CGV – An Interactive Graph Visualization System

Christian Tominski[a,*], James Abello[b], Heidrun Schumann[a]

[a] *University of Rostock, Albert-Einstein-Straße 21, D-18059 Rostock, Germany*
[b] *DIMACS, Rutgers University, 96 Frelinghuysen Road, Piscataway, NJ 08854-8018, USA*

Abstract

Previous work on graph visualization has yielded a wealth of efficient graph analysis algorithms and expressive visual mappings. To support the visual exploration of graph structures, a high degree of interactivity is required as well.

We present a fully implemented graph visualization system, called CGV (**C**oordinated **G**raph **V**isualization), whose particular emphasis is on interaction. The system incorporates several interactive views that address different aspects of graph visualization. To support different visualization tasks, view ensembles can be created dynamically with the help of a flexible docking framework. Several novel techniques, including enhanced dynamic filtering, graph lenses, and edge-based navigation are presented. The main graph canvas interactions are augmented with several visual cues, among which the infinite grid and the radar view are novel. CGV provides a history mechanism that allows for undo/redo of interaction.

CGV is a general system with potential application in many scenarios. It has been designed as a dual-use system that can run as a stand-alone application or as an applet in a web browser. CGV has been used to evaluate graph clustering results, to navigate topological structures of neuronal systems, and to perform analysis of some time-varying graphs.

Key words: Graph visualization, interaction, multiple coordinated views
2000 MSC: 68U05, 68U35

1. Introduction

In many application fields, visualization techniques have been recognized as a potential tool to order, manage, and understand large data. Interaction has long since been an important aspect in information visualization [54, 13]. More recent work confirm its relevance [52, 68, 72]. It is becoming apparent as stated in [58] that: *"Visual representations alone cannot satisfy analytical needs. Interaction techniques are required to support the dialogue between the analyst and the data."*

Interaction in general can be thought of as being driven by some computational serving process that is able to produce within a "small" time frame a "verifiable" answer to a client's query that is formulated in a language common to both the server and the client. Following even this very limited thinking, it is clear that designing interactions in information visualization is a challenging task in several respects:

- First, there is no common "query language" between a visualization system and the human user.
- Second, the time frame used to judge the quality of interactivity is very small. Due to the high involvement of the human visual system what this time frame should be is not really well understood.
- Third, what the data encoding (i.e., the visual representation) shall be is a current research endeavor.
- Fourth, verification of the quality of visual feedback provided by a visualization system to a human is still a largely subjective endeavor that is highly domain dependent.

In the case of graph visualization systems the challenges are very unique but not as challenging as those of general visualization systems or even those of general information visualization. The reasons are the following: The data model is well specified (i.e., graphs) [25]; there are fundamentally not too many visual representations to choose from (i.e., some flavor of low dimensional node-link diagrams or matrix-tensor representations) [17]; the fundamental objects of discourse are nodes, edges, and patterns obtainable from them as sets or sequences (e.g., subgraphs, induced subgraphs, paths, cycles) [46]; and finally graphs are amenable to filtering and aggregation operations that preserve certain properties at large scale [40, 9, 7].

Our focus is on interactive system building rather than on promoting any particular set of visual tools. The fundamental problem we address is how to visually interact with a graph that is too large to fit on the available screen. We present a graph visualization system called CGV (Coordinated Graph Visualization) whose primary focus is

*Corresponding Author
Email addresses: ct@informatik.uni-rostock.de (Christian Tominski), abello@dimacs.rutgers.edu (James Abello), schumann@informatik.uni-rostock.de (Heidrun Schumann)

on interactivity. CGV is based on our personal experiences developing tools to extract information from a variety of "large" graphs arising in industrial and academic settings, including telecommunications, search engine, and biomedical data. We have been facing disk resident multi-attribute labeled graphs that do not fit in the available RAM (i.e., semi-external or fully external graphs [8]). The graphs have been typically sparse, low diameter, and with high clustering coefficients. Typical questions have revolved around informal notions like "global graph views", "high density graph regions", "induced sub-graphs on subsets of vertices whose associated attribute values satisfy a Boolean formula", and visual access to the neighborhood of a given vertex.

Fortunately, global graph views are well encapsulated by the notion of *maximal antichains* in a *hierarchy tree* (see [7] and Section 3); each node in a maximal antichain corresponds to a "graph region", and induced subgraphs can be viewed as the result of specialized range filters. This suggested to base the system on hierarchy tree computations and methods to extract graph macro-views that are small enough to fit on the available screen [6], where they are shown as multi-linked visual representations together with mechanisms to filter nodes and edges via range sliders.

To achieve scalability two special RAM resident macro-views (Top and Bottom) are used in conjunction with a fast disk index that is triggered by the Bottom macro-view. The Top and Bottom macro-views are parameterized by the amount of RAM available. Their views can be panned, zoomed and tagged for future reference. Panning has been enhanced by a special radar view introduced in [60]. Visual access to the neighborhood of a point required the introduction of special lenses and edge-driven navigation (first introduced in [61, 10] and considered later on in [48]). To ensure interactivity, multi-threading became a necessity. To reach a wide user base and to facilitate future system evaluation CGV has been designed as a dual-use system that can run as a stand-alone application or as an applet in a web browser. To our knowledge, there is no single turn key system that offers to an unsophisticated user the "large" graph visualization and navigation facilities offered by CGV.

In summary, any system for visualizing large graphs has to perform the following three tasks:

1. Build a graph hierarchy of logarithmic depth and bounded degree and extract from it graph macro-views that fit on the available resources.
2. Come up with multiple linked visual representations of a given hierarchy tree and its associated macro-views at a multi-scale level.
3. Devise visual interaction mechanisms that allow users to pan a large macro-view and to navigate among macro-views in a smooth fashion.

We describe how each of the three tasks is implemented in CGV, compare our system to a selection of existing systems, and provide feedback from early adopters. CGV's predecessor is the system ASK-GraphView [6]. To achieve flexibility, we opted for a modular multiple view design instead of a monolithic architecture.

1.1. Main Contributions

CGV's design allows for integration of synchronous and asynchronous computations. This is a necessary requirement for interactive systems to be able to generate and present users with feedback across multiple views in a consistent, uniform and timely manner.

Besides standard visual graph representations (i.e., node-link diagrams, matrix views, graph splatting), CGV uses focus+context hierarchy views to drive overall navigation. These views include a textual tree representation, two color-coded representations, and a projection of the hierarchy onto a 3D hemisphere. These views are complementary.

One of the system's novelties rests on the tools provided to interact in a coordinated fashion with these visual representations of "large" multivariate graphs at different levels of abstraction. Commonly accepted pan and zoom operations, are enhanced with pan-wheel interaction and edge-based navigation. These are augmented with visual cues that include smooth viewport animation together with a novel use of an infinite grid and a look ahead radar view. At the most refined level of granularity, CGV offers a variety of lenses that help reduce node and edge clutter. This is a feature we have not seen in existing graph visualization systems. Data filtering in CGV is aided by an interface to compose complex filters out of basic range sliders and textual filters.

CGV provides a history mechanism to allow for undo/redo of interactions. This is a feature rarely seen in current graph visualization systems.

In summary, CGV incorporates traditional and non-traditional interaction techniques to support the exploration of graphs at different levels of abstraction. One of CGV's strengths is its extensibility and ease of maintenance. CGV is implemented in Java and can be operated as a stand-alone desktop application across multiple platforms or as a visualization client in a web browser.

1.2. Paper Overview

In Sect. 2, we describe how we follow the Model-View-Controller design to meet our specific interactivity needs. CGV's data model is discussed in Sect. 3. The different views provided by the system are the subject of Sect. 4. Sect. 5 details the set of user tools that the system offers to interact in a coordinated fashion with multi-scale visual representation of large graphs. It also discusses visual augmentation of interactions, the user interface, and the history mechanism provided by CGV. Sect. 6 compiles a list of common interaction mechanisms and details the extent at which they are supported by CGV and four other graph visualization systems. Sect. 7 describes how early adopters feedback helped improve CGV. In Sect. 8,

we conclude and point out some directions of further research. The appendix illustrates the use of the system in two case studies.

2. CGV's Design and Architecture

2.1. Overall Goals

The literature offers a wealth of sophisticated algorithms for graph visualization. When such algorithms have to work in concert with each other and with different interaction methods, architectural aspects become important. In this section, we discuss related challenges and give an overview of how CGV addresses them.

One of the major challenges in graph visualization stems from the sheer size of readily available multi-attributed data. We refer to the size of a data set as the number of objects that it contains, e.g., the overall number of nodes, edges and labels for the case of string labeled graphs. The problem of visualizing large graphs (without explicit consideration of node or edge attributes) has been addressed in previous work (see [40, 9, 7] for a review). The basic idea is to transfer the big and hard to solve visualization problem to one that is scalable and amenable for interaction at different levels of abstraction. This is achieved by computing a hierarchy on top of the raw graph data (see Sect. 3). The hierarchy provides access to specially selected macro-views of the graph and is the backbone for interactive information drill-down.

In the case of graph data where the elements have associated a vector of attributes, multiple view approaches enable users to look at the data from different perspectives (see Sect. 4). The use of multiple coordinate views in a more general setting has been documented in [16, 69]. For exploratory analysis, users need to switch the perspective frequently. Therefore, it is crucial to provide means to navigate between different size data subsets and different data attributes. Current graph visualization systems offer standard interaction (e.g., brushing, zoom & pan) to support basic data navigation. Large and more complex data demand for better support.

Our goal has been to enhance existing interaction mechanisms to aid visual graph exploration. This is partially achieved by providing flexible interaction access to different data views. A critical aspect we have dealt with is the integration of synchronous interaction with otherwise asynchronous computations.

In summary, at a very high level the major driving issues behind our design decisions were:

- Flexible access to different views of the data,
- Enhanced interaction methods for graph exploration,
- Visual augmentation of interaction methods, and
- Integration of synchronous interaction with asynchronous computations.

Further decisions were made based on [56, 36].

2.2. System Design

To achieve the aforementioned goals, we follow the Model-View-Controller (MVC) design [43] as the basis for CGV. It dictates a strict separation of the *data model*, the *views* on the model, and the *controllers* that regulate interaction.

The basic elements of CGV's *data model* are attributed nodes and edges that constitute a graph hierarchy (see Sect. 3). This graph hierarchy allows for multi-scale access to large graphs. It is the fundamental data structure for interactive navigation between different data subgraphs. We use the *decorator pattern* [31] to equip nodes and edges at runtime with view-specific information depending on the visualizations used. Since not only the data determine the visual output, but also the visualization parameters, CGV integrates them in the data model. Making visualization parameters first-class objects becomes a necessity for the development of flexible, enhanced interaction.

CGV provides several dedicated *views*. Each view is designed as a monolithic interactive component that implements its own visual mapping and rendering as well as a common interface to access and alter data and visualization parameters (see Sect. 4). This allows us to utilize different graphics engines simultaneously to drive the visualization. We use Java2D for 2D views and rely on Java OpenGL Bindings (JOGL) for 3D views. To be flexible in choosing perspectives on the data, we opted for a design that is based on dynamic view composition, rather than on a hard-wired layout of visual components. In the default setup, six different views offer a broader perspective on a single data model. Users can create alternative view arrangements for different visualization tasks and store them for later reuse. To support visual comparison of two or more data models, several instances of CGV can be executed.

Controllers regulate interaction with the data model and its views (see Sect. 5). In information visualization, interaction is modeled as adjustment of the data model, which includes the raw data and its visualization parameters [41]. Adjustments can be conducted in two different ways: by direct manipulation and via a dedicated graphical user interface. We further distinguish between interactions that are coordinated and, hence, have global effect (e.g., dynamic filtering), and interactions that are not coordinated, i.e., have local effect (e.g., changing the zoom level of a view). To achieve a high degree of consistency among interactions, especially for those with global effect, the system follows a common interaction policy that ensures that a physical interaction (e.g., double left click) results in the same effect (e.g., expansion of a node) in all views.

To assist the user in interactively exploring the data and the parameter space of a visualization, CGV incorporates a history mechanism for undo and redo of interactions. Our implementation is based on the *command pattern* [31], which avoids holding duplicates of the data model in memory.

To ensure system responsiveness, CGV implements a threading component that is responsible for handling asynchronous computations. It has been designed as a worker queue that allows for the necessary level of control to handle synchronous interaction requests. However, there is no general rule to decide this. It is the task of the developer to incorporate proper synchronization points to allow for intermediate feedback and cancelation of long running computations. Careful implementation is necessary to avoid race conditions and deadlocks. This non-trivial task is currently not well supported [47]. The class `SwingWorker` of Java 1.6 provides some assistance – this is the main reason for using this most recent version of the Java language.

