# A Framework for Visual Data Mining of Structures

**Hans-Jörg Schulz** **Thomas Nocke** **Heidrun Schumann**

Department of Computer Science,
University of Rostock,
18051 Rostock, Germany,
Email: {hjschulz, nocke, schumann}@informatik.uni-rostock.de

**Abstract**

Visual data mining has been established to effectively analyze large, complex numerical data sets. Especially, the extraction and visualization of inherent structures such as hierarchies and networks has made a significant leap forward. However, it is still a challenging task for users to explore explicitly given large structures. In this paper, we approach this task by tightly coupling visualization and graph-theoretical methods. Therefore, we investigate if and how visualization can benefit from common graph-theoretical methods – mainly developed for the investigation of social networks – and vice versa. To accomplish this close integration, we introduce a design of a general framework for visual data mining of complex structures. Especially, this design includes an appropriate processing order of different mining and visualization algorithms and their mining results. Furthermore, we discuss some important implementation details of our framework to ensure fast structure processing. Finally, we examine the applicability of the framework for a large real-world data set.

## 1 Introduction

Visual data mining (VDM) has been proven to be an effective method to explore large data sets. It combines automated mining algorithms with visualization techniques. A variety of powerful methods and tools (e.g. the InfoVis (Fequete 2004) and the Prefuse (Heer, Card & Landay 2005) Toolkit and the systems Polaris (Stolte, Tang & Hanrahan 2002), Spotfire (Ahlberg 1996) or Visage (Roth, Lucas, Senn, Gomberg, Burks, Stroffolino, Kolojejchick & Dunmire 1996)) have been developed in the last few years. These tools combine linked views on the data with a high amount of interactivity, enabling users to switch quickly between automated and visual methods. Therefore, they integrate mining methods to explore numerical data from a variety of research areas, for instance AI, statistics and KDD. These methods can extract structures that are inherent in the data (e.g. hierarchical clustering). Furthermore, a number of visualization methods have been developed and integrated to visualize such abstract data as well as structures, gathered by the VDM process or already given with the data set. Examples for such structures are web link graphs and chemical molecule bonds. Another prominent example for such structures are social networks that apply methods to analyze social structures, i.e. to identify central nodes that can be understood as essential for the entire data set.

In this paper we want to discuss the question, if and how calculation methods from graph theory can be employed to essentially enrich VDM tools to explore structures. Our intention is to design a uniform framework that integrates a variety of well-known graph-theoretical and visualization methods for structures. For this purpose, dependencies of such methods have to be considered to design an appropriate control flow.

Up to now, the integration of graph-theoretical methods into VDM environments has not been in the research focus. A main reason for this is the high complexity of these algorithms that does not allow an interactive linking and brushing in VDM environments. To achieve this goal, special effort needs to be done.

Ankerst classifies current visual data mining approaches into three categories (Ankerst 2001). Methods in the first group apply visualization techniques independent of data mining algorithms. The second group uses visualization in order to represent patterns and results from mining algorithms graphically. The third category tightly integrates mining and visualization algorithms in such a way that intermediate steps of the mining algorithms can be visualized. In our approach we focus at the second level, separating the mining process into two parts:

1. execute time-consuming (automatic) algorithms and store their results, and then
2. enable users to do an interactive exploration of the structures in real-time, combining different visualization and less time-consuming graph-theoretical measures.

Although separating the time consuming execution of certain algorithms from the VDM process, performance issues are still of high relevance. Thus, efficient data structures and access mechanisms - managing both graphs and trees - are of high benefit for the interactive VDM (see e.g. (Fequete 2004)). In our framework implementation, we developed mechanisms that allows efficient storage of both structures and structural measures and algorithm results.

The paper is organized as follows: first we outline the background of graph-theoretical algorithms, visualization methods for structures and inspiring application areas (section 2). Afterwards, we discuss graph-theoretical methods suited for VDM, which especially includes their interaction with visualization techniques in section 3. Then, in section 4, we introduce our framework for VDM of structures. This includes the development of a general design and the discussion of internal data structures and their performance for different graph theoretical measures and algorithms. Afterwards, we discuss challenges of

our approach and demonstrate its application to real-world data sets in section 5. Finally, we conclude the paper and discuss future work in section 6.

## 2 Background

On the one hand, in the last few years visualization of large structures, especially of trees and graphs, has been remarkably improved. Visualization techniques enable users to interactively explore complex structural relationships between the information objects. Well-known examples for hierarchy representations are (Lamping, Rao & Pirolli 1995), (Robertson, Mackinlay & Card 1991), (Shneiderman 1992), (Granitzer, Kienreich, Sabol, Andrews & Klieber 2004) and for networks examples are (Tollis, Eades & di Battista 1999) and (Brandes & Corman 2002).

A major challenge in this context is an intuitive navigation through large data sets to quickly find interesting patterns while preserving orientation. Therefore, focus+context techniques on structures have been developed (see e.g. (Gansner, Koren & North 2004), (van Ham & van Wijk 2004)).

A further challenge is the amount of data to be processed. Methods to explore and visualize huge structures that do not even fit in memory have been developed (e.g. (Abello, Finocchi & Korn 2001), (Abello & van Ham 2004)). Here, to ensure interactive data exploration, mechanisms that decide to precompute long-lasting algorithms needed to be developed.

On the other hand, there is a variety of automatic methods introduced by graph theory to explore structures. General work has been done to determine the complexity of graph-theoretical algorithms and to estimate their efficiency and effectivity for practical data sets (e.g. (Valiente 2002)). Furthermore, these methods have been applied and refined for practical application fields, such as social sciences (e.g. to detect community structures within social and biological networks (Girvan & Newman 2002)) and biotechnology (e.g. the usage of generalized interval graphs to solve the physical mapping problem that occurs when sequencing fragments of DNA (Zhang 1994)).

Moreover, there are a few approaches to apply graph-theoretical measures to parameterize visualization and vice versa. For instance, van Ham and van Wijk (van Ham & van Wijk 2004) use hierarchical clustering on graphs and represent nodes in the focus in another hierarchy level than nodes in the context. Other examples are (Abello & van Ham 2004), (Frischman & Tal 2004) and (Gansner et al. 2004).

However, a systematic approach that integrates measures and algorithms of graph theory with interactive visualization methods is still missing. In fact, this can be very helpful to support VDM of structures, for instance to select and parameterize tree and graph visualization by graph measures. Therefore, nodes of high connectivity or of other specific values of interest can be laid out into the focus. Moreover, a tree visualization technique resp. a graph visualization technique can be chosen to represent a structure in dependency of its similarity to a tree. If there are only a few edges to be deleted from a graph to form a tree, a tree visualization technique can be a good choice to visualize this graph, representing the non-tree edges in another way (see figure 1 right and (Fekete, Wang, Dang, Aris & Plaisant 2003)). If, on the other hand, the graph is less similar to a tree, tree visualization techniques are not appropriate (see figure 1 left), and a default graph-drawing technique is the better choice.

