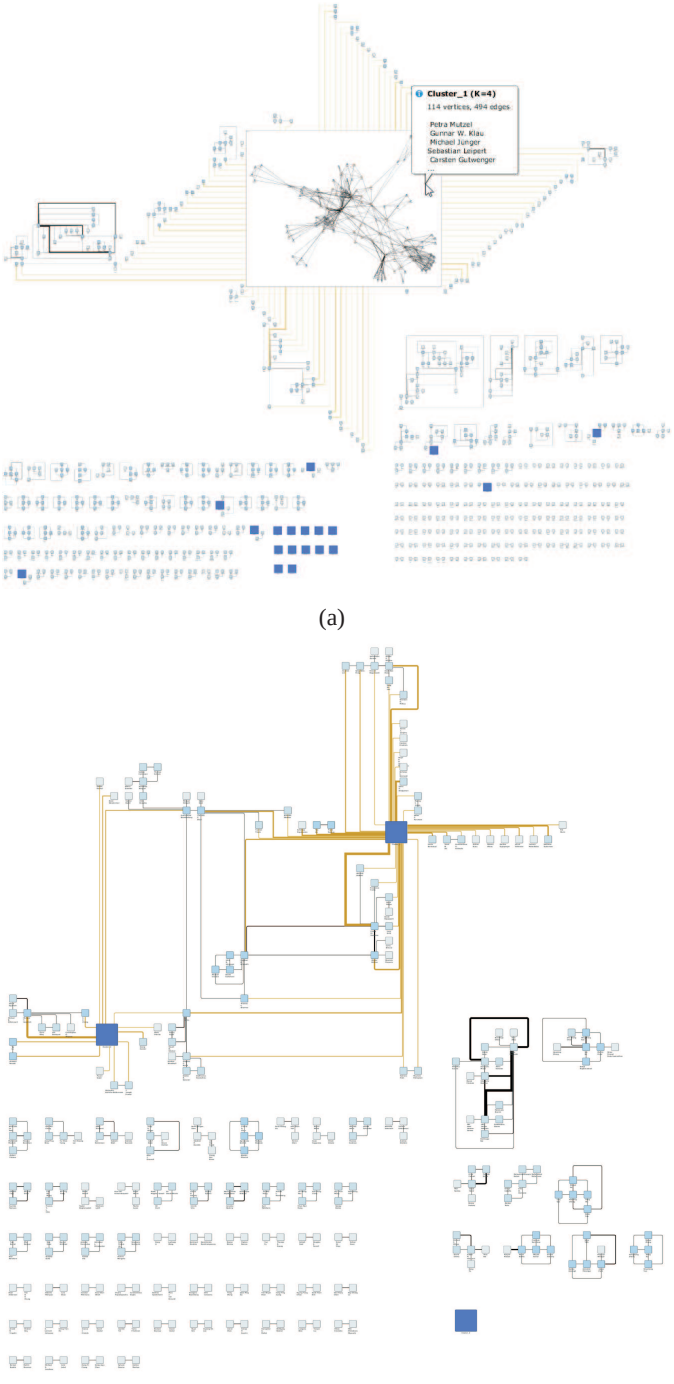
6 Application Examples.

We present an application to the visual analysis of co-authorship networks in computer science. It is based on the popular DBLP dataset [40], which contains most data about papers in computer science and holds information like authors, conference/journals, and publication year. We focus on this task: *Giving a certain topic t, discover the authors who published papers about t and their relationships*. The topic t can be specified by means of an advanced query expression, consisting of words and boolean operators (AND/OR). The user can also apply filters on the time interval and on the types of publications.

The graphical interface is a customized version of VHYXY. The user is presented with a co-authorship network that is automatically clustered by using the (planar, k-core component)-clustering algorithm available in VHYXY, and can perform all the exploration actions described in the previous sections. The network is constructed as follows: (i) There is a vertex associated with each author, and the label of a vertex is the name of the corresponding author. (ii) There is an edge between two vertices if the corresponding authors wrote at least one paper together in the specified time interval; each edge has a weight that corresponds to the number of papers associated with the edge; the label of the edge is the set of titles and years of the corresponding publications.