To target a broad user base, we developed a dual-use system that can be run as a stand-alone application or as an applet from a web browser. As recent developments like for instance ManyEyes [67] have shown, web-based systems are a good choice to reach users and foster collaboration. Fig. 1 summarizes the architecture of CGV. Its key elements are:

- Threading (worker queue for asynchronous computations),
- Model (graph hierarchy and visualization parameters)
- View (various visualization techniques)
- Controller (data and views interaction)
- User interface (docking framework, parameter panels, and toolbars)

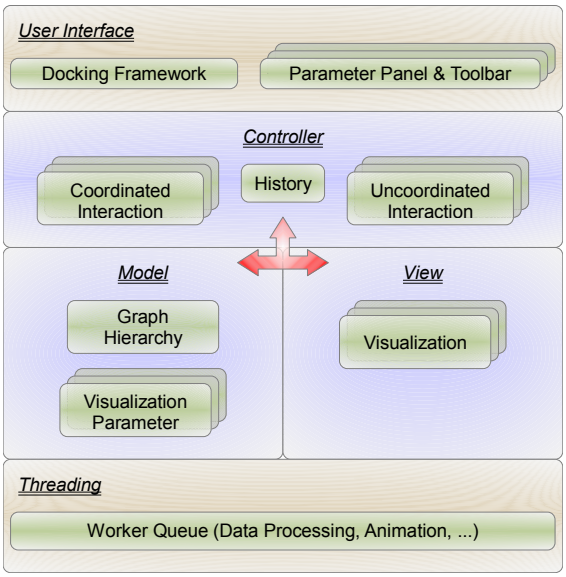

Figure 1: The Model-View-Controller architecture of CGV.

3. Data Model

CGV operates on data organized as a *multivariate graph* $G = (V, E)$ where $V \subseteq \{A_V^1 \times \ldots \times A_V^n\}$ is a set of attributed nodes and $E \subseteq (V \times V) \times \{A_E^1 \times \ldots \times A_E^m\}$ is a set of attributed edges. $A_V^i : 1 \leq i \leq n$ and $A_E^i : 1 \leq i \leq m$ are domains of *node attributes* and *edge attributes*, respectively. The attributes can encode quantitative or qualitative values. Some of them may be computed by the system itself (i.e., degree of a node or flow on an edge) or may come from external sources (i.e., a textual label). Conceptually one can extend attributes to subsets of nodes and edges. In the case of numerical attributes some form of aggregation function provides a way to associate a meaningful value to a node or edge set (i.e., sum, max, min, average, mean, centrality, eccentricity, etc.). In the case of qualitative string labels the computation required to obtain such group labels varies in complexity depending on the application.

Since interactivity is at the essence we rely on methods presented in [9] and [6] to handle computationally and visually large graphs. The main data structure used is a *graph hierarchy* H over the nodes of G, i.e., a rooted tree whose set of leaves is in one to one correspondence with the nodes of G. Non-leaves of the graph hierarchy are sometimes called macro-nodes or cluster nodes. Central substructures in a graph hierarchy are *maximal antichains*, which are "cuts" through the hierarchy tree, or put differently, they are maximal collections of hierarchy nodes such that no pair is a descendant of the other. Each maximal antichain represents a partition $< V_1, \ldots V_k >$ of V. It determines a *macro-view* of G with k nodes where node i represents the set $V_i \subseteq V$. Details on the construction of H and on determining "good" macro-views can be found in [6]. The system can work with pre-computed graph hierarchies as well, which is important in certain application scenarios.

4. Visual Methods

The different aspects of graph hierarchies we address here require multiple views on the data. We need views to communicate the general macro structure of G, dedicated tree visualization techniques to represent the graph hierarchy H, and overviews to preserve the user's mental map. Moreover, quantitative and qualitative attributes, such as weights, statistical meta data, or labels need to be displayed. In this section, we briefly describe purpose and characteristics of the proposed views. Most of them are instantiations of known approaches, so we refer interested readers to the original publications for more details.

4.1. Main Graph View

At any point in time CGV displays in its main canvas a node-link representation of a macro-view of the graph

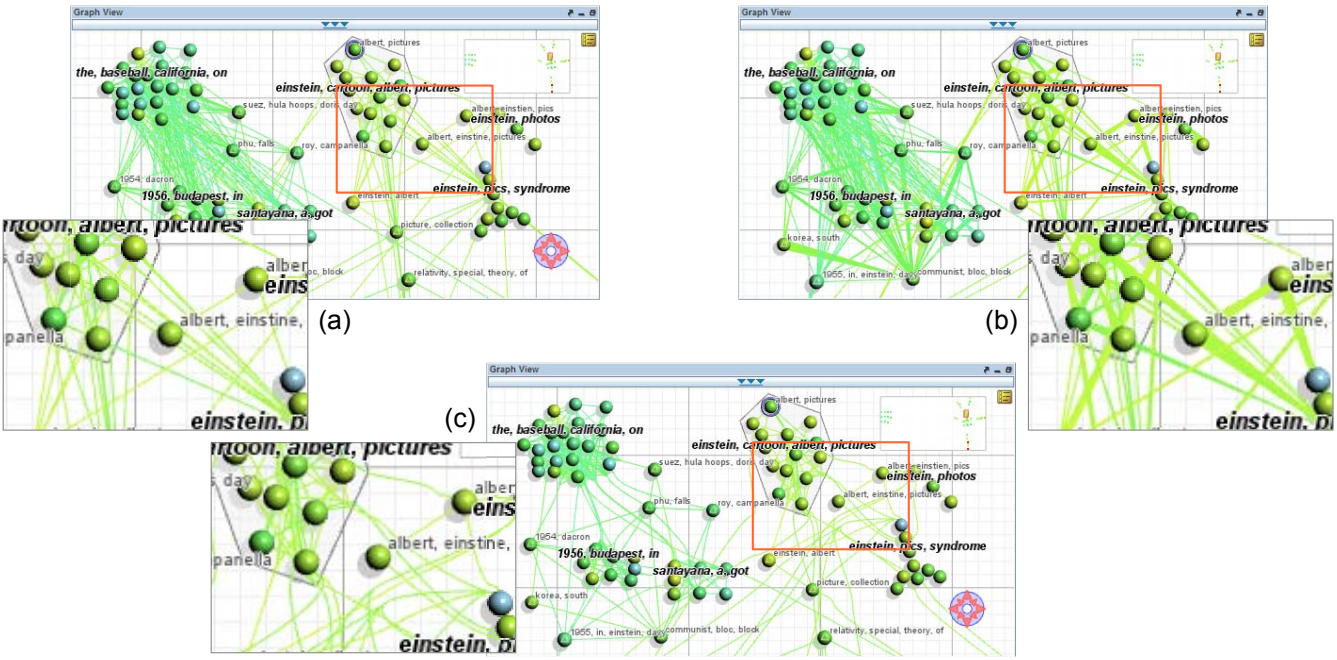

Figure 2: Different edge representations in the main graph view – (a) Plain edges; (b) Proportional edges; (c) Bundled edges.

G (see Fig. 2). Currently, we use classic spring embedders [17], squarified treemap layouts [20], and LinLog layouts [49] depending on the characteristics of the graph region being examined. To reduce computation time, layouts are lazily created, i.e., layout information is computed only for those portions of the graph that are currently being explored. Users can reposition the displayed nodes to improve the layout interactively.

In this view, nodes are represented as spheres. Color and size of the spheres are used to encode numerical attributes on demand according to user specifications. Additional iconic shapes can be attached to the spheres to convey structural properties of the sub-graph induced by a node. Edges can be represented in three ways (see Fig. 2). First, a classic straight line representation can be used to visualize connectivity. Second, CGV represents an edge as a line whose thickness at the source node and at the target node is proportional to the relative amount of edge weight that the edge contributes to the sum of all edge weights at the source node and the target node, respectively. This *proportional encoding* results in tapered edge shapes that indicate the "information flow" in a network. Third, edge bundling and convex hulls are used to de-clutter the view and to emphasize nodes cluster affiliation. String labels provide textual information (see Sect.5.6). A novelty is that the displayed graph layout can be easily tiled for individual printing. These tiles can be manually assembled as a large poster for presentations or off-screen collaborative data analysis.

4.2. Hierarchy Representations

CGV uses the following coordinated views to represent the graph hierarchy H, which is the backbone for the navigation between different macro-views of G. These views give an overview of H, represent the current level of abstraction (i.e., maximal antichain), and visualize selected node attributes (qualitative or quantitative).

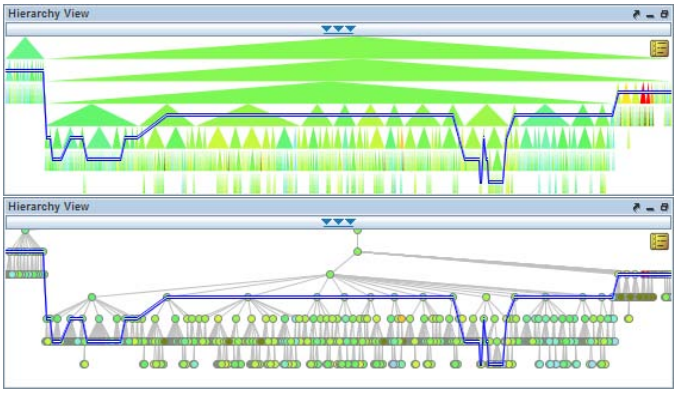

Figure 3: Two alternative representations of the hierarchy view.

Hierarchy Views. Two alternative representations of the overall topology of H are offered that follow Reingold-Tilford layouts [51]. One representation uses triangles the other dots and lines (see Fig. 3). In both cases, a superimposed polyline indicates the current antichain. Color coding is applied to visualize a selected node attribute. While the triangle-based representation makes it easier to recognize colors and, hence, to grasp an overview of the value distribution, the mapping to dots and lines is better suited to convey structure.

Textual Tree View. This view (see Fig. 4) complements the hierarchy view in that it presents textual labels of the

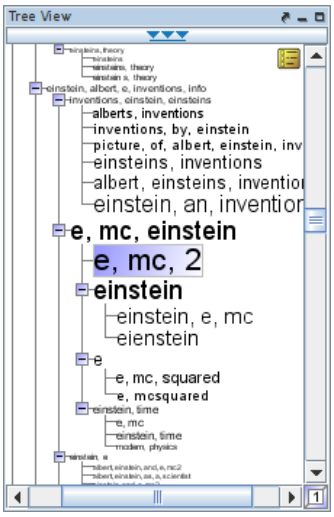

Figure 4: The textual tree view shows labels that can be optionally magnified by a fisheye transformation.

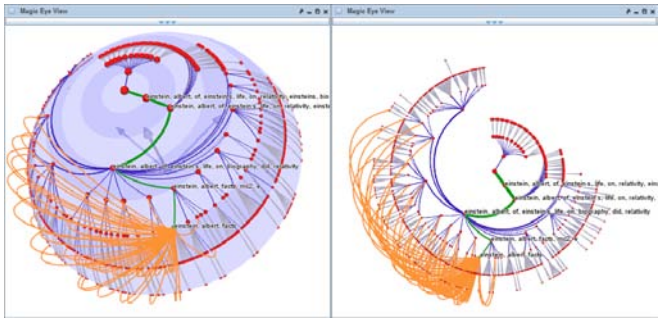

Figure 5: The magic eye view represents cross-edges in addition to the edges of the graph hierarchy in 3D or 2D.

nodes of H (which is not possible in the hierarchy view). The expanded and collapsed nodes in the view represent the current maximal antichain.

Magic Eye View. The magic eye view is based on a Walker layout [21] that is projected onto a 3D hemisphere. It was originally developed as a focus+context technique for hierarchies [45]. We extended the original design to visualize selected cross-edges among nodes of the hierarchy by representing them as arcs spanning around the hemisphere surface (see Fig. 5(left)). With this extension, the magic eye view represents both the graph hierarchy H and selected edges of a macro-view of the underlying graph G. As an alternative, users can also switch to a 2D representation of the radial tree layout (see Fig. 5(right)).

4.3. Complementary Overviews

The aforementioned views are all suited to see details of nodes and edges of the hierarchy H and the underlying graph G. To help users maintain an overall mental map of the data two additional representations are incorporated:

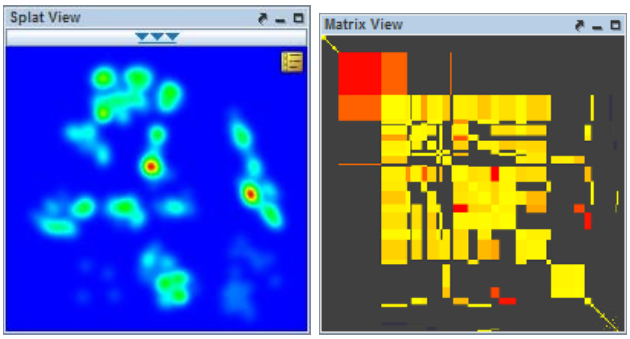

Figure 6: The splat view (left) provides a condensed overview of the graph layout. The matrix view (right) visualizes edge weights and edge density.

Splat View. This view represents the layout of a macro-view of the underlying graph G in condensed form as a color-coded map [65]. It is helpful in spotting quickly dense layout regions, even if the available screen real estate is limited (see Fig. 6(left)). As an alternative to the default rainbow color scale, users can opt to use a gray scale encoding to avoid the disadvantages inherent in rainbow color scales [19].

