Visual Data Mining of Graph-Based Data

Oliver Niggemann

From the Department of Mathematics and Computer Science
of the University of Paderborn, Germany

The Accepted Dissertation of Oliver Niggemann

In Order to Obtain the Academic Degree of Dr. rer. nat.

Contents

Acknowledgments

This work has profited very much by suggestions from colleagues and by cooperations with other researchers. I especially wish to thank Professor Dr. H. Kleine Büning, my thesis advisor, for supporting me and my work. Furthermore, I thank Professor Dr. F. Meyer auf der Heide for evaluating this thesis.

I owe Dr. Benno Stein many thanks for setting the standard for this work and for reminding me that there's no such thing as a free lunch. The discussions with Dr. Theo Lettmann, Sven Meyer zu Eissen, Andrè Schulz, and all my other colleagues and their constant support were very important to me during the last years.

Michael Lappe from the Structural Genomics Group of the European Bioinformatics Institute in Cambridge (UK) provided me with many interesting problems from the field of biology and also helped me to solve some of them. I also thank Jens Tölle from the University of Bonn (Germany) for answering all those questions about computer networks.

Finally, I would like to express my deepest gratitude to my wife Petra. She made this work possible.

Part I

Concepts of Visual Data Mining

Chapter 1

Fundamental Methodology

The field of data visualization has reached a point of inflection. The rapidly growing amount of data does not anymore allow for a presentation of all given data items. Furthermore, the complexity of many datasets surpasses the user's ability to identify the gist or the underlying concepts. One solution to these problems is to analyze the given data and confront the user with an abstract view of the information.

The following examples may help to illustrate the breadth and complexity of current datasets:

"For instance, Wal-Mart, the chain of over 2000 retail stores, every day uploads 20 million point-of-sale transactions to an AT&T massively parallel system with 483 processors running a centralized database. At corporate headquarters, they want to know trends down to the last Q-Tip." (in [63])

Visualizing traffic in computer networks is one of six applications described in this work. Currently several million IPs (computer addresses) are used in the internet (from [118]). Even if not all of these addresses have to be visualized at the same time, most realistic scenarios still comprise several thousand computers which interact in highly complex ways.

SGI describes customers for its data visualization software package from the telecommunication market as follows: *"According to the Technology Research Institute, 26 percent of the carriers it interviewed had warehouses of 500GB or greater, with that percentage projected to increase to 63 percent in 1997. Terabyte, and multi-terabyte-sized telecommunications databases are not unusual."* (in [171])

Another typical domain is the visualization of protein interactions within the framework of the human genome project (see section 6.3): E.g. the SCOP hierarchy for the classification of proteins comprises 11410 entries (from [150]). Again, the interactions between proteins can be very complex.

In this chapter, a methodology is presented which combines data abstraction techniques and visualization methods.

1.1 Structure of this Work

This work is structured as follows: Two main parts exist. Part I introduces the visualization concepts which are applied to six domains in part II.

This chapter describes the ideas discussed in this work. Section 1.3 introduces the general methodology for visualizing complex data according to the AI-visualization paradigm. Section 1.4 formally defines the to-be-visualized graphs. In section 1.5, the visualization approach is extended for tabular data sets. A short overview of existing methods for visual data mining is given in section 1.6.

Each of the next four chapters explains a single step of the new visualization methodology. These chapters end with a section called "Evaluation" which verifies whether or not that step complies with the four features given above. After all steps are explained comprehensively, section 4.3 elaborates on the combination of those steps, i.e. the visualization methodology is discussed in detail.

Chapter 6 describes applications where data is given directly as a graph. Tabular data sets are the subject of chapter 7. Each application closes with a discussion of the results. A summary is given in chapter 8.

While the first part of this work gives only theoretical evaluations of the described algorithms, the second part assesses the usefulness of the methods by applying them to real-world problems.

1.2 The AI Visualization Paradigm

The field of Artificial Intelligence (AI) tries to solve complex problems by analyzing human problem solving approaches. This work applies the general AI-philosophy to the visualization of complex data: Humans understand information by forming a mental model which captures only the gist of the information. Manual visualizations often resemble this mental model. Thus, applying Artificial Intelligence to the automatic visualization of data results in the following AI-visualization paradigm:
"Every visualization should aim at the human mental model."