6.1 Case Studies

We describe two examples of analysis of co-authorship networks for queries in the area of graph visualization.

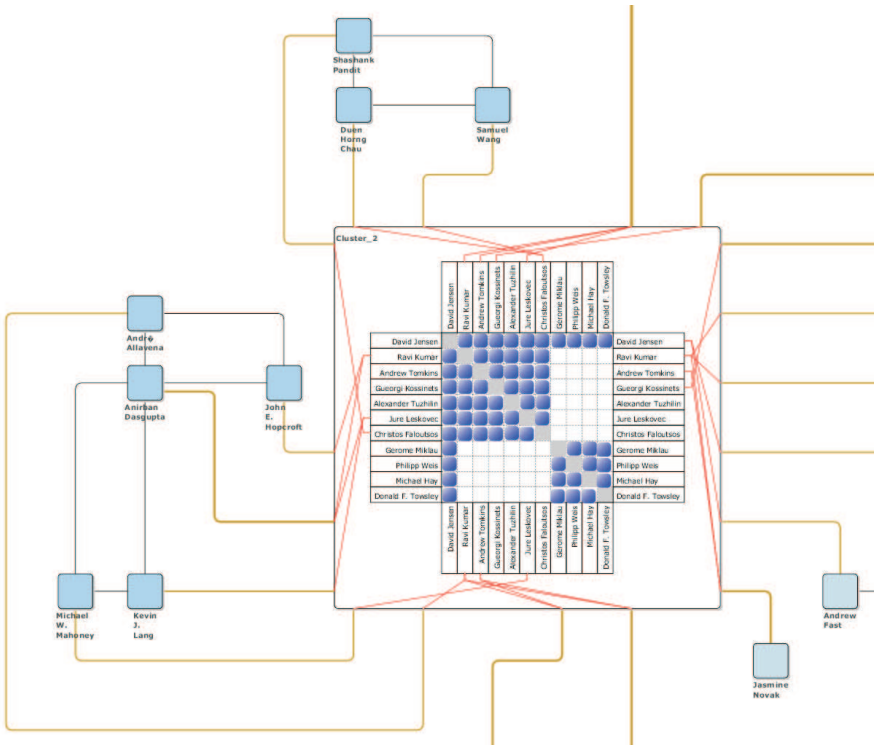


(a)

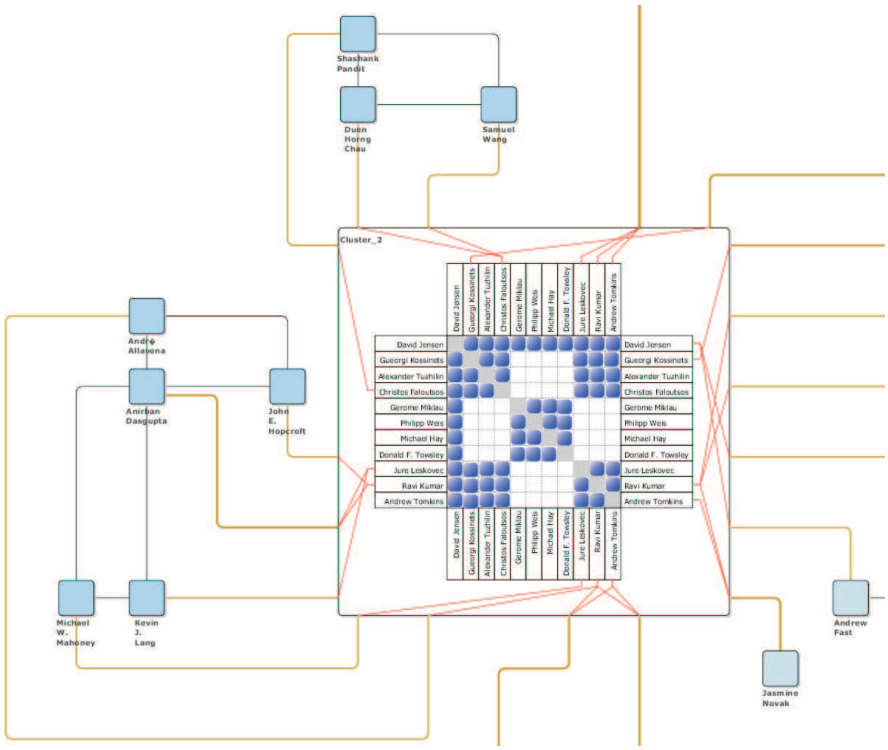
(b)