Matrix View. Alternatively, CGV uses a matrix representation to give an overview of the edge weight distribution of a macro-view of G (see Fig. 6(right)). The weights are color-coded in the matrix cells using a yellow to red color scale; gray is used to color those cells where no corresponding edges exist. The main diagonal cells are used to color-code node characteristics, which in the case of clustered graphs may be taken to represent edge density in the cluster. This helps users in spotting dense clusters that may be interesting to explore next.

4.4. Remarks on Views

This section provided an overview of the different views available in CGV. In addition to these views, which all represent connectivity and edge/node attributes the system provides several meta views. They contain meta information like statistics about the data set, detailed textual descriptions of node properties, search results, or the current system status. Meta views provide users with assorted context information that assists in the exploration process.

It is worth mentioning that the graph view, the hierarchy view and the textual tree view are endowed with fisheye-based focus+context representations that allow users to access detailed information quickly without loosing the overall context [61]. This capability greatly enhances readability of colors and labels and it is particularly useful for browsing visually large data sets. The magic eye view is by design a focus+context technique.

The architecture of CGV enables different assemblies of views (see Sect. 5.7). This mechanism can be used to provide customized view arrangements depending on the application context and the visualization task.

5. Interaction

For visual data exploration, interaction is a necessity. Since explorative analysis tasks can be many-faceted, an a-priori decision for a particular visual mapping is impractical. Well-designed interaction helps users choose relevant data subsets and adjust the visual mapping to suit a particular course of visual exploration. To that end, CGV provides several ways of interacting with the data and their visual representations.

5.1. Coordinated vs. Uncoordinated Interaction

Our system distinguishes between coordinated and uncoordinated interaction. Uncoordinated interactions are those that are local to a particular view. For example, there is no need to coordinate changes of visualization parameters (e.g., color scale or font size) across views, because each view is monolithic and implements its own mapping strategy and has visualization parameters that are independent of other views.

In contrast, coordinated interactions are those that change the global perspective on the data, that is, all views are consistent in what they represent (but not how). Operations on the data model like filtering graph elements or choosing a different macro-view are examples of coordinated interactions. CGV implements coordinated interactions as follows. When a user performs an interaction in a view, the request is propagated to a controller that in turn notifies all other views of the pending operation. This gives all views the opportunity to take any actions required to prepare for the pending interaction. After that, the controller performs the interaction on the data model, which might include scheduling costly calculations on the worker queue (e.g., layout computations). Once the work is done, the controller informs all views about the particular change, the view having initiated the interaction being informed first. Each notified view updates itself to react to the change in the data model. Again, this can involve computations that need to be asynchronous (e.g., layout generation or animation). Given the multitude of views that can initiate this procedure, it is necessary to follow a consistent common interaction policy. For instance, a double left click on a node should result in expansion of the node, no matter in which view the double click was performed.

5.2. Basic Interactions

In any visual interactive system we can think of, interaction presupposes some form of object selection [70], together with mechanisms to temporarily lock a view focus and to coordinate data and visual operation updates across multiple views. Next we present a more or less high level description of fundamental CGV interactions. To convey an impression of how these basic interactions can be applied by users, they are presented in an order that resembles a real usage scenario, rather than in an order that relates to interaction complexity.

Zooming, Panning, and Fisheye Magnification. One common interaction is to adjust the viewport to a level of graphical abstraction that suits the task at hand. This includes zooming and panning operations as well as scrolling. In situations where only a quick view on details is sufficient, users can shortcut viewport adjustments by using dynamic fisheye magnification techniques (in the hierarchy view, the textual tree view, and the graph view).

Identify. Once users have found suitable viewports they usually want to *identify* the visualized data objects. This can be accomplished by hovering the mouse cursor over visual elements in any view. The identified object is highlighted in all visualizations and its detailed information is shown in a dedicated meta view.

Locate. Making an identified data object visible in all views is cumbersome if this has to be done by adjusting all viewports separately. As a novel alternative to manual viewport adjustment, we implement the *locate* interaction, which is activated by a left click. Its purpose is to broadcast to all the views the fact that a data object has become the focus of interest in a particular view. When a view receives notification of this type of interaction it is obliged to adjust itself so that the involved data object becomes visible. This operation is particularly useful in cases when users spot an "interesting" element in one view and want all other views to automatically focus on the same element.

Lock/Unlock. To help users stay focused on an identified data object, it is possible to *lock* it (CTRL+left click). In lock mode the focus is fixed and any request to change the focus is neglected (i.e., hover is deactivated) until the lock is released by an *unlock* operation (CTRL+right click). While being in lock-mode, CGV can provide further interactions that work only with the fractional size of the focus, but would be hard to apply to all or even a subset of the data objects. One such interaction is edge-based traveling, which we describe later.

Brushing. Setting a primary focus on a single data object via locking is complemented by *brushing*, which can be used to set a secondary focus on multiple data objects by holding SHIFT and using the left mouse button (click or drag). Since there is no restriction on the number of brushed elements, CGV highlights them sparingly to avoid cluttering (i.e., using little screen space and few visual attributes).

Expand/Collapse. Since CGV's basic data structure is a graph hierarchy, *expand* and *collapse* can be used to interactively change the macro-view of the graph, i.e., to control the information drill-down. The expand operation (double left click) provides access to more details by replacing a selected node with a layout of its induced subgraph. Collapse is the inverse operation (double right click) and can be used to get back to an overview.

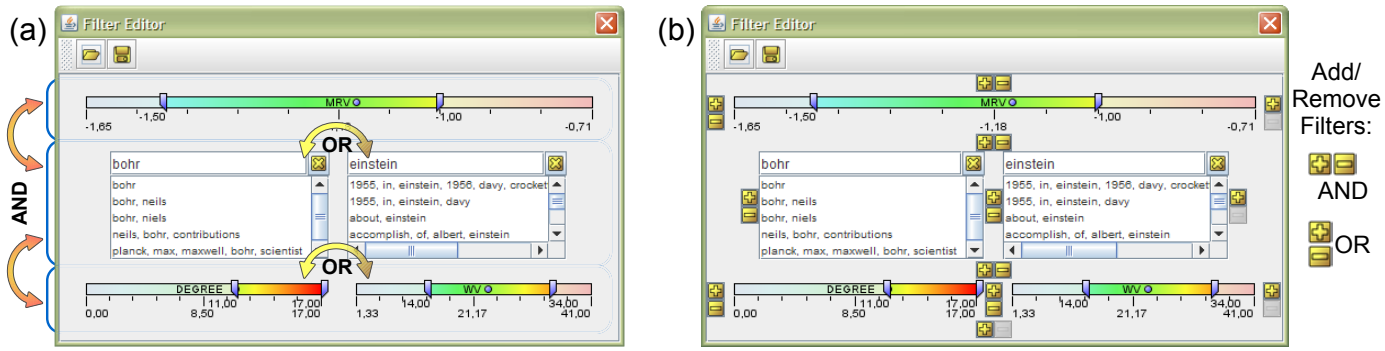

Figure 7: Enhanced dynamic filtering via a logical combination of five elementary filters – (a) Basic mode; (b) Advanced mode.

Visualization Parameter Adjustment. All views provide some mechanism to adjust visualization parameters either by direct manipulation (e.g., mouse wheel rotation) or by entering values in a user interface. In Sect. 5.7, we discuss the interface facilities that CGV offers in this regard.

-

In contrast to visualization parameter adjustments (which are uncoordinated), identify, locate, lock/unlock, brushing, and expand/collapse are coordinated interactions. All these basic interactions are supported by all views. In the previous descriptions, we also indicated the common interaction policy that all views should follow: The left mouse button is associated with "positive" interactions (e.g., identify, locate, lock, brush, expand), while the right mouse button handles "negative" interactions (e.g., unlock, de-brush, collapse). This distinction has been made to help users apply the interactions consistently, even though the views are quite different.

5.3. Dynamic Filtering

When exploring a graph with respect to certain attributes, it can be very helpful to dynamically *filter* out irrelevant nodes and edges [13]. Depending on view characteristics and visualization tasks, two alternatives exist to display filtering results: filtered objects can be dimmed or they can be made invisible. Dimming objects is useful in views that maintain an overview, where all information needs to be displayed at all times, but filtered objects need only to be indicated. Making objects invisible is useful in views that notoriously suffer from cluttering.

Basic Dynamic Filtering. Elementary filters are commonly applied to describe conditions that must be satisfied for an object to pass through. Range sliders are an effective mechanism to filter by any particular numerical attribute. These include exogenous data or numerical attributes computed by the system like degrees, peeling numbers, or weights. In addition to range sliders, CGV provides a textual filter that extracts objects with specified labels. Both basic filters allow users to specify manifold filter conditions by simply moving a slider or typing letter strings. Further elementary filters can be integrated if required.

Enhanced Dynamic Filtering. For complex data sets where nodes and edges are associated with a multitude of attributes, relying solely on elementary filters is not sufficient. The next natural step is to combine elementary filters to provide some form of multidimensional data reduction. The usefulness of this approach is discussed in [26]. In CGV, composite filters can be created by logically combining elementary filters. A logical AND combination generates a filter that can be passed only if an object obeys all conditions. An object passes a logical OR filter if it satisfies any of the composed filter conditions.

The question that needs to be answered is how to enable the user to create composite filters dynamically during runtime. While other systems offer only fixed filter combinations or require users to enter syntactic constructs of some filter language, CGV implements a visual interface where the user can visually specify logical combinations of filters. The interface can be operated in two modes – basic and advanced. In basic mode, only the elementary filters can be adjusted, whereas the structure of the filter (i.e., the logical combinations) is fixed (see Fig. 7(a)). In advanced mode, the structure of a filter is subject to change (see Fig. 7(b)). This allows users to compile more complex filters, which can be conserved for later reuse once they have proven useful. The advanced mode is kept at a steerable level by following the sieve metaphor, according to which objects have to pass the filters from top to bottom. The sieve is modeled as a single logical AND combination of an arbitrary number of logical OR combinations, which in turn contain arbitrary elementary filters. The elementary filters involved in an OR combination are arranged horizontally, indicating that an object can pass any of them. The AND filter is represented as vertical arrangement of the participating OR combinations, indicating that an object has to pass all of them. Note that the restriction to the sieve metaphor only affects the user interface. Internally any logical combination of filters is possible.

Fig. 7 shows an example of a filter that can only be passed by nodes that have an attribute value named *mrv* in the interval [1.5; 1.0], and that contain in their labels the terms "bohr" or "einstein", and that have a *degree*

between 11 - 17 or a node weight (wv) between 14 - 34:

$$mrv \in [-1.5; -1.0] \wedge$$
$$\text{“bohr”} \subset label \ \vee \ \text{“einstein”} \subset label \wedge$$
$$degree \in [11; 17] \ \vee \ wv \in [14; 34].$$

As an illustration of the flexibility of these filters consider the exploration of time varying multi-graphs. Browsing with respect to time is a typical task when exploring such data. By modeling time as an attribute one can associate a range filter to it and use a corresponding time slider to obtain a sequence of time varying snapshots of the graph's evolution (an implementation of a dedicated time slider is not necessary).

5.4. Graph Lenses

Dynamic filtering provides a means to globally adjust what is being displayed in CGV's views. For more local interaction, we apply interactive lenses. Although lenses are recognized as useful tools to support visual exploration [18, 57], they have been considered only recently for graph visualization [61]. In CGV, lenses are used in the main graph view to locally tidy up layout regions that are cluttered and to generate local overviews. They can be switched on/off by a single click and can be moved around using drag and drop.

Local Edge Lens. A lens that operates on the rendering stage is the *local edge lens* [61]. It is used to unveil the connectivity of nodes when the visual representation is cluttered with edges. To tidy up a lens perimeter, we consider as relevant only those edges that are adjacent to nodes inside the layout area covered by the lens. During rendering we apply stencil buffering to allow only relevant edges to be displayed in the lens view. Compare Fig. 8(b) to Fig. 8(a) to see the effect of this lens.

Layout Lens. Consider the example depicted in Fig. 8(b). The application of the local edge lens clearly reveals that the focused node is connected to eight neighbors. However, only five of them are visible in the current viewport. To discern the characteristics of the other three neighbors, the user would have to navigate to each of them manually – a costly procedure that is common in known graph visualization systems. To address this and similar exploration tasks, CGV provides the *layout lens*, which is a generalization of the bring-neighbors-lens introduced in [61]. Its purpose is to adapt the graph layout to gather nodes that exhibit certain similar characteristics, but that might be scattered in the layout (e.g., as the set of neighbors in Fig. 8(b)). When applying this lens, nodes change their position according to the lens position and the lens perimeter. While moving the lens towards a focus node, the affected nodes are attracted towards the lens. When the lens reaches the position directly above the focus node, all affected nodes become located within the lens perimeter, with the node originally farthest away now on the lens perimeter. Fig. 8(c) shows that this generates a local overview of the focus node and the affected nodes (including the three previously unknown neighbors). While the focus node is simply defined as the node nearest to the center of the lens, affected nodes can be determined in different ways. Currently, we limit ourselves to affect nodes that are neighbors of the focus node in the current macro-view. In this sense, our current implementation can be termed topological, but one could think of extending the current capabilities to provide semantic lenses based on similarity of nodes as long as smooth interactivity is maintained.