Finding a presentation close to the mental model supports the human understanding of complex data. Thus, information hidden in the given data is made explicit. In this sense, such a visualization may be viewed as visual data mining or visual knowledge discovery. Data mining[1] or knowledge discovery is a field of computer science which deals with the problem of finding unknown structures and relations in complex data sets. The main idea of visual data mining is to combine the advantages of computers and humans, i.e. the computer's ability to quickly process large amounts of data and the human talent of identifying visual patterns. In this work, visualization is used solely as a means of knowledge discovery, esthetic considerations are less important.

While it is difficult to find general features of visualization methods following the AI-visualization paradigm, it is possible to analyse problem instances and to identify common features of good solutions. The author does not claim that all visualization methods following the AI-visualization paradigm must comprise these features. But he claims that in many domains the application of the AI-visualization paradigm results in visualization methods comprising the four features presented below.

[1] An overview concerning data mining can be found at [43, 59, 14, 18, 113].

In this work, six problem domains are analyzed and solutions to the visualization problem are developed. Four common features are identified. The new methodology of structure-based visualization is introduced in section 1.3; this methodology leads to visualizations which comprise all four features.

These features can also be used as criterions for the applicability of the methodology to a new problem: The user assesses whether the features are desirable for a solution.

The result of the visualization as shown to the user is called presentation. The first two features describe this presentation:

1. **Object-based Presentation:** Whenever the given data comprises distinct objects, the user wants to recognize these objects in the representation, i.e. objects and the spatial relations between them should carry the visual information. Objects are often represented by simple graphical objects (e.g. circle, texts, etc.).

   Thus, the given data should either consist of objects and their relations or it must be transformed into this structure. Such data can be modeled by a graph[2], where objects form the nodes, while edges represent the given relations between the objects.

2. **Constant Dimension Presentation:** The presentation always uses a constant number of dimensions, i.e. more complex data does not result in a more complex representation. In this work, all data sets are visualized on a 2-dimensional plane because for cognitive reasons many users prefer 2-dimensional presentations. Most results can easily be extended for 3-dimensional presentations. Since humans often have problems with presentations having more than three dimensions, the constant dimension restriction makes sense.

   This work deals with data for which no inherent visualization exists. Data which includes inherent visualizations often describes geometrical or spatial information: A mathematical function $y = f(x)$ has an inherent visualization (its 2-dimensional plot). A set of 3-dimensional polygons defining the outline of a human face also has an inherent visualization (any rendering of the polygons). The reader may note that since the data directly defines the corresponding visualization, the number of dimensions used for the representation is determined by the data. Therefore, a variable dimension presentation is used.

   A set of people and their relationships to one another does not have an inherent visualization. Nevertheless most humans have a clear idea of an appropriate visualization: Visualize people as objects on a plane and place two people with a close relationship closely together. The number of dimensions used for the visualization is defined by the capabilities of the user, not by the data.

The last two criterions define features of the visualization process itself:

3. **Abstraction:** In order to approximate the mental model, the data is abstracted: Machine learning is used to identify the gist of the information, i.e. abstract concepts hidden in the data are identified. In this process

[2]This graph is formally defined in section 1.4, an introduction to graphs can be found in [80].

details are lost, thus the methodology presented here is not suited for problem classes where no information can be lost. Artificial data sets which often do not comprise any underlying concept are also not suited.

Abstraction also allows for the visualization of larger data sets: Working in a top-down approach, visualization algorithms can first layout only the identified abstract concepts. In a second step, more detailed information within the abstract concepts are visualized. Such a method therefore creates from the original dataset several smaller datasets which are laid out separately.

4. **User Adaption:** Since the precise mental model of the user is unknown, the user should be integrated into the visualization process. When appropriate, the needs and possibilities for user interactions are examined in the following chapters.

In this work, the to-be-visualized data is restricted to graphs. Graphs are a very common means of modeling and always present an object-oriented view of the data. Tabular data is only discussed in one special context: It is viewed as a special type of graph (see section 1.5) and graph visualization methods are applied. The reader may note that different visualization techniques exist for tabular data, details can be found in section 1.6.