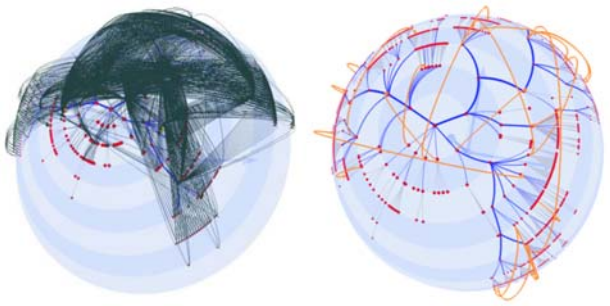

Figure 1: Networks represented by the tree visualization technique MagicEyeView (Kreuseler & Schumann 2002) with a large (left) and a small (right) number of non-tree edges [images taken from (Voigt 2001)].

## 3 Algorithmic mining techniques for complex structures

Even though the visualization is a powerful and important part within the concept of VDM, each visualization technique has its limits on how much data it can possibly display. This most critical problem occurs rather often when analyzing real-world data, but it can be overcome to some extent by appropriate information-hiding, brushing or focus+context techniques embedded within the visualization used. To parameterize those techniques, further information about the given data set can be computed by a thorough algorithmic pre-processing:

- irrelevant data, like statistical outliers that can be hidden
- somehow "important" data that needs to be emphasized
- bits of data that are very similar and can be clustered

The field of graph theory already provides a wide variety of such mining techniques (i.e. finding maximum cliques, shortest paths or calculating modular decompositions). Different domains, like the theory of social networks, the so-called *web structure mining* that is used by search engines throughout the WWW or the bio-chemical analysis of protein structures, supply further methods for analyzing large amounts of structured data. Roughly, these methods can be divided in three categories which can be subtle interrelated or simply used one by one if needed:

- structural measures that capture important attributes of the graph (the list in section 3.1 is based upon an overview given in (Brandes & Wagner 2003)),
- clustering algorithms to decompose large structures,
- methods for graph matching to identify and locate substructures of interest.

Additionally to these three categories, efficient approaches to automatically preselect well-fitting methods can help to maintain a comprehensive overview about the huge number of applicable graph algorithms (section 3.4).

### 3.1 Structural measures

Structural measures can be computed locally (separately for each node) or globally (for an entire graph or subgraph). Local measures of interest are i.e. the

following centrality measures, which can be understood as an index of how "important" a node is (The examples after the formal description of each measure refer to figure 2):

- The **node degree**, that returns the number of incident edges for a node $v$ (i.e. $\deg(c) = 3$).
- The size of the **k-neighborhood** $|N_k(v)|$, that equals the number of nodes within distance $\leq k$ from a given node $v$ (i.e. $|N_1(c)| = 3, |N_2(c)| = 5, |N_3(c)| = 6$).
- The summed up lengths of all shortest paths from a node $v$ to every other node yield the **closeness** of that node
  (i.e. $\text{cls}(c) = 1 + 1 + 1 + 2 + 2 + 3 = 10$).
- The maximum length of all shortest paths from a node $v$ to every other node equals its **eccentricity**
  (i.e. $\text{ecc}(c) = \max(1, 1, 1, 2, 2, 3) = 3$).
- The **node betweenness centrality** of a node $v$, which is defined as the number of all shortest paths that pass through $v$ (i.e. for $c$: 4 from $a$ and $b$, 2 from $d$, $e$, $f$ and $g$, that sums up to 16).

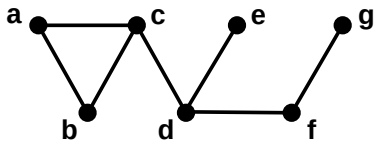


Figure 2: An example graph G(V,E).

The following similarity measures on the other hand can be used for graph clustering or within the layout algorithm to position comparable nodes closer together:

- The **connectivity** of the nodes $u$ and $v$ is the minimum number of edges that have to be removed from the graph in order to separate both nodes in a way that no path between them exists (i.e. $\text{conn}(a, b) = 2$).
- The **dependency** of node $u$ from node $v$ returns the number of shortest paths originating in $u$ and passing through $v$ (i.e. $\text{dep}(c, d) = 3$ and $\text{dep}(d, c) = 2$).

Since different concepts of centrality and similarity stand behind those measures, their outcome often differs in many ways, as can be seen when comparing connectivity (a symmetric measure, since $\text{conn}(u, v) = \text{conn}(v, u)$) with dependency (usually asymmetric, except for some graph classes like circles or cliques). The user must be aware of these differences at all times and should choose the most suitable and expressive measure to model his special goal of analysis.

While these local measures are available only for single nodes, the more general global measures give an overall view of the structure. The simplest form of a global measure is of course the average of a local measures (i.e. the average node degree) that can easily be computed. Other global measures are:

- The **diameter** $diam(G)$ of a graph $G(V, E)$, which equals the largest eccentricity value, or the **radius**, which equals the smallest eccentricity value. (i.e. $\text{diam}(G) = 4$, $\text{rad}(G) = \text{ecc}(d) = 2$)
- The **compactness** or **density** of a graph that provides information about how many of all possible edges are actually present. (i.e. $G$ got 7 out of 21 possible edges)
- A **treelikeness-value** can be computed to obtain a measure for structural resemblance with a tree (a graph without any induced circles). Among many existent treelikeness measures, we introduce an adaption of this term that is optimized for the use within the visualization process: a graph is called $(p, k)$-treelike, if it has no more than $k$ cross-edges and the fraction of cross-edges with respect to all edges is less than the percentage $p$.

Such global measures can be very useful for the determination of an appropriate visualization method: i.e. an underlying treelike structure with only a few crossedges can be identified as such by its treelikeness-value and hence laid out with a hybrid approach as described in section 2. This approach generalizes the ideas introduced in (Fekete et al. 2003), where a tree visualization is extended in a similar way.

### 3.2 Graph clustering

Over the years different flavors of clustering have been developed and evolved further. The clustering techniques that are most often used for the purpose of VDM are the hierarchical clustering algorithms. They do not only yield a clustering of a desired granularity but also a way to explore the data set via browsing the clustering's dendrogram (Herman, Marshall & Melançon 2000). Hierarchical clusterings can be computed either bottom-up by combining similar elements (normalized-association-method (Shi & Malik 1997), single- or average-linkage-method,...) or top-down by separating elements that differ (normalized-cut-method, edge-betweenness-centrality-clustering,...). Hence for both approaches, similarity or distance measures need to be computed beforehand.