Fig. 7. (a) A drawing of the graph of clusters for the query "*graph* AND *(drawing* OR *embedding)*". (b) A drawing of the graph of clusters after the edges with weight smaller than 2 have been removed.

In the first case study, we perform the query "*graph* AND *(drawing* OR *embedding)*", so to match a large



(a)



(b)

Fig. 6. (a) An expanded cluster drawn with a matrix-based representation. Connections to outside are represented by red links. Gray squares denotes the cells of the main diagonal. (b) A different matrix-based representation of the same cluster, where rows and columns have been re-ordered to minimize crossings between red links.

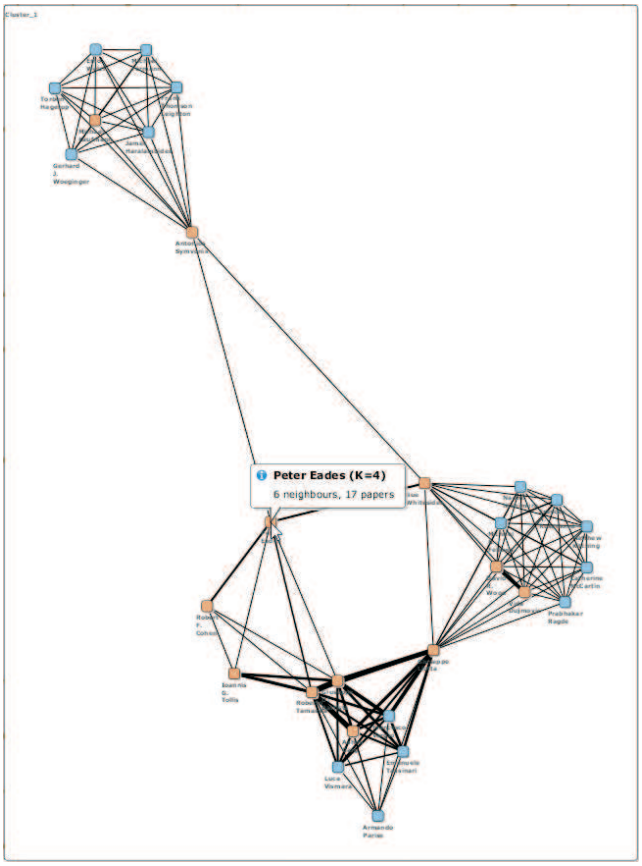


Fig. 8. Focusing on a core component with 28 vertices and 113 edges.