1.3 Structure-Based Visualization

Human understanding is based on structure identification. Faced with a highly complex world, the mind reduces the problem of comprehending a situation by means of abstraction: structures are identified and classified. E.g. visual recognition relies on identifying already known objects, the human design process has often been explained by a pattern recognition approach, and people diagnosing a defective technical device exploit the structure of the device by identifying faultless or defective substructures. Automatic structure identification emulates this process. Since most data sets model a limited part of reality, reality's structure is also imprinted on the data.

In many cases, classified structures should be visualized in a deterministic way, i.e. similar structures should result in similar visualizations. This allows for an easy understanding of already known structures and repetitive patterns.

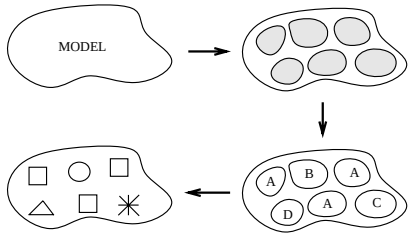

Figure 1.1: *The general Methodology of Structure-based Visualization*

These thoughts lead to the new general methodology of structure-based visualization (see also figure 1.1):

1. **Structure Identification:** The first step identifies the model's structure which is defined by closely related submodels and their relationships.

2. **Structure Classification:** The second step consists of classifying the individual substructures.

3. **Structure Envisioning:** The substructures and their relationships are visualized in the third step. Similar substructures are visualized in a similar way and repetitive substructures are emphasised. The visualization places nodes on a plane.

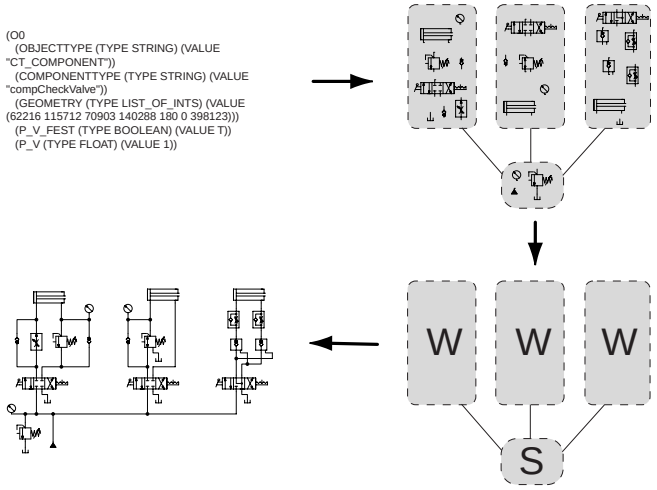

Figure 1.2: *An Example for the Visualization Methodology*

Figure 1.2 shows an example of this: First of all, the structure of the textually defined hydraulic circuit is identified, i.e. substructures are outlined. These substructures are classified: S denotes a supply substructure and W denotes a working substructure. Finally, the model is drawn. The layout emphazises the structure by placing elements of the same substructure closely together. The repetitive pattern of the three working units is emphazised by their parallel positioning.

1.4 Visualizing Graphs

As mentioned in the beginning of this section, the method of structure-based visualization requests the data to be given in the form of a graph. Thus a visualization graph is now defined formally:

Definition 1.4.1 (Visualization Graph)
The visualization graph $G = (V, E_c, E_v, \delta, w, \Delta)$ *is defined as follows:*

- V *denotes the set of nodes. Each node corresponds to an object.*
- $E_c \subseteq V \times V$ *is the set of edges used for finding a layout.*
- $E_v \subseteq V \times V$ *is the set of visible edges.*

- *$\delta \rightarrow \Delta^*$ defines the node labels over the alphabet Δ.*
- *$w : E_c \rightarrow R$ defines the edge weights. The larger the weight, the closer the relationship between nodes.*

Remark 1.4.1 *By introducing two separate set of edges E_c and E_v, it becomes possible to model situations where one set of edges (E_c) should be used to calculate positions for the nodes and a different set of edges (E_v) is shown to the user. Situations where $E_c \cap E_v = \emptyset$ are possible.*

One example is the visualization of hydraulic circuits: The user should only see edges representing tubes between hydraulic components. For the visualization task more edges exist: These edges connect components which should be placed closely together, e.g. a cylinder and its controlling valve.