One way to circumvent this need is the use of graph decompositions, which also results in a hierarchical graph partition. They work directly on the graph's structure without any additional measures needed and can usually be computed in linear time. Examples are: modular-decomposition, k-core-decomposition (Batagelj, Mrvar & Zaveršnik 1999) or decomposition through distance-k cliques (Edachery, Sen & Brandenburg 1999).

### 3.3 Graph matching

The search for special structures within the data set is a tedious task that is difficult to automate. Several different kinds of graph matching can be used:

- The **exact graph matching** searches for a subgraph that is identical to a specified pattern (Subgraph Isomorphism Problem)
- The **inexact graph matching** searches for a subgraph that is as similar as possible to a specified pattern.
- The search for the largest given configuration, i.e. the largest clique or the longest path.
- The detection of the most frequent subgraph of given minimum size.

Since all of these matching problems are quite complex from an algorithmic point of view, mostly heuristic approaches are used to find approximate solutions (Bunke 2000).

### 3.4 Semi-automated technique selection through metadata

To achieve a certain mining goal, different algorithmic methods can be applied. Usually some of them fit a particular case better than others. To determine suitable techniques, metadata describing the overall structure can be used to derive helpful indications on which method to employ.

An example would be the choice of fitting runtime-efficient algorithms depending on the graph's overall structure. As already mentioned, many of the described graph theoretical methods are painfully slow due to their polynomial or even exponential runtime complexity. An algorithm is usually considered to be efficient on very large data sets, if its complexity is subquadratic. The lack of efficient algorithms results in intolerable high computation times and prevents interactive techniques (*time bottleneck*). But in case the data set fulfills certain structural constraints, efficient algorithms do exist for most of the above presented problems (graph matching, clustering, decomposition, etc.) Since it is widely known that academic worst-case-constructions occur rather seldom in real-world-scenarios, some of the desired constraints are virtually always fulfilled.

An example would be the *sparseness* of a graph, which means that the number of edges is much less than the possible number of edges within the graph. Most graphs from different areas are sparse, e.g.:

- A biochemical molecule must be sparse, since every atom can have only a small number of chemical bonds.
- A social network is usually sparse, because a person normally does not have some hundred acquaintances.
- A large website rarely links from each hypertext document to every other document, as well as scientific papers do not cite every other paper in their field and vice versa.

Therefore, algorithms can be optimized to take advantage of the sparseness and compute their results in less time. An example for such an optimized algorithm is the method to determine the node-betweenness-centrality as described in (Brandes 2001). Another example is the $k$-core-clustering (Batagelj et al. 1999) that decomposes the data set within a linear timebound with respect to the number of edges:

- In the worst case, the graph $G(V, E)$ features all of its possible edges $|E| = |\wp_2(V)| = \frac{1}{2} \cdot |V| \cdot (|V| - 1)$ and has therefore a quadratic runtime bound with respect to the number of nodes.
- In the average and more practical case, the graph is usually sparse and contains only a small fraction of its possible edges. So the runtime complexity will be subquadratic (in terms of the size of the node set) and thus acceptable.

Besides the already provided global structural measures like density or treelikeness (see section 3.1), other structural descriptors can be useful:

- Testing whether a directed graph is **acyclic** can lead to very efficient algorithmic solutions to many NP-complete graph problems that are hard to solve on arbitrary graphs. This test runs in linear time and tries to sort the data set topologically. If this succeeds, the resulting topological ordering can be used as input for fast algorithms that have been especially adapted for this case.
- Determining the **data relationship** (Bertin 1981) that gives an impression of how the overall structure is organized: linear, circular, hierarchical, etc.

These descriptors can also be used to select and parameterize an appropriate visualization technique that can be especially suited to display exactly the described kind of a structure. An example for this method would be the graphs shown in figure 1: contrary to the right part of figure 1, the treelikeness-value of the graph on the left side is obviously out of the range and the MagicEyeView-technique actually not applicable.

Additionally, certain graph classes can even be visualized in very special manners. For instance, if a graph is detected to be an interval graph, it can be displayed as an intersection model consisting of intervals on a straight line. To view a graph as such an intersection model is quite common in genetic engineering and computational biology. Furthermore many algorithmic problems can be efficiently solved on interval graphs. Hence a set of detection-procedures for certain graph classes of interest could further yield useful hints for the choice of a well-suited visualization and speed up the mining process if tailor-made implementations for the detected graph class are provided.

## 4 A general framework for Visual Data Mining in complex structures

### 4.1 Design criteria

Designing a visualization or VDM framework is a sensible process. Many decisions made in early development stages are complicated to redo. Furthermore, a variety of backgrounds with varying data sets and tasks require varying software architectures. In the following, we list five main design criteria for a VDM framework for structures:

- **Generality**
  - Adaptability to ***different application backgrounds*** (e.g. social networks, organic chemistry),
  - Scalability to ***various users*** with varying background knowledge,
  - ***Modular design*** allowing to plug in any kind of visualization techniques and mining operators on structures
- **Flexibility**
  - Flexible ***control mechanisms*** to select, connect and parameterize measures, mining algorithms and visualization techniques on structures (e.g. using scripts, or interactively using menus or data flow charts),
  - ***Visual queries*** with a direct visual feedback,
  - ***Support to derive additional data*** to gain a deeper insight into data features (e.g. by extracting relevant substructures)
- **Usability**
  - ***Data abstraction*** to get easy access to different kinds of structure data sources independent of their internal and external storage format,
  - ***Acceptable reply times*** of calculations (approximation techniques might need to be considered in case of unfavorable runtime complexities or a low upper time bound given by the user),

  - ***Intuitive means of interacting*** with even complex mining methods

- **Efficiency**
  handle fairly large data sets and avoid screen, storage space and temporal bottlenecks
  - ***Memory Efficiency*** through smart data structures as described in 4.3.
  - ***Runtime Efficiency*** by decoupling the interactive parts of the VDM-process from the non-interactive ones as discussed in 4.2.
  - ***Screen Efficiency*** to effectively apply the whole screen space displaying large structures

- **Task orientation**
  Can the user fulfill all tasks to gain the exploration target? This includes a variety of paradigms, including the following:
  - Focus+context
  - Overview+detail
  - Brushing
  - History (especially Undo and Redo)
  - Sorting and filtering
  - Zooming.

However, it is not reasonable to design a VDM framework that is applicable for all kinds of possible applications and tasks. Thus, we design a general architecture for the VDM of structures that can be easily extended by any measures and methods. Therefore, in the following section, general modules in the field of structure exploration and their processing will be introduced in an abstract scheme. These modules are containers for measures, visual and non-visual mining methods as well as for units supporting general tasks such as dynamic queries or history.