Fisheye Lens. A lens that is useful to temporarily separate local accumulations of nodes and to reduce node clutter is the *fisheye lens*. It operates on the node positions only, not on node sizes.

Composite Lens. The layout lens can sometimes accumulate too many nodes in a local region (see Fig. 8(c)). The *composite lens* resolves this problem by superimposing the effects of the local edge lens, the layout lens, and the fisheye lens in one operation. It tidies up the lens area, gathers relevant nodes, and spreads those that get accumulated in the lens center as depicted in Fig. 8(d).

-

The on-demand character of the described lenses make them an interesting alternative to resolve visibility issues (i.e., edge clutter, out-of-view objects, and node accumulation) in cases where classic zoom and pan are impractical or cumbersome to apply. CGV's architecture allows to programmatically place lenses in the graph layout, for instance to emphasize interesting data elements.

5.5. View Space and Data Space Navigation

Navigation of the view space is an essential task during visual exploration. A fairly standard approach not only in graph visualization is to let users grab a view and drag it as necessary. Though being a very natural and intuitive interaction, it implies higher physical costs in terms of mouse milage [35]. CGV alleviates these costs by providing alternative navigation methods: *pan-wheel navigation* for the view space and *edge-based travelling* as a dedicated data space navigation method.

Pan-wheel Navigation. In contrast to common pan or scroll operations, for which users know the specific position to where they want to go, the *pan-wheel* interaction allows users to travel the view space freely when no particular graph region needs to be reached. This opens up the possibility of spotting “interesting” areas for further inspection via zooming into more specific regions. Pan-wheel navigation is activated by simply holding the left mouse button and dragging the mouse to specify direction and speed of the navigation. The pan-wheel navigation can optionally be augmented with a radar view as described in Sect. 5.6. As interesting aspect for future work will be to integrate speed-dependent zooming, which has shown to be effective for larger view spaces [22]

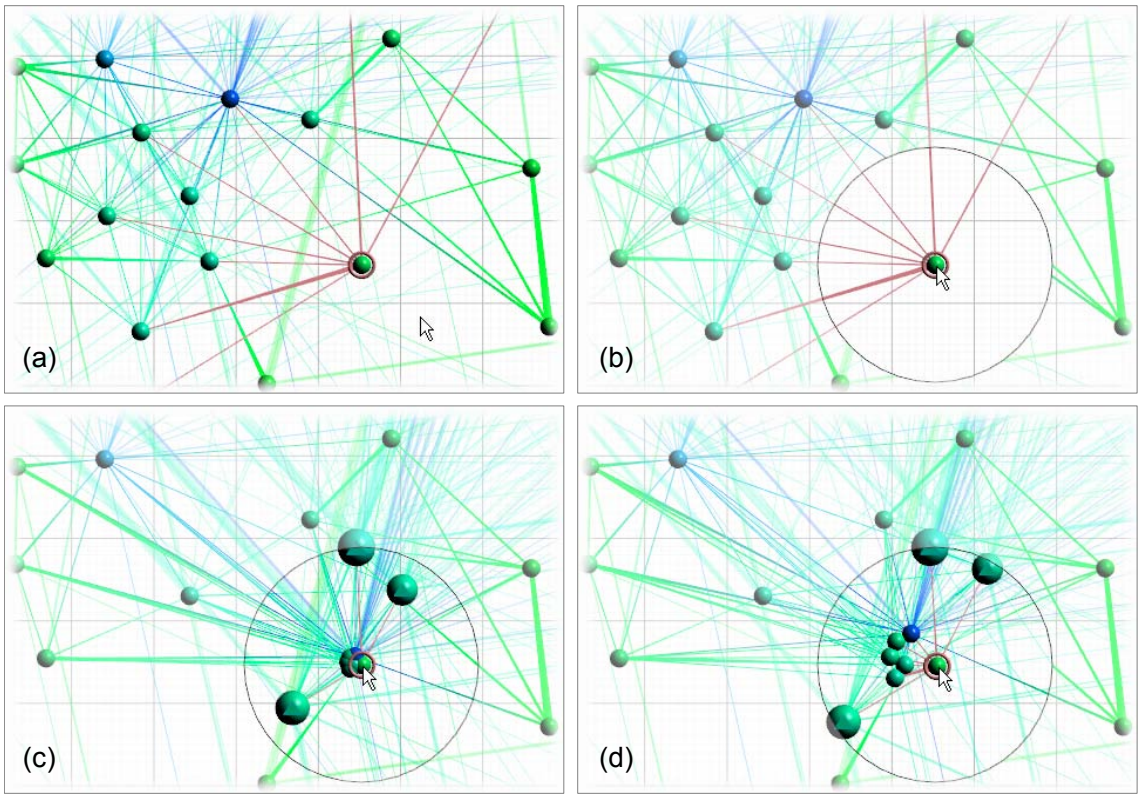

Figure 8: Interactive graph lenses – (a) View with a focused node; (b) The local edge lens removes edge clutter; (c) The layout lens gathers nodes that are adjacent to the focus node, but might be scattered in the layout; (d) The composite lens combines (b), (c), and a fisheye lens to tidy up the lens area, gathers relevant nodes, and spreads those that get accumulated in the lens center in a single operation.

Edge-based Traveling. Current graph visualization systems offer only limited support for data space navigation. Most systems map the task of data space navigation to one of view space navigation. But graphs offer a well-defined structure, which should be utilized for data space navigation. CGV follows this idea and provides a navigation method called *edge-based traveling* [10]. It operates on the structure of the currently explored macro-view of G and allows users to explore the macro-view's structure by performing a series of simple mouse clicks. To activate edge-based traveling, the user has to lock on a node u. After that it is possible to click any edge adjacent to u. From then on there are two options to follow: travel or preview. The user can left click an edge to travel to the node v that is connected via the clicked edge (u, v). This is implemented as a composition of the following basic operations: unlock u, locate v, and lock v. The first unlock operation is a prerequisite. The locate operation causes all views to center on v. The final lock operation ensures that edge-based traveling can proceed further from the just focused node v. Since not all nodes that can be reached via edge-based traveling are necessarily visible in the graph view, users sometimes do not know where they will be traveling when clicking an edge. In such cases, the preview mode helps. Users can right click an edge (u, v) to center it in the view so that both u and v become visible. In the preview, users can decide to actually perform edge-based traveling via a left click or to abort the preview and return to the original view by right clicking a second time. Both preview and traveling are augmented with smooth viewport animations (see Sect. 5.6). This edge-driven navigation is a novel interaction mode that to our knowledge has not been incorporated in previous systems. In combination with appropriately set filters and lenses, it is a useful tool to explore large graph layouts in concert with very localized data characteristics.

5.6. Visual Augmentation

Interactions can encapsulate complex semantics, which are not immediately apparent to users. To make CGV's interactions more understandable and efficient to use, they are augmented with visual cues. The goal of this section is to illustrate the potential of such visual cues.

Use of Animation. Animation is a helpful tool to support the user's mental map [50, 39]. We apply two kinds of animations. One is view-centric and augments the navigation in the view space; the other is data-centric and augments the transition between different levels of abstraction.

Viewport switches are caused among others by zoom and pan operations, locate interactions, and edge-based traveling. When interacting with large graph layouts, viewport switches can be difficult to comprehend. CGV's graph view applies smooth viewport animation following the method

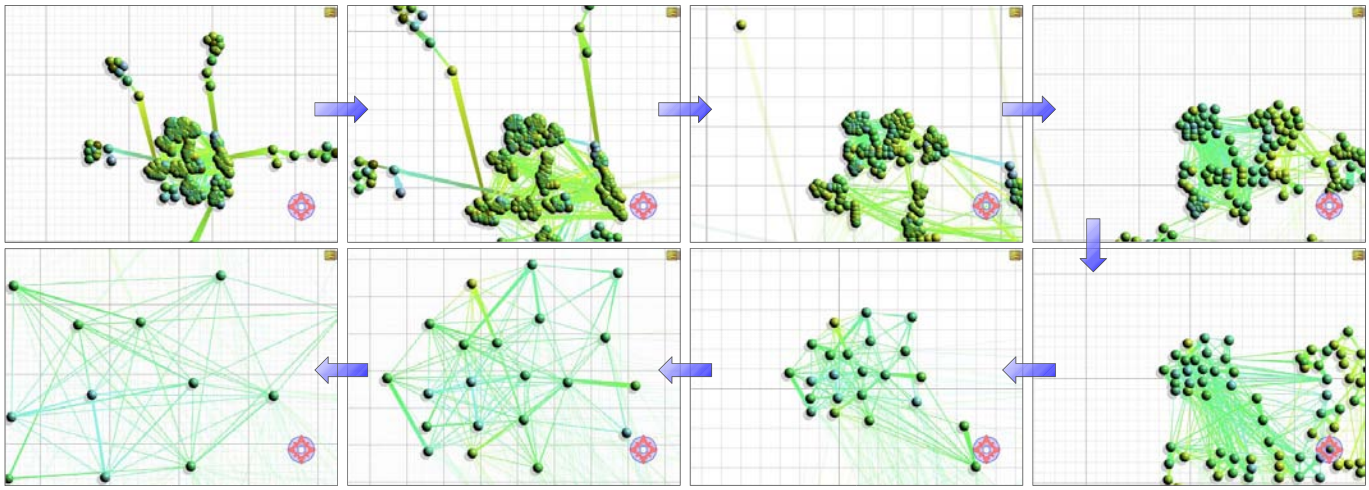

Figure 9: The infinite grid – The image series shows how the grid changes when zooming into a cluster of nodes.

described in [66]. During animation the viewport is parametrically interpolated between origin and destination. It is simultaneously slightly enlarged to facilitate the overview. The length of the animation is determined by the distance between origin and destination.

The data-centric expand and collapse interactions change the level of abstraction by exposing new nodes or removing some of them. Since this changes the layout of nodes and edges significantly, smooth animation is applied in the textual tree view and the graph view. This is implemented as an interpolation between the position of a newly exposed node and the position of its parent node. Expand animations appear as if nodes emanate from their parent, whereas collapse animations work the opposite way. These animations are time-bound, that is, they are finished within an assured time frame. We figured 500ms and 2s to be good time frames for the textual tree view and the graph view, respectively. However, these values heavily depend on user preferences, and thus, are subject to parameterization.

Both kinds of animation can be interrupted by the user at any point (recall synchronous interaction vs. asynchronous animation). While expand and collapse animations immediately switch to the final result upon interruption, the viewport animation just stops. This gives users the option to pause if they spot something interesting during the animation.

While animation in a single view is a well-investigated solution, a question remains for multi-view systems: Can animation really be useful if we animate multiple views simultaneously? Users could hardly follow all changes, and animation could lead to confusion rather than support [63]. Therefore, all animations in CGV are queued and played one after the other. The view that has been the source of an interaction (i.e., is in the user's focus) is animated first. Possible extensions of this very basic animation scheduling could account for proximity and sizes of views to prioritize larger neighbors of the source view and neglect animations of distant views, which can be necessary for a larger number of views.

Infinite Grid. CGV's main graph view visualizes the general structure of a macro-view of G and uses node size to encode a selected node attribute. To maintain expressiveness, node sizes are kept fixed during zooming operations. However, this impairs perception of closeness and distance of nodes in the layout during zooming, because size is an important visual cue to judge them. To overcome this disadvantage, we use a novel grid display termed *infinite grid* that provides a form of visual reference for the perception of distances. An ordinary grid is not sufficient, since the layout coordinates span several magnitudes (due to incremental layout computation). The infinite grid shows a primary grid for the magnitude of the current zoom level. A secondary grid is sketched in a dimmed color to indicate the next lower magnitude. When the zoom level is increased the secondary grid smoothly takes on the fully saturated color of the primary grid. At a certain point, the secondary grid fully resembles the primary grid and replaces it, and a new very dimmed secondary grid appears (see Fig. 9). This procedure works analogous when decreasing the zoom level. Our current users found the infinite grid particularly helpful to judge distances during viewport animations, which perform a smooth change of the zoom level.

Radar View. We introduced the pan-wheel navigation as a method for navigating in CGV's main graph view. A weak spot of this approach is that users are unaware of what they might discover during the course of the navigation. To address this concern, pan-wheel navigation is augmented by providing users a novel look ahead guide of what is coming next in the current travel direction. Look ahead guidance has recently been suggested for visualization on mobile devices [34], but has not yet been proposed for graph visualization.