We normally presume that edges are undirected, i.e. E_c and E_v comprise sets of nodes. Directed edges are modeled as lists of nodes. If methods are only suited for directed or undirected graphs, it is explicitly mentioned in this text. The reader may note that directed graphs can always be treated as undirected graphs. The final visualization can still show directed edges. Of course, in this way all information coded into the edge directions is lost. If undirected graphs are transformed into directed graphs, either additional knowledge about edge directions is used or the directions are randomly chosen. In the later case, the visualization method uses nonexisting information. This may lead to unsatisfactory results.

For simplicity reasons, only the currently important elements of a visualization graph are listed in this work, e.g $G = (V, E_c)$ may denote a visualization graph in a context where E_v, δ, w, Δ are of no importance.

In order to solve the graph layout problem, positions for all nodes have to be found:

Definition 1.4.2 (Graph Layout)
A graph layout L is a tuple $\langle G, \rho \rangle$ where $G = (V, E_c, E_v, \delta, w, \Delta)$ is a visualization graph and $\rho : V \rightarrow \mathbf{N} \times \mathbf{N}$ defines a two-dimensional position for each node.

In this work, it will be outlined how the general methodology of structure-based visualization can be implemented using visualization graphs:

The first step can be operationalized by means of cluster algorithms (section 2).

In the second step (classifying the clusters), machine learning is used to abstract the necessary classification knowledge from given examples. Section 3 describes two different approaches: direct classification and case-based classification.

The graph is laid out in the third step using graph drawing algorithms (see section 4).

Similar clusters can be visualized in a similar way by applying the same layout algorithms (e.g. see section 3.1.1). For many applications, the only reason to classify clusters is to use similar layout methods for similar clusters. For this reason, step 2 and step 3 are often highly interwoven.

Name	education	income	occupation	citizenship	gender
Meyer	Ms. CS.	$70.000	Sales	German	f
Smith	Ba. CS.	$55.000	Devolpment	British	m
Wagner	High school	$45.000	Training	Dutch	f
...	...	...	...	...	...

Table 1.1: A typical tabular data set

1.5 Visualizing Tabular Data

A special type of visualization graph is called tabular data set. Each node of a tabular data set has a node label which comprises a number of node features. The characters $',', ' (', ')'$ are used to separate the features in the node label and are therefore part of the alphabet Δ.

Definition 1.5.1 (Tabular Data Set)
Let $G = (V, E_c, E_v, \delta, w, \Delta)$ be a visualization graph. G is called a tabular data set if and only if:

- $E_c = \emptyset$.
- $\delta(v_i) = (f_1^{(i)}, \ldots, f_p^{(i)}), v_i \in V, p \in \mathbf{N}, f_j^{(i)} \in (\Delta \setminus \{',', ' (', ')'\})^*$
- $\{',', ' (', ')'\} \subset \Delta$

The variables $f_j^{(i)}$ are called features. Tabular data sets can be viewed as a table (see table 1.1), each row represents a node or object and each column defines a feature. In order to transform a tabular data set into a visualization graph, the edges E_c must be defined; this is the subject of section 5.

The analysis of tabular data sets arises in many different domains:

- ☞ Case-based reasoning (CBR) tries to solve a new problem by finding a similar old problem and adapting the old solution to the new problem. Old problem and their solution are saved in form of a table, each row represents an old problem and its solution. The columns model features of problem instances or their solutions respectively. CBR is explained shortly in section 3.2.

- ☞ Relational database also use tables to store data. An introduction can be found in [40, 176].

- ☞ Spreadsheets are used in many domains to store and analyze financial data and calculations.

All these fields developed highly specialized methods to analyze and visualize tabular data; the most popular methods are outlined in the next section. In this work only one aspect is examined: How can the methodology of structure-based visualization be applied to tabular data sets? For this, weights between objects are computed by analyzing the given features. The visualization itself uses only these weights, the original features are disregarded. Details can be found in chapter 5.

1.6 Existing Methods for Visual Data Mining

Several methods for visual data mining exist. In this work, the four features from the beginning of this chapter are used to classify these techniques. We furthermore differentiate between data given in the form of a graph and tabular data. This approach may result in an AI-centered view onto the field of visual data mining, but the reader may find such a classification helpful in comparing existing methods with the methodology presented in this work.