### 4.2 Conceptual foundation

In the following, we introduce our design of a VDM-framework that consists of several different functional modules:

- interfaces for user interaction before and after the extensive mining operations,
- a preprocessing unit and a unit to compute structural descriptors,
- the algorithmic kernel that does the mining and lays out the data for its graphical representation.

Thus it is possible to extract the complex algorithmic kernel to do extensive calculations without the need for user-input on different, eventually faster machines, while the user interaction before and afterwards is done within the framework itself. This is the only way to efficiently process the needed graph theoretical algorithms, since one cannot work around the fundamentals of complexity theory. A schematic overview of the framework is depicted in figure 3, where the several fragments are colored according to their function within the overall VDM-process. For its modular design as required by the design criteria in section 4.1, the fragments can be extended with different visualization techniques and algorithmic modules, to realize their function in detail. The fragments provide the following functionality:

- During the **initial interaction**, the user specifies additional properties of the structure that are not explicitly included in the data set. Here the user should also be able to roughly parametrize upcoming algorithmic computations by setting upper runtime bounds and the like.

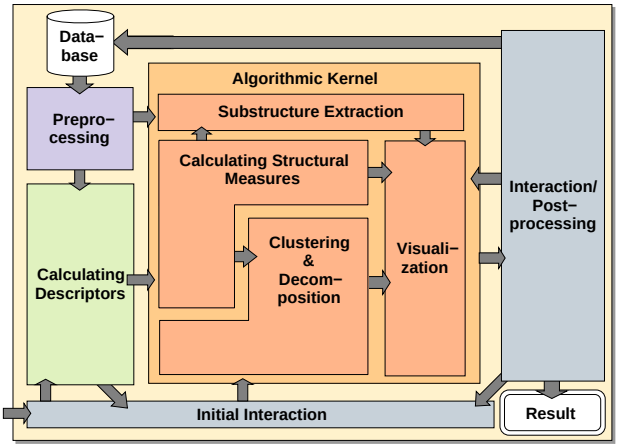


Figure 3: Our general VDM-framework design.

- The **preprocessing** can be used for cleansing and filtering the data.
- The **calculation of descriptors** tries to gather enough metadata as discussed in section 3.4 enabling the user to determine fitting mining and visualization techniques.
- The **algorithmic kernel** does the actual work of calculating additional data, which is one of the crucial features a VDM-framework must fulfill: computing measures (like those in section 3.1), extracting substructures, clustering or decomposing the graph (see section 3.2) and finally calculating a graphical layout for the resulting data.
- The **interaction** on the gained graphical representation is used for the actual visual exploration of the data set. Here some post-processing can be done to further manipulate and query the data set interactively through the visualization and to write back those changes to the data base.

For each of these computational steps, user defined modules can be plugged into the framework and executed in the given order. Usually, the VDM-process based on this architecture starts off by determining promising mining methods through analyzing the automatically computed descriptors and taking the user's goals of analysis into account. Afterwards, the chosen techniques will be employed upon the data and their results will be stored for further visual investigation later on. Depending on how the results are structured, an appropriate visualization technique is selected and used to generate an interactive overview of the outcome that can be graphically explored. As demanded by the design criteria, a wide variety of navigational elements, filtering and searching techniques can be provided through a standardized user interface that applies to all modules. Figure 4 shows a detailed view on the framework with several representative modules plugged-in to illustrate typical operations within the different fragments. In detail, the modules shown in figure 4 add the following functionalities to the framework:

**Modules for the initial interaction:**
After starting-up the framework, these modules provide a first possibility to augment the mining process with additional information about the data, the mining goals and any given constraint on the mining process. For example by distinguishing between *undirected and directed graphs*, the user