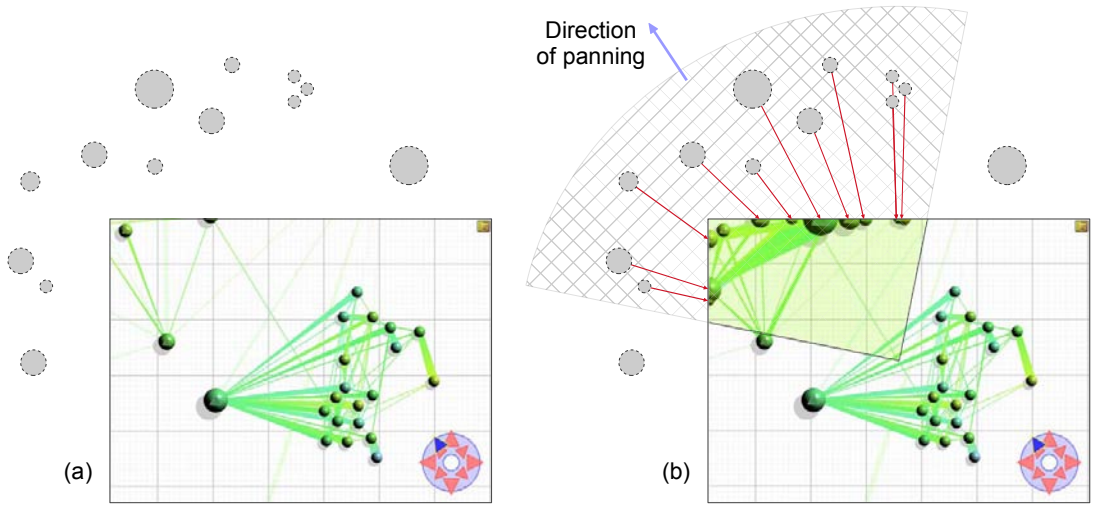

Figure 10: The radar view is designed as to project off-screen nodes to the viewport. This provides guidance during pan-wheel interaction toward possibly interesting spots in the data. (a) Navigation without radar view; (b) Navigation with radar view.

What we call the *radar view* [60] shows a semi-transparent circular sector that emanates from the center of the view to infinity and that is aligned with the selected navigation direction of the pan-wheel. The sectors's central angle can be parameterized. We currently use a fixed angle of 90°. Dynamically changing the angle depending on traveling speed is also feasible, but it is not currently implemented. During pan-wheel navigation, all nodes that fall into the circular sector, but are currently not within the viewport, are projected to the borders of the viewport so that they become visible. The different sizes and colors of the projected nodes provide a look ahead mechanism of what will be found in the current navigation direction. Users can use this as a guide to adjust their travel direction with the pan-wheel. The radar view then adjusts itself accordingly. Fig. 10 illustrates pan-wheel interaction with and without the radar view.

Labeling. Labeling is an important visual cue. It helps users establish connections between visualized data characteristics and particular data objects identified by some textual labels (e.g., the larger red-colored node (characteristics) represents "Einstein's work" (object label)). This is one of the reasons for treating string labels as first-class objects in the data model and in the system views.

In particular, labels are always embedded explicitly in a visual representation if they do not occlude the view. The textual tree view representation of the graph hierarchy H can be seen as a complete taxonomy of the underlying graph data in label space. Coordinated linking of any view with the textual tree view together with the text search facility makes the connection between the displayed data objects and their string labels quite apparent. CGV strictly adheres to the policy of always highlighting node labels when corresponding nodes are activated in any view.

Because avoiding occlusion among labels and between labels and other visual features is a computationally complex task, the main graph view and the magic eye view display labels in a selective manner only. The magic eye view shows labels for the currently highlighted node, and only optionally for its ancestors or for nodes connected via cross-edges. This keeps labeling costs at a manageable level. The graph view follows a global labeling strategy, that is, it tries to label as many nodes as possible. If "enough" display space is available, labels get attached to nodes directly. Otherwise, it attempts to label ancestors with aggregated labels (visually differentiated by font type and size, see Fig. 2). This simple strategy accounts for label density, i.e., the number of labels per available screen space. The net effect is that overviews show few aggregated labels only. During zooming, the labeling is refined and labels are displayed in an incremental fashion according to the zooming level.

5.7. User Interface

Interactive visualization using multiple views requires a well designed user interface. This includes both, the arrangement of views on the display and the interface to allow users to parameterize views.

The integration of visualization parameters in CGV's data model allows us to export them via two different user interfaces (see Fig. 11). One is a panel that lists all available parameters using standard widgets (e.g., buttons, sliders, etc.). Users can adjust parameters to their needs and optionally apply the changes to get visual feedback. We refer to this as asynchronous parameterization. But there are certainly parameters that are used more often than others during interactive visual exploration. To ease the adjustment of important parameters, we provide instant access via a toolbar that processes requests for parameter changes immediately; we call this synchronous parameterization. Recently, it has been shown that duplicating important functionality from an all-encompassing

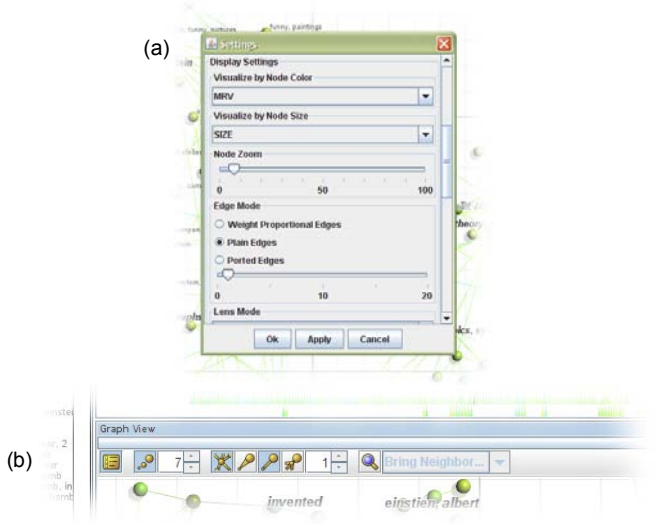

Figure 11: User interface for visualization parameters – (a) All-encompassing settings panel; (b) Toolbar for quick access to important parameters.

control panel to an exposed position (e.g., a toolbar) is a useful way to drive adaptable user interfaces [30]. In CGV, the toolbar can be considered to be adapted to the common user, whereas the panel for setting all visualization parameters is for advanced and expert users only.

When it comes to arranging multiple views on the display, there are two extremal positions one can take. One is to use a fixed arrangement that has been designed by an expert and has been proved to be efficient. The other is to provide users with the full flexibility of windowing systems, allowing them to move and resize views dynamically. Both extremes have their pros and cons and are actually applied in the literature, e.g., [6] for fixed arrangement and [15] for full flexibility. We do not go for either extreme, but instead make use of a docking framework[1]. The main reason is that docking maintains flexibility, but imposes certain order in terms of what arrangements are possible. Preferably, views should not overlap partially; a view should either be visible or not. To achieve this, the available screen space is partitioned into regions, each containing one or more views. The regions can be resized and moved as long as the arrangement remains a partition, i.e., remains overlap-free. A region that contains multiple views provides an interface to switch between them (usually tab-based). Partially visible views can thus be avoided. Fig. 12, 13, and 14 (see last pages) show possible arrangements of the views provided in CGV. The docking framework we use also allows for predesigned arrangements. This is a requirement for user-centered and task-oriented visualization. By default CGV uses an arrangement of views that is suited for visual exploration, which is similar to that in [6]. We have also experimented with other view arrangements for specific tasks or data sets, as for instance for browsing through time-varying graphs. Besides relying on pre-designed view arrangements, experts or users can design their own customized ones and store them for later reuse. This is a useful feature that allows for easy adaptation of CGV to domain- and task-specific needs.

[1] See `www.infonode.net` for details.

5.8. Interaction History

As described in [55], keeping a history of user interactions supports iterative visual exploration. This allows users to undo operations that were performed by accident or that yielded unsatisfactory results. Even though history mechanisms have long since been recognized as important in the HCI community (e.g., [11]) and are gaining importance in the visualization community (e.g., [24, 44, 41]), they are rarely present in current graph visualization systems.

We provide for basic undo and redo to close this gap. A history manager enables us to keep track of interactions and to undo them. Undone interactions can be redone, unless a new interaction is executed, which causes the history manager to dispose of all previously undone interactions. This behavior resembles a linear history as provided in many computer applications (e.g., web browsers).

Currently, CGV's history mechanism supports commands that are deterministic and do not require a copy of the whole data model to undo them (e.g., brushing as well as expand and collapse operations (see Sect. 5.2). Synchronous parameterization performed via the toolbars of CGV's views (see Sect. 5.7) is logged and can be undone. This helps users tune different parameter settings that best suit their visualization task.

-

In summary, CGV supports visual graph exploration by integrating standard and enhanced interaction techniques. Some of the interactions introduced in this section can be applied in other visualization contexts (e.g., the user interface to create advanced filter queries during runtime, the radar view as a look-ahead guide for off-screen objects, or the infinite grid as a spatial reference). We compare CGV to selected graph visualization systems in the next section.

6. Related Work

A wide range of graph visualization software has been and is still being developed. These include *general graph libraries* like JUNG (Java Universal Network/Graph Framework) [3] or JGraphT [2], which focus on algorithms and data structures, but consider visualization more or less as an add-on. Dedicated *graph visualization tools* like Walrus [5], HyperTree [1], or Vizster [37] focus on the implementation of a single visualization approach. *Multiple-view systems* like Improvise [69] or Tableau [4] integrate several visualization approaches, but are often focused on data visualization, rather than graph visualization. More

general *information visualization frameworks* like the InfoVis Toolkit [28] or Prefuse [38] (and recently VTK [71]) provide the platform to develop graph visualization software, but they are not self-contained ready-to-use systems.

Given the wealth of software out there, the reader might ask what makes CGV special or in what aspects does CGV differ from other systems. It is beyond the scope of this work to compare CGV with an exhaustive list of visualization software. Instead, we elaborate on the strengths and weaknesses of CGV compared to selected approaches keeping our focus on graph visualization systems and interactivity.

We decided to compare CGV to four selected systems: ASK-GraphView, Tulip, Pajek, and GUESS. All these systems support the visual exploration of graphs. However, they are very different in terms of system architecture (ASK-GraphView and Pajek: monolithic architecture; Tulip: plug-in system; GUESS: scripting language), implemented views (single view vs. multiple views), and offered interaction.

ASK-GraphView [6] is the direct predecessor of CGV. The system has been developed with a focus on large graphs and, hence, employs sophisticated data structures and data handling mechanisms. A fixed arrangement of several linked views allows for easy graph exploration. Basic search, navigation, and filtering functions are offered to the user.

Tulip [15] is an open source visualization system that is capable of showing different representations of graphs. The system is comprehensive in terms of provided layout and analysis algorithms. It is suited for experts to compare research results or to test new algorithms. Extensions can be easily integrated thanks to Tulip's plug-in mechanism. Tulip is not restricted to exploration tasks, but can also be used to edit a graph (i.e., add/remove nodes and edges).

Pajek [23] is similar to Tulip in that it provides several different layout algorithms. From the number of algorithms implemented in Pajek it is apparent that the system has a clear focus on graph analysis, including clustering and graph partitioning. The visualization facilities include node-link diagrams and simple view navigation.

The GUESS system [12] in its basic version provides a single node-link view of a graph and a command prompt to allow users to manipulate the visual representation in manifold ways. Behind the command prompt, GUESS employs a scripting language called Gython to drive the interaction. The embedded language is the core innovation of the system as it provides a very flexible and capable means to interact with the graph and its representation.

Since visual interaction is the major subject of our work, we will compare the aforemention systems to CGV from that perspective. To that end, we consider a list of interactions, which we derived from [72]:

1. *Select*: Mark something as interesting.
2. *Explore*: Show me something else.
3. *Encode/Reconfigure*: Show me a different representation/arrangement.
4. *Abstract/Elaborate*: Show me more or less detail.
5. *Filter*: Show me something conditionally.
6. *Connect*: Show me related items.
7. *Undo/Redo*: Let me go where I have been already.
8. *Change configuration*: Let me adjust the interface.

These interactions are supported differently by the systems Pajek, Tulip, ASK-GraphView, GUESS, and CGV, as we will see next.

1. Select – All systems provide means for selection of nodes and edges, which are basically implemented as point-and-click interactions.

2. Explore – Pajek, Tulip, ASK-GraphView, and GUESS offer exploration in terms of navigation of the presentation space. Standard zoom and pan interaction are offered. In addition, CGV enhances view space navigation and, additionally, supports navigation in the data space. The pan-wheel in conjunction with the radar view or the locate interaction are useful for exploring the view space, whereas CGV's edge-based traveling is a novel way of navigating the data space, which is not possible in the other systems.

3. Encode/Reconfigure – Pajek offers different layouts and allows for linking of data attributes to visual variables. As mentioned before, Tulip is comprehensive with regard to layouts and graphical encoding. Similarly powerful is GUESS, whose users can flexibly script the visual encoding by using the embedded language. A list of different layout algorithms is offered as well. ASK-GraphView and CGV are not that flexible. They only allow users to choose the node attribute to be visualized via color coding. Nonetheless, alternative arrangements of the graph are offered in the different views of both systems, but CGV provides more choices than ASK-GraphView.