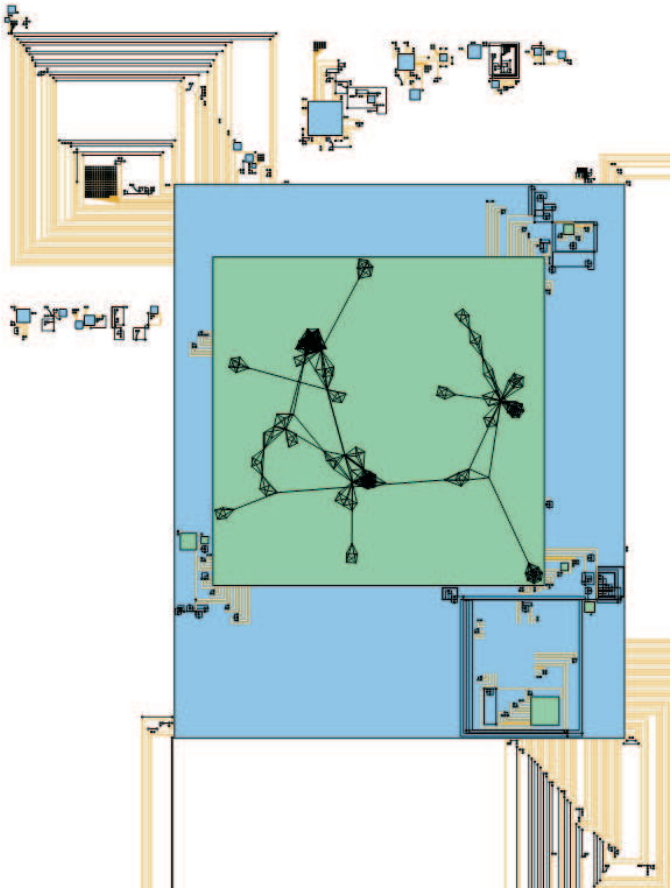


Fig. 9. Portion of a clustered co-authorship network with $28,758$ vertices and $59,161$ edges, where edges having weight one have been filtered out.

body of literature in graph drawing; we specify the interval time 1960–2009. The system returns a network with 1,056 papers and 1,717 edges; a clustering and a drawing are then computed in about 3 seconds on a PC with a 1.6 GHz Intel Centrino Duo processor and 2GB of RAM. According to our clustering algorithm, this graph contains a big core component of 114 vertices and 494 edges (Fig. 7(a)), which is rather confused due to its high density. Since many edges correspond to occasional cooperations among authors, we can discard such edges from the whole network by filtering on edge weights. For example, to have a clearer idea of the graph communities, we can ask the system to consider only those connections with weight larger than 2 and to perform a new clustering. The new network has 496 edges and the biggest connected component has now two core connected components, one containing 18 vertices and 58 edges and the other containing 28 vertices and 113 edges (see Fig. 7(b)). Looking inside these clusters one can clearly discover the corresponding communities. For example, in Fig. 8 we have expanded and zoomed on the cluster with 28 vertices and 113 edges, where author *Peter Eades* appears to be a central node of this subnetwork.

In the second case study, we perform the query "*visual*", which gives rise to a much larger graph, consisting of $28,758$ vertices and $59,161$ edges. Vertex core numbers vary from 1 to 40. For $k = 3$ we get the main planar core-clustering and VHYXYcomputes it in about 16 seconds. The resulting graph of clusters has several connected components, which represent well distinct communities concerned with the query. The largest connected component contains 2,681 vertices and 3,411 edges; 24 vertices represent clusters. One of the clusters is much bigger than the others and contains 7,053 vertices and 23,843 edges. If we recursively apply on the biggest cluster the main d-density core-clustering algorithm with $d = 2$, we detect a clique of 41 authors after four recursion steps. It identifies a big joint work focused on a global cooperative effort between visualization and collaboration researchers to build a persistent virtual distributed facility. With the system we can also discover that only 13 of these 41 authors worked together in some other papers, mainly concerned

with Virtual Reality. If we try to filter out from the whole graph all the edges whose weight is equal to one, the remaining subgraph has 8,908 edges and 6,570 non-isolated vertices. The main planar core-clustering is computed in 3 seconds and consists of 305 clusters. The drawing of the resulting graph of clusters is computed in 26 seconds. Fig. 9 depicts a portion of the graph of clusters. In the figure the biggest cluster (the blue big box) is expanded and a 2-density core-clustering algorithm is applied on the graph inside it, so getting a second level in the cluster hierarchy (the green boxes represent second-level clusters). The figure also shows the subgraph inside one of the second-level clusters, drawn with our force-directed algorithm. We found that from the drawing it is possible to immediately recognize different groups of authors covering different topics concerned with the query, among which visualization in Bioinformatics, Medicine, Physics, and Robotics.