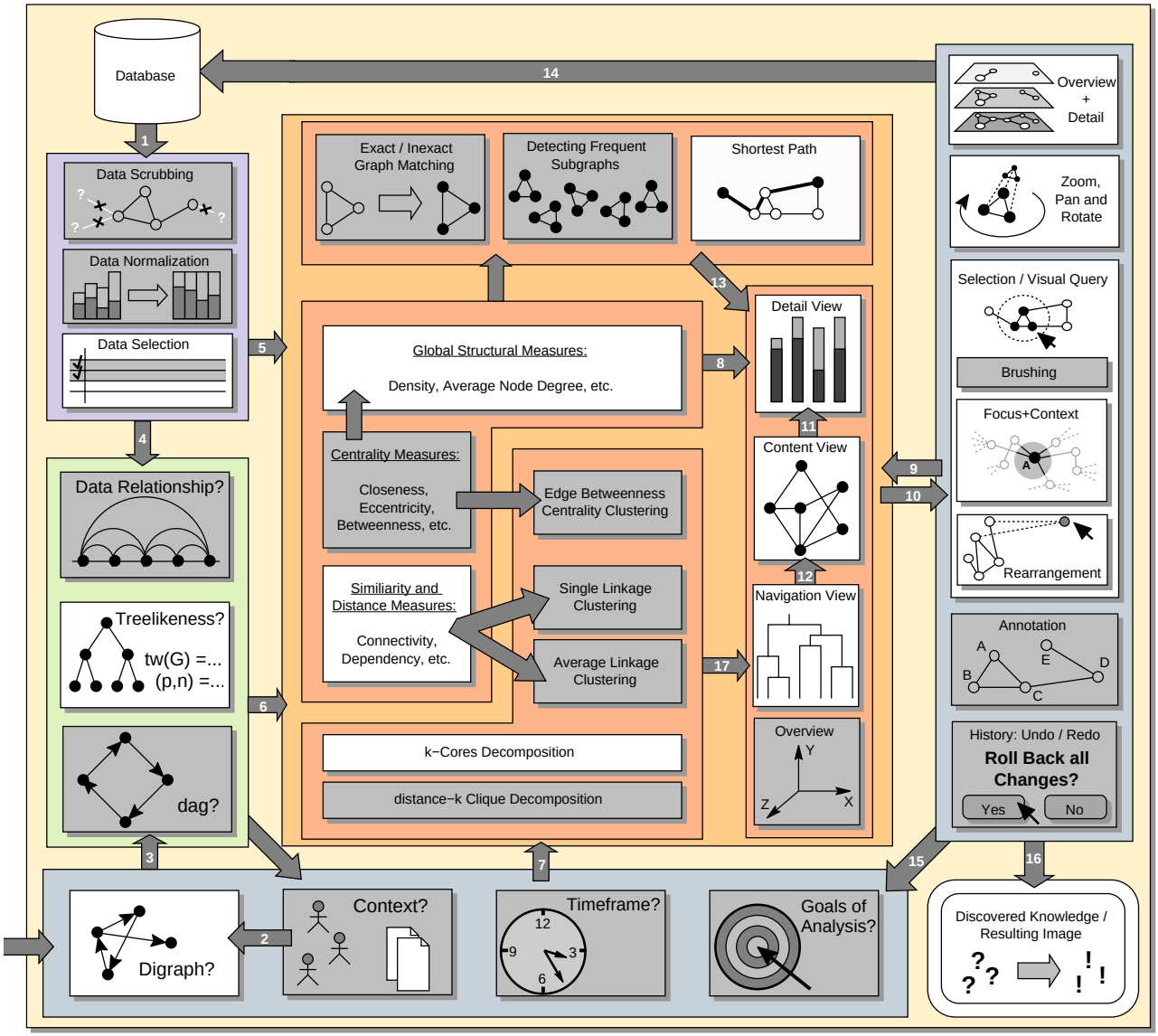


Figure 4: Our VDM-framework design with representative modules. Greyed out Modules have not yet been implemented or their implementation has not yet been adapted to be used within the framework.

indirectly influences the choice of applicable algorithms. Sometimes, this can also be deduced directly from the *context of the data set* – i.e. link structures in hypertext documents always form a digraph. Additional information that affects the selection of appropriate algorithms can contain *upper runtime limits* to insure acceptable reply times or explicitly stated *goals of analysis* (Nocke & Schumann 2004). These modules directly fulfill the design criteria for adaptability to different application backgrounds and scalability to users with varying background knowledge.

**Modules for the preprocessing:**
The modules presented in this fragment are responsible for data *cleansing* from measuring errors such as dangling ends in the set of edges. Furthermore, in the preprocessing phase a *data selection* and edge weight *normalization* can be performed etc. Transformations like these allow to define a standardized, abstract data format to work with, which is an essential design criteria.

**Modules for the processing of structural measures:**
In case of a digraph structure, this fragment could test a graph for *acyclicity*, which might again trigger additional goals of analysis and a selection of especially optimized algorithms in the algorithmic kernel. Furthermore all of the discussed measures in section 3.1 can either be calculated here, or at least be approximated if the exact calculation would be too time consuming.

**Modules for the algorithmic kernel:**
The functionalities of most of the modules shown in this fragment have already been addressed in section 3. Appropriate graph-algorithmic and visualization modules are selected via the calculated measures as described in 3.4. Since there usually does not exist a single optimal graphical representation of every aspect of the data, we propose a fourfold approach in the style of the ideas on multiple views in (Scharl 2002). The data set's inherent structures that have been computed beforehand are presented in a *navigational view*, that makes them accessible in a hierarchical manner. Once a region of interest has been selected in the navigational view, the sub structure associated with it will be shown in a *content view*. Selecting a node, an edge or a substructure within the content view will again trigger a detailed graphical representation of those in the *detail view*. To prevent loosing the orientation within the data set while visually exploring it, a rather static *overview*

should be provided to aid at keeping the global mental map of the structure. It is further possible to use multiple instances of each view that utilize different visualization techniques. This view-concept ensures a simple and uniform way to explore the data while using the available screen space most effectively.

**Modules for the interaction and postprocessing:**
This fragment contains the usual interactions and manipulations upon the visualized data set, which enable the user to carry out common exploration tasks as stated by the design criteria. This includes interactive annotation to add supplementary comments to the presented clusters and substructures and a similar history concept to the one we have developed and presented in (Kreuseler, Nocke & Schumann 2004).

### 4.3 Implementation

The functional modules introduced in figure 4 have been implemented in an interactive framework that is based on C++ and the Qt-Library. White colored modules are fully-integrated, and grey modules are under development.

A major challenge developing this framework was to implement an efficient storage concept for structures. In this context, a lot of different approaches to store graphs have been discussed in literature. Besides delegating storage issues to 3rd party products like relational databases, in practice only three techniques are widely used: **adjacency matrices** for fairly small graphs, **object-based data structures** for medium-sized graphs, that use objects for each node and link them via pointers, and finally **table-based structures** that contain the nodes and edges along with their attributes in a simple linear order, which is suited for large graphs. For its small memory overhead, most frameworks for processing large graphs utilize the last approach. A popular example is the Java-based implementation of such a linear storage for large graphs by the InfoVis ToolKit (Fequete 2004) which extends the table-based approach with several additional features, like fields for **auxiliary metadata** on each column and the ability of **masking** several nodes or edges within the list by toggling certain selection-bits. Beside these, our table-based data structure adds the following functionality:

- Columns may contain **permutations** of the table's rows, i.e. a topological ordering or the sequence of a breadth-first-search. That allows to store multiple orderings and eradicates the need to shuffle around the table's rows to sort them.

- Besides ordinary values, a cell of the table may contain an entire list to efficiently store **adjacency lists** or even **hypergraph structures** with a variable number of nodes per hyperedge within the same data structure.

- Each column can exist as a **placeholder** only, which will be filled in automatically when it is used for the first time. This pushes file-reading operations and computations as far back as possible and may save time and memory footprint in an average use case.

To improve the speed of look-ups in huge lists, a simple caching layer is used which allows direct access to the last couple of entries that where used. This storage concept addresses mainly large and complex structures that take up a lot of space in memory – up to the point, where they just will not fit in there anymore (*memory bottleneck*). It counters this case through a layered set of predefined behaviors, from which can be chosen automatically or interactively:

1. To push storage problems as far as possible, a strong emphasize of the framework lies upon an efficient storage of the data, that is painstaking space-saving and still tries to maintain an acceptable average access time. This is done by using the before mentioned table-based approach that can be augmented with supplementary data if enough memory is available or if an algorithm definitely needs it. Furthermore, the data set can be split up in smaller subsets if the overall structure has unconnected components that can be computed separately.

2. In case the data still does not fit into the memory, unneeded attributes like edge weights or previously computed measures that are not vital to run a specific algorithm, will not be loaded into the memory unless the user explicitly says so (*placeholder columns*).

3. If the memory's sufficiency is of further concern, all standard modules of the framework must be exchangeable with external algorithms that are especially optimized for this special case.

4. For those modules that do not provide a special external version, a smart caching layer has to be introduced to the frameworks I/O to minimize memory swaps.

In most cases, the first layer that employs a deliberate usage of memory will be effective enough to fit all data in. For larger data sets, the framework has to utilize one of the latter layers until every needed attribute can be accessed. Moreover, it is imaginable to introduce additional intermediate layers like certain garbage collection functionalities or semi-external versions of frequently used modules that make use of the situation where at least the node set fits in the main memory. This would further increase the chance to prevent the use of generally slower external algorithms. Thus far, the first two of the presented layers have been successfully implemented.