4. Abstract/Elaborate – The GUESS systems allows users to cluster nodes based on different metrics. Clusters can then be made distinguishable by applying different visual encodings (incl. convex hulls). The concept of super nodes that can be expanded or collapsed is not available. It is available in Pajek and Tulip. However, expand and collapse cannot be conducted in the main view by means of direct interaction, but must be performed in separate views. Nodes are not literally expanded or collapsed, but are rather made visible or invisible. In contrast to that, ASK-GraphView and CGV are based on the philosophy of expanding and collapsing nodes on demand. Expand/collapse are visually augmented in both systems by means of animation. Moreover, any view provided by CGV can be used to perform this type of interaction, which is a usability plus compared to the other systems.

5. Filter – Interestingly, neither Pajek nor Tulip offer filtering facilities. ASK-GraphView offers a basic filter slider to focus on nodes with specific properties. GUESS allows for filtering via its scripting language. A programming language allows for very complex filtering conditions and manifold visual encodings of the filter results, which,

	Pajek	Tulip	ASK-GraphView	GUESS	CGV
Select	+	+	+	+	+
Explore	∘	∘	∘	∘	+
Encode/Reconfigure	∘	+	∘	+	∘
Abstract/Elaborate	+	∘	+	∘	+
Filter	−	−	∘	+	+
Connect	−	∘	∘	∘	+
Undo/Redo	−	−	−	−	+
Change configuration	−	∘	∘	∘	+
Edit	∘	∘	−	∘	−

Table 1: Comparison of different graph visualization systems with respect to offered interaction facilities (+, ∘, and − stand for advanced, basic, and limited/no support).

however, must be typed at the command prompt. Yet, by extending the system, helpful user interface elements (e.g., sliders) can be incorporated. CGV provides an advanced filter mechanism that is based on logical AND and OR combinations of basic filters and an intuitive user interface. Filtered data elements are dimmed or omitted automatically.

6. Connect – The connect interaction, which shows the user related items, is more difficult to grasp. Basically, all systems allow users to relate items via their clustering and filtering facilities. Tulip, ASK-GraphView, and CGV have multiple views that, through appropriate linking, can show related views of the same data. CGV provides additional methods: The layout lens is suited to bring related nodes to the viewport and edge-based traveling supports navigation between topologically neighboring nodes that appear distant in the layout.

7. Undo/Redo – Tulip, and ASK-GraphView do not provide a history mechanism. Pajek's log files and GUESS's command prompt offer a way to reuse or edit previously issued commands, which however are not history mechanisms in terms of undoing or redoing interaction. CGV is the only system that offers undo/redo in the classic sense.

8. Change configuration – The views of Pajek, ASK-GraphView, and GUESS are subject to basic reconfiguration such as expand or resize. Tulip uses a mix of docking and freely adjustable windows. The docking framework applied in CGV is more flexible as it allows for arbitrary arrangements, including grouping of views (within the constraint of the docking paradigm) or task-based arrangements of views, which can be stored.

-

The previously made statements are summarized in Table 1. It is worth mentioning that Pajek, Tulip, and GUESS support an additional aspect of interaction, namely graph editing. In this sense, they are not only graph visualization systems, but visual graph editors. This is an aspect that would be interesting to pursue in the future development of CGV. In general, we found that other systems and tools described in the literature do not focus on interaction as much as CGV does. In particular, advanced navigation methods, history mechanisms, view arrangements (e.g., via docking), or combinations of 2D and 3D views are scarce. Moreover, the lack of web-accessibility is a limiting factor of some of the existing software.

7. Evaluation Desiderata

Evaluation of visualization and interaction techniques in information visualization is a challenging task [14, 27]. For systems that incorporate many different visualization and interaction techniques, the challenge is even bigger. This is due to the multitude of parameters that influence the evaluation, a fact that raises questions like: Which parameterization should be tested for which visualization?, Which combinations of visual and interaction methods should be evaluated?, Which tasks and user groups should be addressed?, and so forth. These are questions that we are not ready to address in this writing.

Our current approach is very rudimentary, i.e., "Deploy the system and get feedback". For the continuous evaluation of CGV we are employing a three-fold strategy. (1) Early focus on users, empirical measurements, and iterative design [32]. (2) Expert interviews are conducted to evaluate CGV in the context of real world application scenarios [62]. (3) We provide access to CGV over the Internet and are preparing to collect feedback from a broader audience. This type of evaluation can yield valuable results [42].

7.1. Users Feedback

Next, we discuss feedback collected from early adopters of CGV mainly concerning the user interface and interaction techniques. The different visualizations were not a major issue since they are mainly well-accepted approaches from the literature. The feedback of users influenced the development of the system in several aspects.

Views Arrangement. In the early stages, users complained about the fixed arrangement of views. Even though they were able to resize the views independently, instant maximization of a view or rearrangements were not possible. To overcome these difficulties we incorporated the docking framework as described in Section 5.7.

Parameter Settings. Another aspect was the setting of visualization parameters. As the views in CGV became more and more complex, the number of visualization parameters increased and the panel to set the parameters grew. Users found it difficult to locate important parameters and demanded easy access. This prompted us to incorporate tool bars to allow instant adjustment of the most important parameters on a per view basis (see Section 5.7).

Zoom Direction and Center. Some design flaws were noticed regarding some of the interaction mechanisms offered by CGV. An example is the zoom operation performed via the mouse wheel in the graph view. The direction of zooming was opposite to the intuition of some users. Interestingly, other users had no problems with the zoom function. Only recently, we found evidence that both directions of zooming are appropriate, depending on the mental model of the user [33]. So we added a parameter that allows users to control the direction of zooming operations. Moreover, when zooming in, we considered the current location of the mouse cursor as the center of the zoom operation. Since this was counterintuitive for some users, we introduced a zooming mode that takes the center of the viewport as the center for the zoom operation, rather than the mouse cursor. In the textual tree view, initially we used the mouse wheel for zooming as in the graph view. However, this differs from the standard behavior in classic tree views. Now, users can scroll this view via the mouse wheel, while for zooming the CTRL key has to be held down in addition to rotating the wheel.

-

Further users feedback led to the implementation of the infinitive grid, the pan wheel and the radar view, the preview mode for edge-based traveling, the search box, and the history mechanism mentioned earlier (see Section 5). In summary, user feedback has been incorporated continuously during CGV's development. This supports the thesis that formative testing is indeed *a valuable technique during the development of an information visualization to gain design feedback and fix bugs.* [14].

7.2. Expert Interviews

On a second evaluation stage, we provided CGV to domain experts and asked for their criticism. Besides the general feedback, two specific needs were raised by the experts.

"dIEM oSiRiS" is a joint effort of several disciplines to investigate regenerative systems with the goal to assist the research for a cure of Parkinson's disease and other neuronal defects [64]. The need to store complex filters for later re-use was pointed out to us by researchers from this group. As we described earlier, such functionality is now available in CGV.

Investigators who conduct research related to neuromapping and development atlases of neuronal systems [53] pointed out that filtering based on node and edge attributes is not enough in their domain. They need methods to filter their data according to structural graph properties. These suggestions motivated us to start a project to introduce new tools in CGV for graph motifs finding. Though we were able to generate some initial results, this project is still in its early stage, and we hope to report this in the future.

7.3. Web-Based Evaluation

The third stage in our evaluation strategy is to deploy CGV on the Web and to collect feedback from a broader audience. Currently, we provide access to CGV and a medium size data set for experimentation [59]. We encourage the readers to go to the web site, test our tools, and send us their general feedback. The flexibility of CGV also allows for more controlled experiments. For that purpose we have begun to identify specific aspects of CGV for user testing. We derived, for instance, an applet that focuses on the use of lenses and an applet that solely consists of the Magic eye view. One could also think of two applets that employ the same visual and interaction concepts, but use different parameterizations. We plan to embed these applets in web pages that automatically collect timings and/or success rates and provide an interface to rate user satisfaction. Even though we are aware of the risks of web-based questionnaires, we hope that the feedback collected over the internet reveals hidden issues that otherwise would have remained undiscovered.

8. Conclusions and Future Work

We presented the interactive graph visualization system CGV. We described its views and how they support the visual exploration of graph hierarchies. A particular focus of this work is on interaction. We introduced several novel interaction techniques. They include filter composition, graph lenses, a pan wheel, edge-driven navigation, adjustable parameter settings, and a flexible docking framework. Novel visual cues are used to augment the provided interaction techniques (i.e., the infinite grid and the radar view). A history mechanism is available to allow for undoing and redoing of interaction, which is usually not available in current graph visualization systems.

CGV is a general modular system that is applicable in a variety of scenarios. First real world applications have been in the context of evaluating graph clustering algorithms (see Fig.2) and the exploration of topological structures of neuronal systems (see Appendix A and Fig.12). We are currently experimenting with CGV's filtering and history mechanisms to explore time evolving graph data. The flexibility of CGV has paid off significantly during adaptation to different application scenarios.

When interactivity is the focus of system development, the separation of visual interface and backing computations becomes a necessity. Interaction techniques and their visual augmentation require tight integration with the visualization. Well designed models for coordination and history are required to keep the system consistent.

General questions we want to address in future research are: Is it possible to come up with common interaction patterns? Can user interfaces for visualization parameters be generated automatically from a proper description of the main parameter characteristics? How can the flexible docking mechanism be used to achieve task-based adaptation of the system?

A more specific future objective is the integration of motif detection as indicated in Section 7.2. This requires investigation of aspects related to the specification of motif patterns, the detection of motif patterns in the graph hierarchy, and the visualization and interaction with detected pattern matches.

Improving the history mechanism to support more complex interactions is another aim for future work. An approach is to keep track of parameter adjustments performed via the settings panel (asynchronous parameterization) and interactive changes applied to the dynamic filtering mechanism. To accomplish this we have to extend our implementation in two aspects. First, we need composite commands to encapsulate multiple basic commands. Secondly, we have to face continuous interactions. CGV's current history mechanism reaches its limits when users apply continuous parameter adjustments via sliders or continuous browsing in the view space via the pan-wheel. In these cases, continuous interaction is mapped to a series of discrete commands, all being tracked in the history, which is rather impractical. One possibility to overcome this shortcoming is to aggregate a series of related commands into a single command based on well-chosen timing. In other words, during continuous interaction commands are aggregated. Only if the user pauses for a certain time period during interaction (e.g., to check an intermediate result more closely), we do issue a separate command to the history and begin aggregating anew.

Finally, extension of the proposed interaction mechanisms to navigate large directed graphs requires major breakthroughs in hierarchy generation. This is due to the fact that current methods for undirected graphs are hindered by their inability to handle a sense of topological direction.

References

[1] HyperTree Java Library. `http://hypertree.sourceforge.net`.

[2] JGraphT. `http://jgrapht.sourceforge.net`.

[3] JUNG – Java Universal Network/Graph Framework. `http://jung.sourceforge.net`.

[4] Tableau Software. `http://www.tableausoftware.com`.

[5] Walrus – Graph Visualization Tool. `http://www.caida.org/tools/visualization/walrus`.

[6] J. Abello, F. van Ham, and N. Krishnan. ASK-GraphView: A Large Scale Graph Visualization System. *IEEE Transactions on Visualization and Computer Graphics*, 12(5), 2006.

[7] James Abello. Hierarchical Graph Maps. *Computers & Graphics*, 28(3), 2004.

[8] James Abello, Adam L. Buchsbaum, and Jeffery Westbrook. A Functional Approach to External Graph Algorithms. *Algorithmica*, 32(3), 2002.

[9] James Abello and Jeffrey Korn. MGV: A System for Visualizing Massive Multidigraphs. *IEEE Transactions on Visualization and Computer Graphics*, 8(1), 2002.

[10] James Abello, Hans-Jörg Schulz, Heidrun Schumann, and Christian Tominski. Interactive Poster: CGV - Coordinated Graph Visualization. Poster presentation at IEEE Information Visualization Conference (InfoVis), 2007.

[11] Gregory D. Abowd and Alan J. Dix. Giving Undo Attention. *Interacting with Computers*, 4(3), 1992.

[12] Eytan Adar. GUESS: A Language and Interface for Graph Exploration. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*. ACM, 2006.

[13] Christopher Ahlberg, Christopher Williamson, and Ben Shneiderman. Dynamic Queries for Information exploration: An Implementation and Evaluation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*. ACM, 1992.

[14] Keith Andrews. Evaluating Information Visualisations. In *Proceedings of the Workshop BEyond time and errors: novel evaLuation methods for Information Visualization (BELIV)*. ACM, 2006.

[15] D. Auber. Tulip : A Huge Graph Visualisation Framework. In P. Mutzel and M. Jünger, editors, *Graph Drawing Softwares*, Mathematics and Visualization. Springer-Verlag, 2003.

[16] Michelle Q. Wang Baldonado, Allison Woodruff, and Allan Kuchinsky. Guidelines for Using Multiple Views in Information Visualization. In *Proceedings of the Working Conference on Advanced Visual Interfaces (AVI)*. ACM, 2000.

[17] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1999.