6.2 Further experiments

The case studies suggest that our clustering strategy performs well both in terms of time complexity and in terms of relevance of the clusters for graphs with some thousands vertices and edges. In order to better investigate the scalability of our clustering approach, we performed some other experiments on a denser social network coming from a different application domain. We applied our main planar core-clustering algorithm on a social network constructed from the popular Internet Movie Database. The vertices of the network are actors and two actors are connected by an edge of weight k if they appeared together in k movies. Selecting about $5,000$ movies with the highest rank, we got a network consisting of $39,924$ vertices and $543,430$ edges. The vertex core numbers vary from 1 to 184 and the main planar core-clustering was found for $k = 4$. The main planar core-clustering algorithm ran in about 5 minutes and found 618 first-level clusters. The resulting graph of clusters contains $2,381$ vertices and $2,228$ edges. This graph consists of several different connected components, two of them containing the majority of the vertices in the network, grouped into two big clusters. In order to get smaller clusters, we also tried to relax planarity and asked the system to compute a main d-density core-clustering with $d = 3$. The resulting graph of clusters consists of $7,268$ vertices and $18,284$ edges; it has 311 clusters corresponding to 9-core components. These clusters are much easier to explore than the previous ones, but the clustering process was much more expensive; it required about 40 minutes. Also, due to its density, the graph of cluster is much less readable if drawn as an orthogonal layout; applying a force-directed algorithm is better in this case.

7 Conclusions and Open Problems

We described a new methodology for the visual analysis of large networks, based on a novel clustering framework that guarantees desired topological properties both for the inter- and the intra-cluster graphs. Within this framework we designed polynomial-time clustering algorithms whose performances are satisfactory for graphs with several thousands vertices and edges, especially when the density of the graphs is relatively small. We also presented a system that implements our methodology and that allows for a visual exploration of large graphs by using hybrid visualizations.

In the near future we plan to investigate other clustering algorithms that work within our framework and we plan to experiment with our methodology on further application domains.

Acknowledgements

We thank the anonymous referees for their valuable comments.

References

[1] J. Abello, S. Kobourov, and R. Yusufov. Visualizing large graphs with compound-fisheye views and treemaps. In *GD 2004*, volume 3383 of *LNCS*, pages 431–441, 2004.

[2] J. Abello, F. van Ham, and N. Krishnan. Ask-graphview: A large scale graph visualization system. *IEEE Trans. Vis. Comput. Graph.*, 12(5):669–676, 2006.

[3] D. Archambault, T. Munzner, and D. Auber. Grouse: Feature-based, steerable graph hierarchy exploration. In *EuroVis*, pages 67–74, 2007.

[4] D. Archambault, T. Munzner, and D. Auber. Topolayout: Multilevel graph layout by topological features. *IEEE Trans. Vis. Comput. Graph.*, 13(2):305–317, 2007.

[5] D. Archambault, T. Munzner, and D. Auber. Grouseflocks: Steerable exploration of graph hierarchy space. *IEEE Trans. Vis. Comput. Graph.*, 14(4):900–913, 2008.

[6] D. Auber, Y. Chiricota, F. Jourdan, and G. Melançon. Multiscale visualization of small world networks. In *IEEE INFOVIS 2003*, pages 75–81, 2003.

[7] V. Batagelj, F. J. Brandenburg, W. Didimo, G. Liotta, P. Palladino, and M. Patrignani. Collapsing k-cliques to get a planar graph is NP-complete. Tech. Rep. RT-DIA-174-2010, Dept. of Computer Science and Automation, Roma Tre University, 2010.

[8] V. Batagelj, A. Mrvar, and M. Zaveršnik. Partitioning approach to visualization of large networks. In *GD 1999*, volume 1731 of *LNCS*, pages 90–97, 1999.

[9] V. Batagelj and M. Zaveršnik. An O(m) algorithm for cores decomposition of networks. *CoRR*, cs.DS/0310049, 2003.

[10] M. Baur and U. Brandes. Crossing reduction in circular layouts. In *WG 2004*, volume 3353 of *LNCS*, pages 332–343, 2004.

[11] F. Bertault. A force-directed algorithm that preserves edge crossing properties. In *Graph Drawing*, volume 1731 of *Lecture Notes in Computer Science*, pages 351–358. Springer, 1999.