### 4.4 Graphical user interface

Sometimes, when trying to visualize a set of data, it turns out that the amount of data is just too large to fit the output device, i.e. the data objects that should be displayed outnumber the available pixels (*screen bottleneck*). To reduce the visual complexity in order to circumvent this bottleneck, additional time consuming clustering steps might thus be needed. For the sole purpose of decreasing the structural complexity through node-aggregation, we propose the use of heuristics or graph decompositions that have a linear runtime bound. Ideally this reduction produces a hierarchy that can be used to filter the results interactively up to a desired level of detail, like the mentioned $k$-core-clustering. Figure 5 presents yet another method to explore a potentially overloaded and overdetailed graphical representation. Since there is no visualization method, that serves all demands equally well, different views upon the data are generated as needed and functionally linked as described in section 4.2: The **content view** is shown after the initial layout is calculated by an appropriate visualization module (in dependency of the treelikeness-value we use Fruchterman-Reingold's spring embedder method (Fruchterman & Reingold 1991) for graphs and the MagicEyeView for

trees and treelike graphs) and provides several possibilities of interaction: zoom, rotation, selection, filtering, etc. The output can be interactively constrained to only those nodes that have a certain centrality measure within the range defined by the sliders at the right. An additional **navigation view** is based upon the dendrogram of a hierarchical clustering. We used a subsequent display of MagicEyeViews (Kreuseler & Schumann 2002) to visualize the huge hierarchical structure of the dendrogram, where each selected cluster can itself be a root-node within a newly generated subview. As an example for a **detailed view**, figure 5 shows the distribution of the $k$-neighborhood from section 3.1 for $k = 1 \ldots 5$. By selecting a certain neighborhood through this display, the content view can be adapted to show it for exploratory analysis.

### 4.5 Summary

The proposed framework architecture has a modular, extensible design. It has a general underlying data structure, and thus, can handle various structures from different backgrounds. Explicitly integrating the application context and user goals makes it scalable to various users from different domains. It offers both automatic and interactive mechanisms to control measures and techniques in the algorithmic kernel, especially enabling users to specify queries visually (for instance to select substructures). It enables users to specify, derive and apply additional data (metadata) which can be used for the semi-automatic selection of suitable algorithms, lead users through the exploration process and increase the user knowledge about the handled structures. Furthermore, the architecture supports to handle large data sets by smart data structures and by a control mechanism for long-lasting resp. interactive processable calculations. Interactive visualization techniques have been integrated, even applicable for large data sets. Therefore, they apply focus+context, overview+detail and brushing+linking paradigms and support interactive sorting, filtering as well as navigational support.

## 5 Case study

We successfully applied our framework to a variety of data sets. This included a web link graph of our institute internet pages (51497 nodes, 425247 edges), a citation network (509 nodes, 1551 edges) and peer-to-peer-networks with a few hundred nodes. For this paper, we demonstrate the usefulness of our framework design and present interesting insights for the medium sized Edinburgh Associative Thesaurus data set (short EAT, see http://www.eat.rl.ac.uk). For the following exemplary analysis, we present the exploration process closely related to the flow of the design chart from figure 4, giving details about the depicted modules and the arrows interrelating them.

The EAT data set consists of an empirical set of word associations (Kiss, Armstrong, Milroy & Piper 1973). Therefore, a list of 8.210 very frequently used English words (stimuli) has been compiled and the associative responses from test persons were gathered. Since the responses themselves are not necessarily stimuli, we eliminated these dangling ends in a data scrubbing preprocess (fig. 4, arrow 1). The resulting graph contains of 8.210 nodes (the stimuli) connected via 261.453 weighted edges.

Then, in an *initial interaction* step, context knowledge about type and history of the data set leads to the specification of the graph as a digraph (arrow 2). Based on this knowledge, a variety of descriptors can be calculated for the preprocessed digraph (arrows 3 and 4). This includes to calculate the *treelikeliness* which is relevant for the later selection of an appropriate visualization (as described in section 3.1). The actual $(p, k)$-treelikeness of the given data set results to (3.1%, 253244), which means that 253244 edges would have to be deleted in order to convert the network into a tree — only 3.1% of all edges would remain to form the spanning tree.

Based on these calculations and specifications, we started the main exploration phase in the *algorithmic kernel* (arrows 5, 6 and 7). First, important global structural measures have been calculated. For instance, to estimate the graph connectivity, we calculated an *average node degree* of 31.85, which means that each stimulus-word is associated with approximately 32 other stimuli-words. Based on this medium *average node degree* and due to a low *treelikeness* value we concluded that the EAT graph is a medium connected network and not suited to be laid out with a tree visualization technique. Hence, we chose a network visualization as content view (arrow 8). Therefore, to get a first overview of 8.210 nodes (arrow 7 and 9), we actually used a 3D-spring-embedder network visualization technique (Fruchterman-Reingold (Fruchterman & Reingold 1991)). This computation lasted about an hour (on an Intel PentiumM 1.4GHz machine with 512 MByte RAM), and thus, was executed non-interactively within the algorithmic kernel.

Then, to get more details about certain nodes, the user can *zoom*, *pan* and *rotate* the graph layout, as well as *select* certain nodes (arrows 9 and 10). Furthermore, to filter the crowded representation (arrow 9), we calculated the associative neighborhoods of chosen words based on the structural measure *1-neighborhood*. Then, using the two sliders depicted on the right side of figure 5, the user can fade out all nodes that do not have a *1-neighborhood-size* within a certain range (arrows 8, 9 and 10). For instance, in figure 5 we applied a 1-neighborhood-filter of 165 as minimum and a value of 1106 neighbors as maximum. Thus, the user can investigate a sparsely crowded graph with the main associated stimuli, leaving out all stimuli that are lesser associated. The maximum value of 1106 belongs to the node of the word MAN, which is the most associated word in this data set. The next heavily associated words are GOOD and SEX (ca. 870 associative links to other words).

Further, the user can get details-on-demand about these selected nodes (fig. 4, arrows 9, 10 and 11), displaying the $k$-neighborhood-diagram of a selected node in a *Detail View* (arrows 8 and 9). This gave us another interesting insight: most nodes lie within a distance of 3 (see in the $k$-neighborhood-diagram in fig. 5). This is a further proof for the observed medium to high density of the graph and indicates that there are no isolated substructures.