[18] Eric A. Bier, Maureen C. Stone, Ken Pier, William Buxton, and Tony D. DeRose. Toolglass and Magic Lenses: The See-Through Interface. In *Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*. ACM, 1993.

[19] D. Borland and R. Taylor. Rainbow color map (still) considered harmful. *Computer Graphics & Applications*, 27(2), 2007.

[20] Mark Bruls, Kees Huizing, and Jarke J. van Wijk. Squarified Treemaps. In *Proceedings of the Joint Eurographics - IEEE TCVG Symposium on Visualization (VisSym)*. Eurographics Association, 2000.

[21] Christoph Buchheim, Michael Jünger, and Sebastian Leipert. Improving Walker's Algorithm to Run in Linear Time. In *Proceedings of the International Symposium on Graph Drawing (GD)*. Springer, 2002.

[22] Andy Cockburn and Joshua Savage. Comparing Speed-Dependent Automatic Zooming with Traditional Scroll, Pan and Zoom Methods. In *In People and Computers XVII: British Computer Society Conference on Human Computer Interaction*, 2003.

[23] Wouter de Nooy, Andrej Mrvar, and Vladimir Batagelj. *Exploratory Social Network Analysis with Pajek*. Cambridge University Press, 2005.

[24] Mark Derthick and Steven F. Roth. Enhancing Data Exploration With a Branching History of User Operations . *Knowledge-Based Systems*, 14(1-2), 2001.

[25] Reinhard Diestel. *Graph Theory*. Springer, 2005.

[26] Helmut Doleisch, Martin Gasser, and Helwig Hauser. Interactive Feature Specification for Focus+Context Visualization of Complex Simulation Data. In *Proceedings of the Joint Eurographics - IEEE TCVG Symposium on Visualization (VisSym)*. Eurographics Association, 2003.

[27] Geoffrey Ellis and Alan Dix. An Explorative Analysis of User Evaluation Studies in Information Visualisation. In *Proceedings of the Workshop BEyond time and errors: novel evaLuation methods for Information Visualization (BELIV)*. ACM, 2006.

[28] J.-D. Fekete. The InfoVis Toolkit. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis)*. IEEE Computer Society, 2004.

[29] Christiane Fellbaum, editor. *WordNet – An Electronic Lexical*

Database. MIT Press, 1998.

[30] Krzysztof Z. Gajos, Mary Czerwinski, Desney S. Tan, and Daniel S. Weld. Exploring the Design Space for Adaptive Graphical User Interfaces. In *Proceedings of the Working Conference on Advanced Visual Interfaces (AVI)*. ACM, 2006.

[31] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Pattern Elements of Reusable Object-Oriented Software*. Addison Wesley, 1994.

[32] John D. Gould and Clayton Lewis. Designing for Usability: Key Principles and What Designers Think. *Communications of the ACM*, 28(3), 1985.

[33] Philip Grew. Steering Wheel or Driving Wheel: Which Way Is Up? In *Proceedings of the IASTED International Conference Human-Computer Interaction*. ACTA Press, 2008.

[34] Sean Gustafson, Patrick Baudisch, Carl Gutwin, and Pourang Irani. Wedge: Clutter-Free Visualization of Off-Screen Locations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*. ACM, 2008.

[35] Mark Harrower and Sheesley Benjamin. Designing Better Map Interfaces: A Framework for Panning and Zooming. *Transactions in GIS*, 9(2), 2005.

[36] Jeffrey Heer and Maneesh Agrawala. Software Design Patterns for Information Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(5), 2006.

[37] Jeffrey Heer and Danah Boyd. Vizster: Visualizing Online Social Networks. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis)*. IEEE Computer Society, 2005.

[38] Jeffrey Heer, Stuart K. Card, and James A. Landay. Prefuse: A Toolkit for Interactive Information Visualization. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*. ACM, 2005.

[39] Jeffrey Heer and George Robertson. Animated Transitions in Statistical Data Graphics. *IEEE Transactions on Visualization and Computer Graphics*, 13(6), 2007.

[40] Ivan Herman, Guy Melançon, and M. Scott Marshall. Graph Visualization and Navigation in Information Visualization: A Survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1), 2000.

[41] T.J. Jankun-Kelly, Kwan-Liu Ma, and Michael Gertz. A model and framework for visualization exploration. *IEEE Transactions on Visualization and Computer Graphics*, 13(2), 2007.

[42] Ron Kohavi, Randal M. Henne, and Dan Sommerfield. Practical Guide to Controlled Experiments on the Web: Listen to Your Customers not to the HiPPO. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*. ACM, 2007.

[43] Glenn E. Krasner and Stephen T. Pope. A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80. *Journal of Object-Oriented Programming*, 1(3), 1988.

[44] Matthias Kreuseler, Thomas Nocke, and Heidrun Schumann. A History Mechanism for Visual Data Mining. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis)*. IEEE Computer Society, 2004.

[45] Matthias Kreuseler and Heidrun Schumann. A flexible approach for visual data mining. *IEEE Transactions on Visualization and Computer Graphics*, 8(1), 2002.

[46] Bongshin Lee, Catherine Plaisant, Cynthia Sims Parr, Jean-Daniel Fekete, and Nathalie Henry. Task Taxonomy for Graph Visualization. In *Proceedings of the AVI workshop BEyond time and errors: novel evaLuation methods for Information Visualization (BELIV)*. ACM, 2006.

[47] Edward A. Lee. The Problem with Threads. *IEEE Computer*, 39(5), 2006.

[48] T. Moscovich, F. Chevalier, N. Henry, E. Pietriga, and J.D. Fekete. Topology-Aware Navigation in Large Networks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*. ACM, 2009.

[49] Andreas Noack. Energy Models for Graph Clustering. *Journal of Graph Algorithms and Applications*, 11(2), 2007.

[50] Kevin Pulo. Navani: Navigating Large-Scale Visualisations with Animated Transitions. In *Proceedings of the International Conference Information Visualisation (IV)*. IEEE Computer Society, 2007.

[51] E. M. Reingold and J. S. Tilford. Tidier Drawings of Trees. *IEEE Transactions on Software Engineering*, 7(2), 1981.

[52] Donna L. Gresh Robert Kosara, Helwig Hauser. An Interaction View on Information Visualization. In *State-of-the-Art Proceedings of Eurographics (EG)*. Eurographics Association, 2003.

[53] Oliver Schmitt, Jan Modersitzki, Stefan Heldmann, Stefan Wirtz, and Bernd Fischer. Image Registration of Sectioned Brains. *International Journal of Computer Vision*, 73(1), 2007.

[54] Ben Shneiderman. Direct Manipulation: A step beyond programming languages. *IEEE Computer*, 16(8), 1983.

[55] Ben Shneiderman. The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. In *Proceedings of the IEEE Symposium on Visual Languages*. IEEE Computer Society, 1996.

[56] Diane Tang, Chris Stolte, and Robert Bosch. Design choices when architecting visualizations. *Information Visualization*, 3(2), 2004.

[57] Conrad Thiede, Georg Fuchs, and Heidrun Schumann. Smart Lenses. In *Proceedings of the International Symposium on Smart Graphics*. Springer, 2008.

[58] J. J. Thomas and K. A. Cook. *Illuminating the Path: The Research and Development Agenda for Visual Analytics*. IEEE Press, 2005.

[59] Christian Tominski. CGV – Coordinated Graph Visualization, 2008. `http://www.informatik.uni-rostock.de/~ct/CGV/CGV.html`.

[60] Christian Tominski, James Abello, and Heidrun Schumann. Two Novel Techniques for Interactive Navigation of Graph Layouts. Poster presentation at Eurographics/IEEE Sysmposium on Visualization (EuroVis), 2009.

[61] Christian Tominski, James Abello, Frank van Ham, and Heidrun Schumann. Fisheye Tree Views and Lenses for Graph Visualization. In *Proceedings of the International Conference Information Visualisation (IV)*. IEEE Computer Society, 2006.

[62] Melanie Tory and Torsten Möller. Evaluating Visualizations: Do Expert Reviews Work? *IEEE Computer Graphics and Applications*, 25(5), 2005.

[63] B. Tversky, J. B. Morrison, and M. Betrancourt. Animation: Can It Facilitate? *International Journal of Human-Computer Studies*, 57(4), 2002.

[64] Adelinde M. Uhrmacher, Arndt Rolfs, and Jana Frahm. DFG Research Training Group 1387/1: dIEM oSiRiS - Integrative Development of Modelling and Simulation Methods for Regenerative Systems. *it - Information Technology*, 49(6), 2007.

[65] Robert van Liere and Wim C. de Leeuw. Graphsplatting: Visualizing graphs as continuous fields. *IEEE Transactions on Visualization and Computer Graphics*, 9(2), 2003.

[66] Jarke J. van Wijk and Wim A. A. Nuij. A Model for Smooth Viewing and Navigation of Large 2D Information Spaces. *IEEE Transactions on Visualization and Computer Graphics*, 10(4), 2004.

[67] Fernanda B. Viegas, Martin Wattenberg, Frank van Ham, Jesse Kriss, and Matt McKeon. ManyEyes: A Site for Visualization at Internet Scale. *IEEE Transactions on Visualization and Computer Graphics*, 13(6), 2007.

[68] M. Ward and J. Yang. Interaction Spaces in Data and Information Visualization . In *Proceedings of the Joint Eurographics - IEEE TCVG Symposium on Visualization (VisSym)*. Eurographics Association, 2004.

[69] Christopher Eric Weaver. *Improvise: A User Interface for Interactive Construction of Highly-Coordinated Visualizations*. PhD thesis, University of Wisconsin-Madison, 2006.

[70] G.J. Wills. Selection: 524,288 Ways to Say "This is Interesting". In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis)*. IEEE Computer Society, 1996.

[71] Brian Wylie and Jeffrey Baumes. A Unified Toolkit for Information and Scientific Visualization. In *Proceedings of Visualization and Data Analysis (VDA)*. SPIE, 2009.

[72] Ji Soo Yi, Youn ah Kang, J.T. Stasko, and J.A. Jacko. Toward

a Deeper Understanding of the Role of Interaction in Information Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 13(6), 2007.

A. Case Study: Topology of Neuronal Systems

CGV has been used to explore the topological structure of neuronal systems. In particular, the researchers goal is to create atlases of rat brains. For that purpose, rat brains are dissected and mapped, for instance with the help of computer vision methods [53]. The atlases are organized in a hierarchical fashion, starting with the whole brain, splitting into its left and right parts, and going further down to more and more specific areas of the neuronal system. This hierarchy is constantly updated according to new findings published in medicine articles. Additionally, this hierarchical structure of rat brains is manually annotated with knowledge about functional dependencies between regions of the brain. That is, if an experiment reveals a relationship between regions A and B, then an edge (A, B) is inserted. These edges can be directed or undirected, depending on the experimental results. It is also possible that an edge connects a rather coarse brain region (in upper parts of the hierarchy) with a quite specific one (in lower parts in the hierarchy). Such edges give researchers hints as to where further experiments could reveal more interesting and more specific results. Patterns of functional dependency play also a role. The researchers are mainly interested in motifs of 3 to 5 nodes, where triads are particularly important.

Next, we describe how CGV can be used to explore the aforementioned data. Currently the data contains about 3000 nodes and roughly the same number of edges, but these numbers will increase in the future as more medical results are entered in the database. Fig. 12(a) shows an initial representation of the data. The user has folded away most views because he is mostly interested in the data's hierarchical structure. The symmetry of the left part and the right part can be seen from the hierarchy view. The colors are pre-defined and have a specific meaning to the researcher. He can use the search box to find a node whose label contains the term `Unclassified`. After typing three letters and performing a single click, the hierarchy view updates itself automatically to focus on the node with the label `Unclassified_cell_group_L` (Fig. 12(b)). Since this might identify a part of the brain that needs further research, the user decides to fold the hierarchy view and switch to the graph view to see the node's relations to other parts of the brain. The graph view has already performed layout computations and set the focus on the corresponding cluster automatically. By looking at the edges, the user can see that the node is related to other parts of the same cluster and to nodes in other clusters, which however are off-screen (Fig. 12(c)). In Fig. 12(d), he then uses the composite lens to create a local overview of the nodes connectivity. From this local overview he decides that the node labeled `Cerebral_cortex_L` is worth further investigation. In order to navigate to that node, the user releases the lens and uses edge-travelling. The navigation to the selected node is augmented with an animation, which also slightly zooms out to create a better overview (Fig. 12(e)). During zooming, the infinite grid serves as a spatial reference to help the user judge the distance travelled. After the node has been reached, the researcher expands it to see details (Fig. 12(f)). However, at this point, the connection to the "Unclassified" node is no longer apparent. To really focus on that connection, nodes can be selected as shown in Fig. 12(g). To shortcut the navigation back to the "Unclassified" node, the researcher performs the selection in the textual tree view (left part of Fig. 12(h)). By zooming out, a full screen overview of the connectivity of the selected nodes has been created. The researcher can now decide whether it is worth to experimentally investigate this connectivity in more detail or can continue his exploration.