[12] B. Bollobás. The evolution of sparse graphs. In A. Press, editor, *Graph Theory and Combinatorics*, pages 35–57, 1984.

[13] F.-J. Brandenburg. Graph clustering 1: Circles of cliques. In *GD 1997*, volume 1353 of *LNCS*, pages 158–168, 1997.

[14] U. Brandes, M. Eiglsperger, I. Herman, M. Himsolt, and M. Marshall. Graphml progress report: Structural layer proposal, 2002.

[15] A. Carmignani, G. D. Battista, W. Didimo, F. Matera, and M. Pizzonia. Visualization of the high level structure of the internet with HERMES. *J. Graph Algorithms Appl.*, 6(3):281–311, 2002.

[16] B. Craft and P. Cairns. Beyond guidelines: What can we learn from the visual information seeking mantra? In *IEEE 9th International Conference on Information Visualization*, pages 110–118, 2005.

[17] G. Di Battista, W. Didimo, M. Patrignani, and M. Pizzonia. Orthogonal and quasi-upward drawings with vertices of prescribed sizes. In *GD 1999*, volume 1731 of *LNCS*, pages 297–310, 1999.

[18] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing*. Prentice Hall, Upper Saddle River, NJ, 1999.

[19] E. Di Giacomo, W. Didimo, L. Grilli, and G. Liotta. Graph visualization techniques for web clustering engines. *IEEE Trans. Vis. Comput. Graph.*, 13(2):294–304, 2007.

[20] E. Di Giacomo, W. Didimo, G. Liotta, and P. Palladino. Visual analysis of one-to-many matched graphs. *J. Graph Algorithms Appl. - special issues of GD 2008*, 2009. to appear.

[21] P. Doreian, V. Batagelj, and A. Ferligoj. *Generalized Blockmodeling*. Cambridge University Press, 2005.

[22] S. N. Dorogovtsev, A. V. Goltsev, and J. F. F. Mendes. k-core organization of complex networks. *CoRR*, abs/cond-mat/0509102, 2005.

[23] T. Dwyer, K. Marriott, and M. Wybrow. Topology preserving constrained graph layout. In *Graph Drawing*, volume 5417 of *Lecture Notes in Computer Science*, pages 230–241. Springer, 2008.

[24] P. Eades. A heuristic for graph drawing. In *Congressus Numerantium*, volume 42, pages 149–160, 1984.

[25] P. Eades, R. F. Cohen, and M. L. Huang. Online animated graph drawing for web navigation. In *GD 1997*, volume 1353 of *LNCS*, pages 330–335, 1997.

[26] P. Eades and M. L. Huang. Navigating clustered graphs using force-directed methods. *J. Graph Algorithms Appl.*, 4(3):157–181, 2000.

[27] M. Eiglsperger, M. Siebenhaller, and M. Kaufmann. An efficient implementation of Sugiyama's algorithm for layered graph drawing. In *GD 2004*, volume 3383 of *LNCS*, pages 155–166, 2004.

[28] M. Gaertler and M. Patrignani. Dynamic Analysis of the Autonomous System Graph. In *IPS 2004, International Workshop on Inter-domain Performance and Simulation, Budapest, Hungary*, pages 13–24, 2004.

[29] E. R. Gansner, Y. Koren, and S. C. North. Topological fisheye views for visualizing large graphs. *IEEE Trans. Vis. Comput. Graph.*, 11(4):457–468, 2005.

[30] M. Ghoniem, J.-D. Fekete, and P. Castagliola. On the readability of graphs using node-link and matrix-based representations: A controlled experiment and statistical analysis. *Information Visualization*, 4(2):114–135, 2005.

[31] A. V. Goltsev, S. N. Dorogovtsev, and J. F. F. Mendes. k-core (bootstrap) percolation on complex networks: Critical phenomena and nonlocal effects. *CoRR*, abs/cond-mat/0602611, 2006.

[32] S. Hachul and M. Jünger. Large-graph layout algorithms at work: An experimental study. *J. Graph Algorithms Appl.*, 11(2):345–369, 2007.