Then, to get a grip at this highly interrelated graph, we clustered the graph hierarchically using a $k$-core-decomposition. This computation lasted a couple of minutes, and was also executed within the algorithmic kernel. Using the resulting dendrogram, the user can explore the whole graph in an *overview+detail* manner, *focusing* on certain hierarchy levels (fig. 4, arrows 8, 9 and 17). Therefore, in a *navigation view*, certain levels of the hierarchy are displayed in a *MagicEyeView* (see fig. 5), and the user can interactively focus on certain clusters and hierarchy leaves of interest, still keeping the context visible (fig. 4, arrows 9 and 10). Furthermore, clusters of interest can be selected in the *MagicEyeView*, to explore the subgraphs they induce within the *content view* (arrows 9, 10 and 12). A *brushing* mechanism displays these subgraphs. Examples for words that have automatically been clustered together are:

- ITS, MUST HAVE, POSSESSIVE

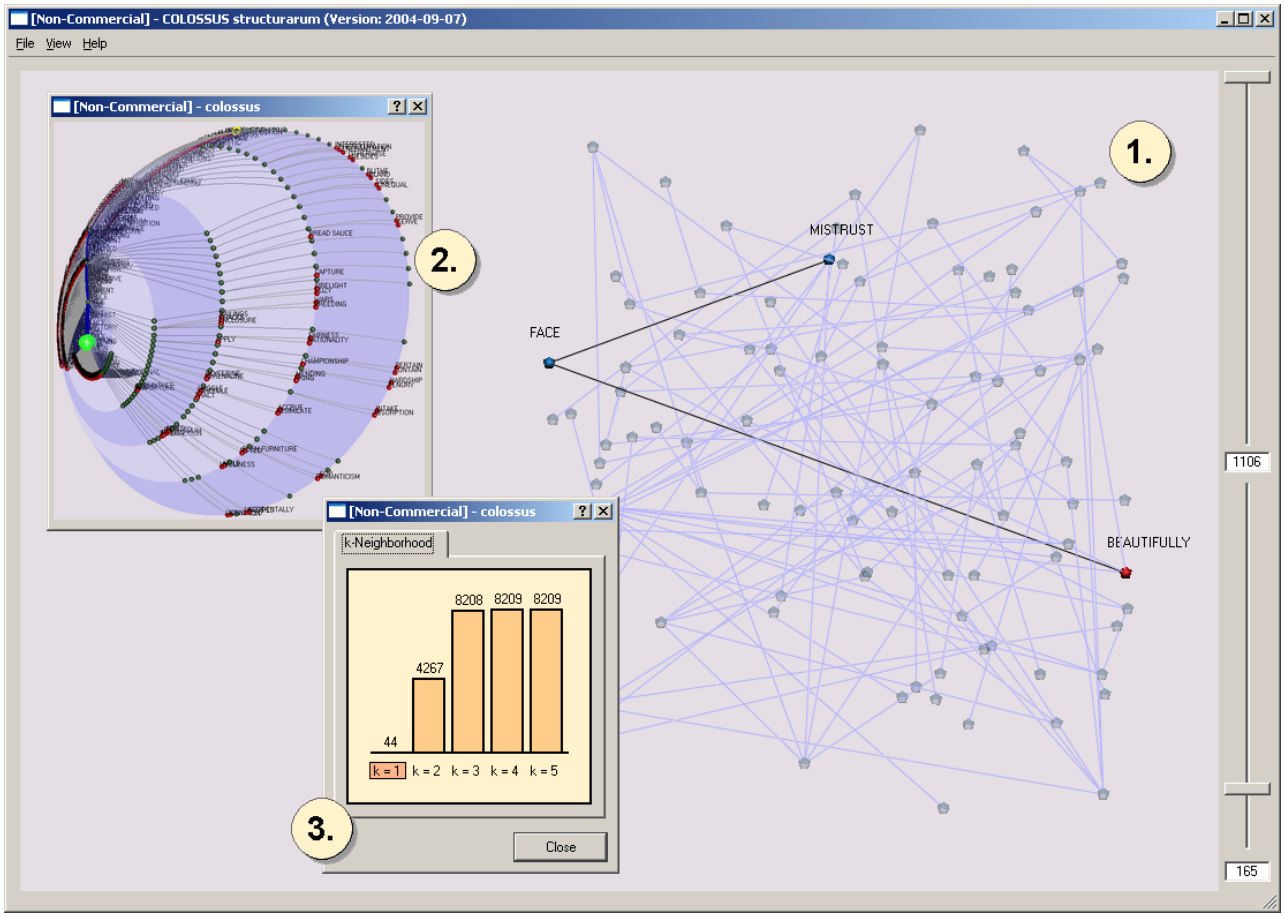


Figure 5: An overview of the framework's GUI: (1.) the content view showing a small part from a larger data set, including a highlighted shortest path and a red colored, selected node; (2.) the navigational tree-view [MagicEyeView (Kreuseler & Schumann 2002)] containing the browsable result of a hierarchical cluster algorithm; (3.) the detailed view of the $k$-Neighborhood-distribution of the selected node from (1.) with its 1-Neighborhood being selected.

- CRUSHING, DESTRUCTIVE, DESTROYING
- ANTICIPATE, INSTRUCTIONS, AWAIT

Moreover, to establish interesting connections between two selected words, the user can interactively select the words, and compute the *shortest paths* between them (arrows 9, 10 and 13). Then, this path can be depicted and *focused* in the *content view* (see fig. 5), showing a path between the words MISTRUST and BEAUTIFULLY), keeping the rest of the graph in the *context* applying alpha blending.

Summarizing, the user has a variety of possibilities to interact with the provided modules. Therefore, our framework delivers a variety of exploration paths, to support various exploration contexts and tasks. The user can refine the focus on the data set and explore substructures (fig. 4, arrow 14), refine exploration context (arrow 15), and then restart the whole process. Thus, as an iterative process, the framework supports alternative visual navigation and mining paths to the desired result (arrow 16).

## 6 Conclusion and future work

In this paper, we investigated the tight integration of methods from graph theory with visualization methods. Therefore, we introduced graph theoretical methods and their applicability for a VDM of structures systematically. In particular, we described how to apply these methods to design good visual representations. Furthermore, we introduced a general, modular and flexible design for a VDM framework for structures, outlined its implementation details and discussed its applicability based on a real-world example.

We tested our framework with different data sets, for instance a WWW-link-structure with 50.000+ web sites and approx. 425.000 links in between them. The gained results where highly satisfying. Nevertheless, there are still challenges for future work. We have to continue testing and evaluating the framework's usability and its scalability to even larger structures. Additionally, more measures, algorithms and visualization techniques need to be integrated.

## Acknowledgements

The authors like to express their thanks to Prof. Andreas Brandstädt for helpful discussions as well as Andreas Pohl and Clemens Nafe for testing, evaluating and using the framework for their research on peer-to-peer networks.

## References

Abello, J., Finocchi, I. & Korn, J. (2001), Graph Sketches, *in* 'IEEE Symposium on Information Visualization (InfoVis'01), San Diego', pp. 67–72.