B. Case Study: Exploration of "WordNet" data

In a second usage example, we explore the well-known "WordNet" data [29]. This data has about 100k nodes and 150k edges. CGV automatically computes a hierarchy tree from the big graph. The user can then use the hierarchy for interactive information drill down. We exemplify this procedure in the following.

Fig. 13(a) shows an initial macro-view of the underlying "WordNet" data. The textual tree view represents the hierarchical structure and the splat view gives a coarse overview. Since the main graph view is cluttered with many nodes and edges, the user performs a filter operation with the slider to reduce the number of nodes. Fig. 13(b) and (c) show the effects of dimming filtered nodes and of omitting them, respectively. Based on the filtered view the user decides to start his exploration with the node selected in Fig. 13(a). He disposes off the filter to have more screen space for the main graph view. After a series of expand operations, the user reaches a node labelled `academic, degree/degree`, which draws his attention. There are several related nodes in the proximity. To get a better overview, the user applies the layout lens to attract the related nodes as shown in Fig. 14(a). As he moves the lens he recognizes that the neighbors accumulate in the lens center and a yellow node appears at the lens boundary (Fig. 14(b)). Because the nodes accumulate even when applying the composite lens, which should spread them, the yellow node must be very far away in the layout. The user decides to explore this distant node in more detail. In order to get an impression of where the node is located, the preview mode of edge-based travelling is used. As shown in the sequence in Fig. 14(c), several zoom levels are passed before the preview is reached and the travelled edge is centered on the screen. At this point, the user can decided to click the edge again, to actually move to the yellow node.

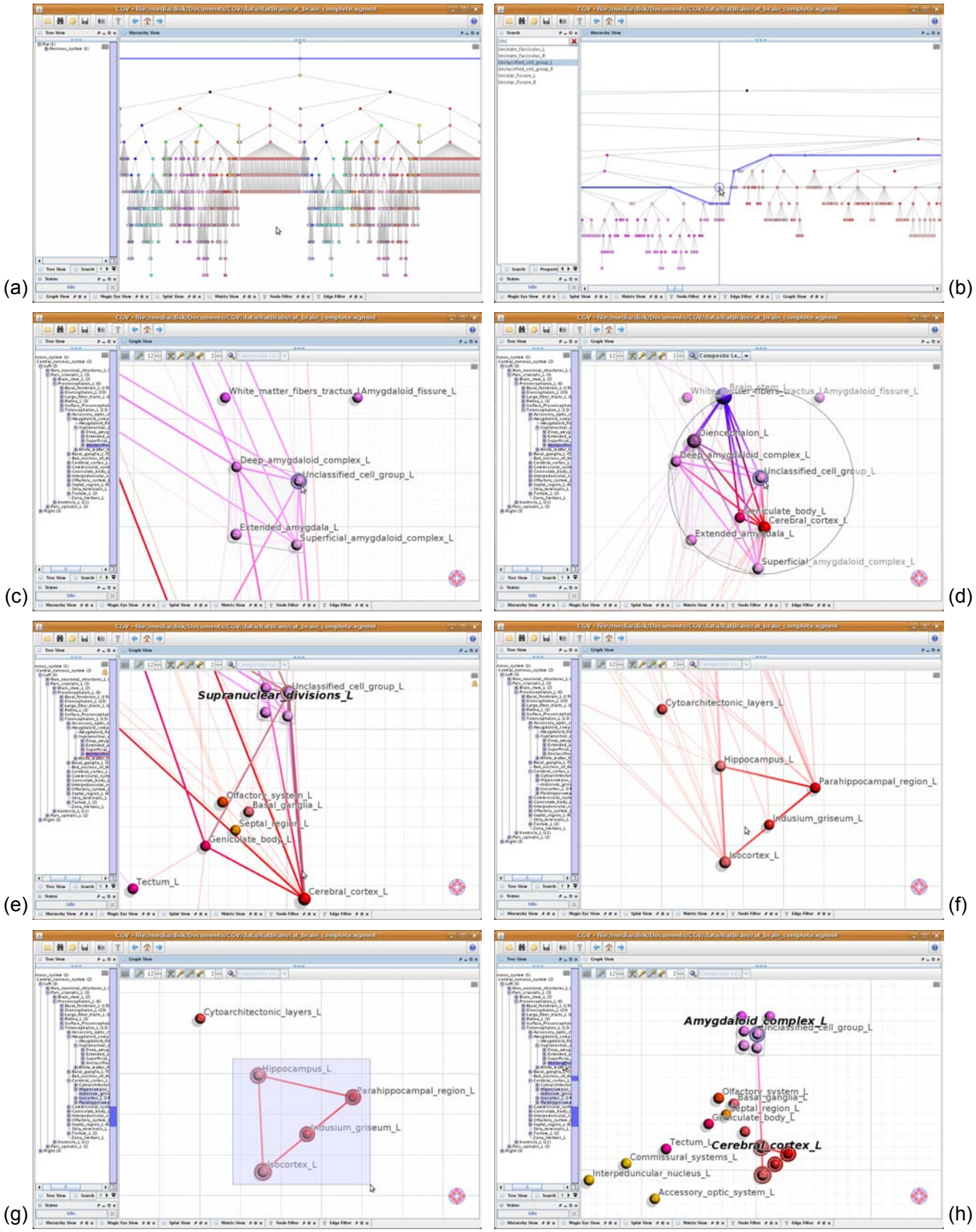

Figure 12: Exploration of the topological structure of rat brain.

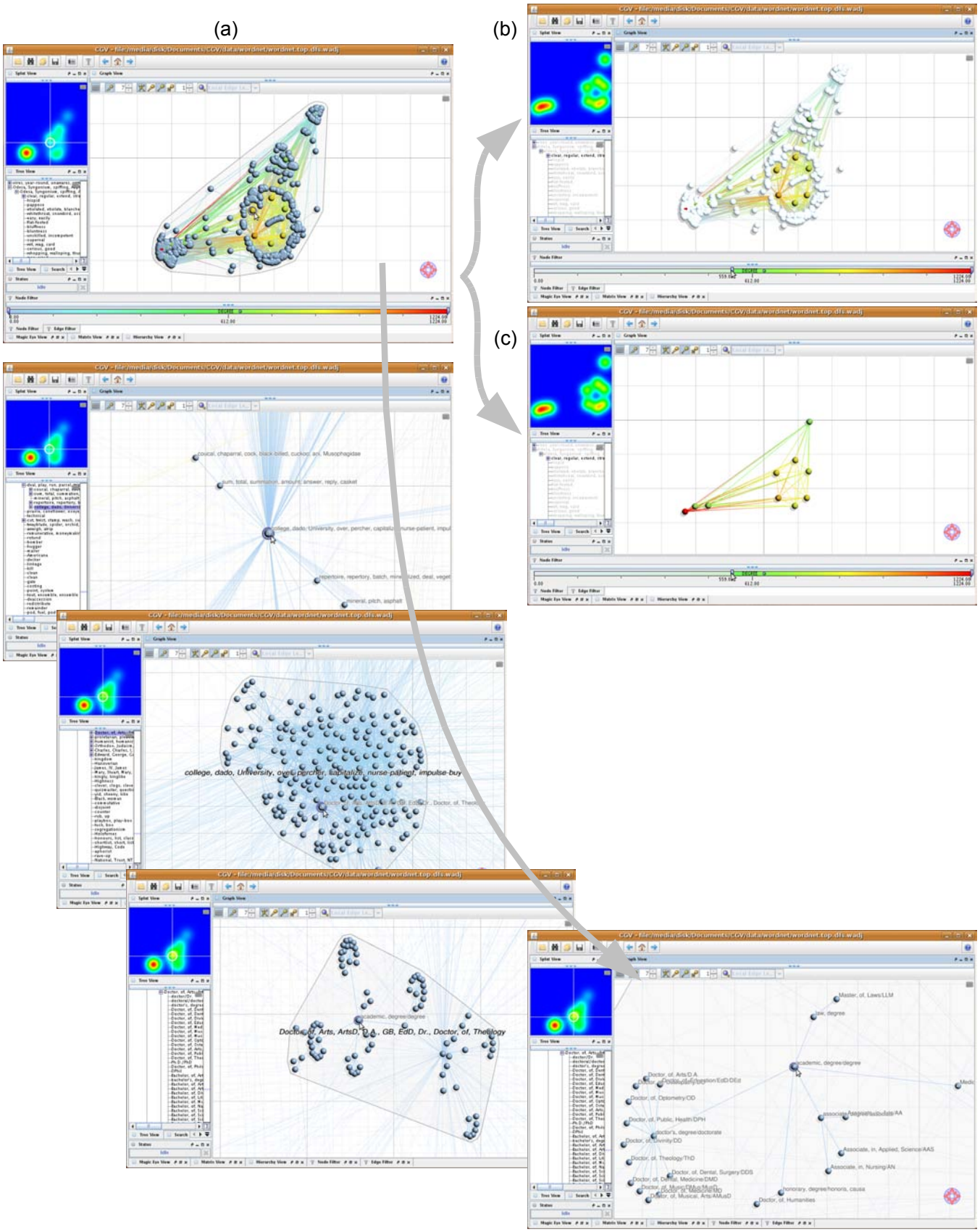

Figure 13: Exploration of “WordNet” data.

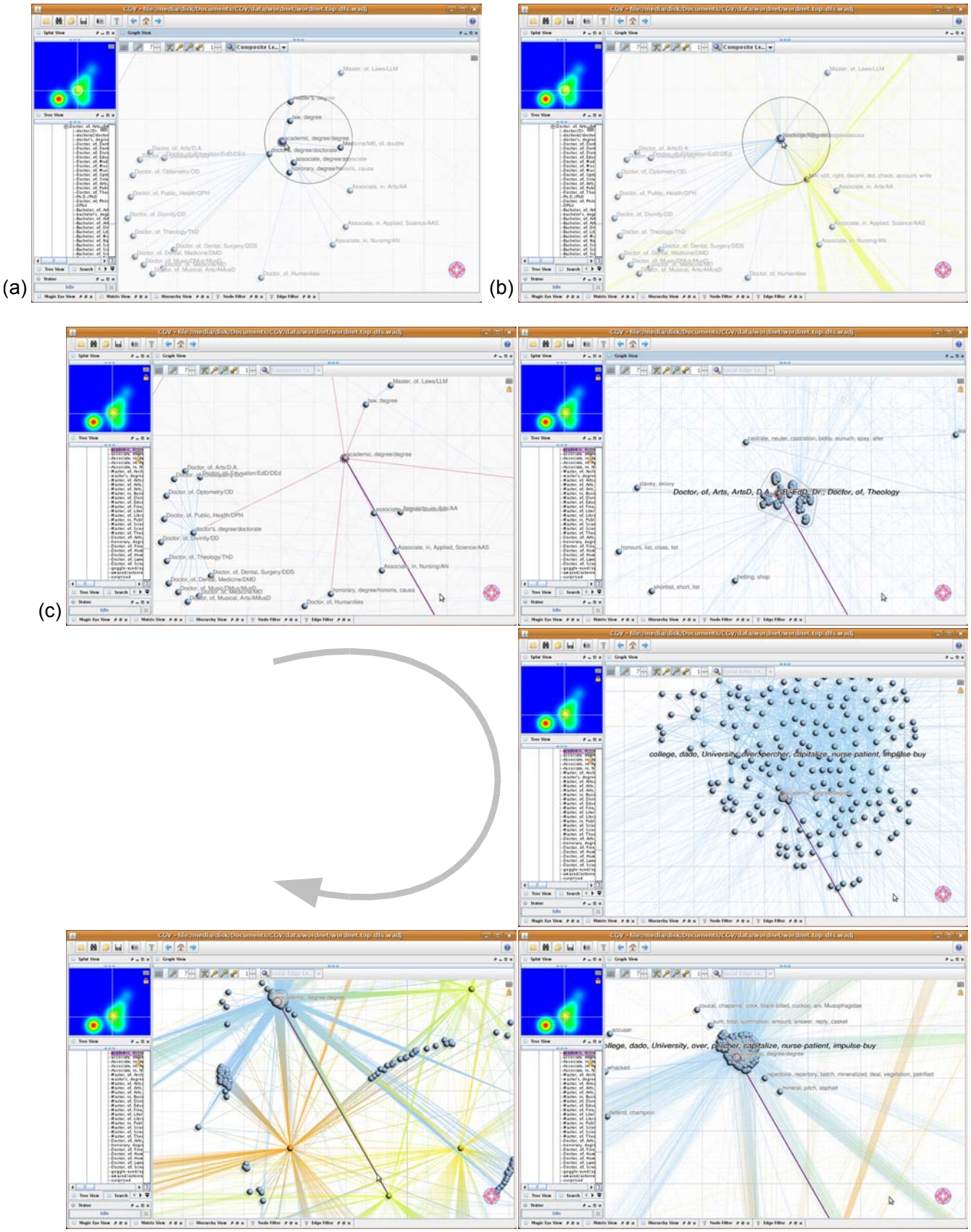

Figure 14: Exploration of "WordNet" data continued.