[33] N. Henry, J.-D. Fekete, and M. J. McGuffin. Nodetrix: a hybrid visualization of social networks. *IEEE Trans. Vis. Comput. Graph.*, 13(6):1302–1309, 2007.

[34] I. Herman, G. Melançon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Trans. Vis. Comput. Graph.*, 6(1):24–43, 2000.

[35] T. Itoh, C. Muelder, K. Ma, and J. Sese. A hybrid space-filling and force-directed layout method for visualizing multiple-category graphs. In *IEEE PacificVis 2009*, pages 121–128, 2009.

[36] M. Kaufmann and D. Wagner, editors. *Drawing Graphs*. Springer Verlag, 2001.

[37] J. Kratochvíl. String graphs II - recognizing string graphs is NP-hard. *J. Comb. Theory, Ser. B*, 52(1):67–78, 1991.

[38] J. Lamping and R. Rao. The hyperbolic browser: A focus + context technique for visualizing large hierarchies. *J. Vis. Lang. Comput.*, 7(1):33–55, 1996.

[39] H. Le, V. Le, and H. Müller. Splitting a graph into disjoint induced paths or cycles. *Disc. Appl. Math.*, 131:199–212, 2003.

[40] M. Ley. The DBLP computer science bibliography. http://www.informatik.uni-trier.de/∼ley/db/.

[41] D. Lichtenstein. Planar formulae and their uses. *SIAM J. Comput.*, 11:185–225, 1982.

[42] T. Łuczak. Size and connectivity of the k-core of a random graph. *Discrete Mathematics*, 91(1):61–68, 1991.

[43] T. Munzner. H3: laying out large directed graphs in 3D hyperbolic space. In *INFOVIS*, pages 2–10. IEEE Computer Society, 1997.

[44] T. Pattison, R. Vernik, and M. Phillips. Information visualisation using composable layouts and visual sets. In *InVis.au*, volume 9 of *CRPIT*, pages 1–10, 2001.

[45] H. C. Purchase. Which aesthetic has the greatest effect on human understanding? In *Graph Drawing (Proc. GD'97)*, volume 1353 of *Lecture Notes Comput. Sci.*, pages 248–261, 1998.

[46] H. C. Purchase, D. A. Carrington, and J.-A. Allder. Empirical evaluation of aesthetics-based graph layout. *Empirical Software Engineering*, 7(3):233–255, 2002.

[47] D. Schaffer, Z. Zuo, S. Greenberg, L. Bartram, J. Dill, S. Dubs, and M. Roseman. Navigating hierarchically clustered networks through fisheye and full-zoom methods. *ACM Trans. Comput.-Hum. Interact.*, 3(2):162–188, 1996.

[48] F. Schreiber and K. Skodinis. NP-completeness of some tree-clustering problems. In *GD 1998*, volume 1547 of *LNCS*, pages 288–301, 1998.

[49] S. B. Seidman. Network structure and minimum degree. *Social Networks*, 5:269–287, 1983.

[50] L. Shi, N. Cao, S. Liu, W. Qian, L. Tan, G. Wang, J. Sun, and C. Lin. HiMap: Adaptive visualization of large-scale online social networks. In *IEEE PacificVis 2009*, pages 41–48, 2009.

[51] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Visual Languages Conference*, pages 336–343, 1996.

[52] G. Sindre, B. Gulla, and H. G. Jokstad. Onion graphs: aesthetics and layout. In *IEEE Symposium on Visual Languages*, pages 287–291, 1993.

[53] R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comp.*, 16(3):421–444, 1987.

[54] F. van Ham and J. J. van Wijk. Interactive visualization of small world graphs. In *IEEE INFOVIS 2004*, pages 199–206, 2004.

[55] S. Wuchty and E. Almaas. Peeling the Yeast protein network. *Proteomics. 2005 Feb;5(2):444-9.*, 5(2):444–449, 2005.

[56] S. Zhao, M. J. McGuffin, and M. H. Chignell. Elastic hierarchies: Combining treemaps and node-link diagrams. In *IEEE INFOVIS 2005*, page 8, 2005.