Abello, J. & van Ham, F. (2004), Matrix Zoom: A Visual Interface to Semi-external Graphs, *in* 'IEEE Symposium on Information Visualization (InfoVis'04), Austin', pp. 183–190.

Ahlberg, C. (1996), 'Spotfire: an information exploration environment', *SIGMOD Record (ACM Special Interest Group on Management of Data)* **25**(4), 25–29.

Ankerst, M. (2001), Visual Data Mining with Pixel-oriented Visualization Techniques, *in* 'Proceedings of ACM SIGKDD Workshop on Visual Data Mining'01; San Francisco'.

Batagelj, V., Mrvar, A. & Zaveršnik, M. (1999), Partitioning Approach to Visualization of large Graphs, *in* 'Proceedings of the 7th International Graph Drawing Symposium', number LNCS 1731, pp. 90–97.

Bertin, J. (1981), *Graphics and Graphic Information-Processing*, Walter de Gruyter.

Brandes, U. (2001), 'A Faster Algorithm for Betweenness Centrality', *Journal of Mathematical Sociology* pp. 163–177.

Brandes, U. & Corman, S. (2002), Visual Unrolling of Network Evolution and the Analysis of Dynamic Discourse, *in* 'IEEE Symposium on Information Visualization (InfoVis'02), Boston', pp. 145–151.

Brandes, U. & Wagner, D. (2003), visone - Analysis and Visualization of Social Networks, *in* 'Graph Drawing Software', Springer, pp. 321–340.

Bunke, H. (2000), Graph matching: Theoretical foundations, algorithms, and applications, *in* 'Proc. Vision Interface 2000, Montreal', pp. 82–88.

Edachery, J., Sen, A. & Brandenburg, F. (1999), Graph Clustering using Distance-k Cliques, *in* 'Proceedings of the 7th International Graph Drawing Symposium', number LNCS 1731, pp. 98–106.

Fekete, J.-D., Wang, D., Dang, N., Aris, A. & Plaisant, C. (2003), Interactive Poster: Overlaying Graph Links on Treemaps, *in* 'IEEE Symposium on Information Visualization (InfoVis'03), Seattle'.

Fequete, J.-D. (2004), The InfoVis Toolkit, *in* 'IEEE Symposium on Information Visualization (InfoVis'04), Austin', pp. 167–174.

Frischman, Y. & Tal, A. (2004), Dynamic Drawing of Clustered Graphs, *in* 'IEEE Symposium on Information Visualization (InfoVis'04), Austin', pp. 191–198.

Fruchterman, T. & Reingold, E. (1991), 'Graph drawing by force-directed placement', *Software – Practice and Experience* **21**(11), 1129–1164.

Gansner, E., Koren, Y. & North, S. (2004), Topological Fisheye Views for Visualizing Large Graphs, *in* 'IEEE Symposium on Information Visualization (InfoVis'04), Austin', pp. 175–182.

Girvan, M. & Newman, M. (2002), 'Community structure in social and biological networks', *PNAS* **99**(12), 7821–7826.

Granitzer, M., Kienreich, W., Sabol, V., Andrews, K. & Klieber, W. (2004), Evaluating a System for Interactive Exploration of Large, Hierarchically Structured Document Repositories, *in* 'IEEE Symposium on Information Visualization (InfoVis'04), Austin', pp. 127–133.

Heer, J., Card, S. K. & Landay, J. A. (2005), Prefuse: a Toolkit for Interactive Information Visualization, *in* 'CHI 2005, Human Factors in Computing Systems'.

Herman, I., Marshall, M. & Melançon, G. (2000), Automatic generation of interactive overview diagrams for the navigation of large graphs, Technical Report INS-0014, Reports of the Centre for Mathematics and Computer Sciences.

Kiss, G., Armstrong, C., Milroy, R. & Piper, J. (1973), An associative thesaurus of English and its computer analysis, *in* 'The Computer and Literary Studies', Edinburgh University Press.

Kreuseler, M., Nocke, T. & Schumann, H. (2004), A History Mechanism for Visual Data Mining, *in* 'IEEE Symposium on Information Visualization (InfoVis'04), Austin', pp. 49–56.

Kreuseler, M. & Schumann, H. (2002), 'A Flexible Approach for Visual Data Mining', *IEEE Transactions on Visualization and Computer Graphics* **8**(1).

Lamping, J., Rao, R. & Pirolli, P. (1995), A focus+context technique based on hyperbolic geometry for viewing large hierarchies, *in* 'ACM Proceedings of Computer-Human Interaction (CHI95); Denver, Colorado, USA', pp. 401–408.

Nocke, T. & Schumann, H. (2004), Goals of Analysis for Visualization and Visual Data Mining Tasks, *in* 'CODATA Workshop Information, Presentation and Design (March 2004), Prague'.

Robertson, G., Mackinlay, J. & Card, S. (1991), Cone trees: Animated 3d visualization of hierarchical information, *in* 'ACM Proceedings of Computer-Human Interaction (CHI'91)', pp. 189–194.

Roth, S. A., Lucas, P., Senn, J. A., Gomberg, C. C., Burks, M. B., Stroffolino, P. J., Kolojejchick, J. A. & Dunmire, C. (1996), Visage: A User Interface Environment for Exploring Information, *in* 'IEEE Symposium on Information Visualization (InfoVis'96), San Francisco', pp. 3–12.

Scharl, A. (2002), Adaptive Web Representation, *in* 'Human Computer Interaction Development & Management', pp. 255–260.

Shi, J. & Malik, J. (1997), Normalized Cuts and Image Segmentation, *in* 'Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'97)', pp. 731–737.

Shneiderman, B. (1992), 'Tree Visualization with Treemaps: A 2D Space Filling Approach', *ACM Transactions on Graphics* **11**(1), 92–99.

Stolte, C., Tang, D. & Hanrahan, P. (2002), 'Polaris: A system for query, analysis, and visualization of multidimensional relational databases.', *IEEE Trans. Vis. Comput. Graph.* **8**(1), 52–65.

Tollis, I., Eades, P. & di Battista, G. (1999), *Graph Drawing - Algorithms for the Visualization of Graphs*, Prentice Hall.

Valiente, G. (2002), *Algorithms on Trees and Graphs*, Springer.

van Ham, F. & van Wijk, J. (2004), Interactive Visualization of Small World Graphs, *in* 'IEEE Symposium on Information Visualization (InfoVis'04), Austin', pp. 199–206.

Voigt, D. (2001), WWW-based Representation of complex Information Structures (in German: WWW-basierte Darstellung komplexer Informationsstrukturen), Master's thesis, University of Rostock, Department of Computer Science.

Zhang, P. (1994), *Method of Mapping DNA Fragments*, United States Patent No. 5667970.