Four typical classes of visual data mining methods exist:

1. **Graph-given Data, Object-Oriented, Constant Dimension Visualization:** An detailed overview of such visualization methods is given in section 4.1.

2. **Tabular Data, Object-Oriented, Constant Dimension Visualization:** The object features of a tabular data set can be viewed as positions in p-dimensional space[3].

   These visualization techniques reduce the p dimensions to only two or three dimensions. Typical methods are Principle Component Analysis and Factor Analysis (see section 4.2), FastMap (this method may be viewed as a heuristic for Principle Component Analysis, see [42]) and Multidimensional Scaling (an overview can be found in section 4.1.2). Clustering may also be viewed as a dimension reduction technique: The p-dimensional features are replaced by a single value, the respective cluster. An extensive overview of existing algorithms for clustering is given in section 2. Another possibility is to transform the tabular data into a graph and apply graph visualization method. This approach is used in this work and an introduction can be found in chapter 5.

3. **Tabular Data, Object-Oriented, Variable Dimension Visualization:** These methods also treat the object features as p-dimensional positions. Unlike the dimension reduction techniques described above, special visualization techniques are used to show more than two or three dimensions. Sometimes several diagrams, each depicting a subset of the original dimensions, are used instead of one single diagram. The general idea is to use more complex visualization for more complex data. Of course, many of these techniques can also be used to visualize dimension-reduced data.

   Thoroughly and extensive overviews can be found in [38, 92, 23, 20]. Since this type of visualization is not the subject of this work, only a few typical examples are mentioned[4]:

   - Geometric Techniques:
     - Scatterplots: For all feature subsets, 2- or 3-dimensional visualizations are created, see [24].

[3] We assume metric features; as explained in section 5.1, non-metric features can be transformed into metric features.

[4] The following enumeration of the variable dimension visualization methods closely follows the classification suggested in [92].

  - Prosection Views: Selected value ranges are projected orthogonal onto a plane. This results in several different diagrams, each showing one projection (see [48]).
  - Parallel Coordinates: Objects are depicted as lines in a 2-dimensional drawing. The original p dimensions are placed on the x-axis. The y-value of a line (i.e. object) at a special x-value (i.e. dimension) is defined by the corresponding value of the object feature. Details can be found in [74].

- Icon-based Techniques: These techniques depict objects as icons on a plane. The appearance of the icon is used to visualize additional dimensions. An overview of these techniques can be found in [92].
- Pixel-oriented Techniques: These methods depict each feature in a single window. Feature values correspond to exactly one pixel in a window. The main idea is that similar feature value sets result in similar visualizations. An examples is Recursive Pattern Visualization ([90]).
- Hierarchical Techniques: Such methods depict more than two-dimensions in one 2D-diagram by placing smaller diagrams within larger diagrams. Examples are:
  - Dimension Stacking: Each position on a 2-dimensional plane comprises another 2-dimensional diagram showing dimensions three and four. Details can be found in [103]. An extension to 3D (boxes within transparent boxes) has been introduced in [140].
  - Treemap ([152]): For this method, a classification tree for the data has to be given. Such a tree subdivides the data recursively according to value ranges. Different features are used on different levels of the tree. This classification tree is used to subdivide the visualization plane.

4. **Tabular Data, Non Object-Oriented, Constant Dimension Visualization:** These methods concentrate on the visualization of aggregated values. The objects themself are no longer shown, only aggregations of their values are visualized. Typical examples are histograms, pie charts, and function plots. The OLAP method for managing databases also comprises such methods (see [27, 153, 136]). This type of visualization causes totally different questions than the visualization method described in this work. Thus, the reader is refered to [173, 174] for an an overview.

Most of these methods have been combined with distortion techniques. Such techniques distort a visualization in such way that important aspects are shown in detail while unimportant parts are blended-out.

Abstraction is mainly used by techniques in the visualization classes 1,2, and 4. In this work, a class 1 visualization technique is described which uses clustering to abstract graphs. Section 4.1.4 gives an overview of existing combinations of clustering algorithms and class 1 visualization techniques. The dimension reduction algorithms used with visualization methods from class 3 are also abstraction techniques. All of these methods work automatically.

Since most aggregations used by methods from class 4 create more abstract information, they may be viewed as a form of abstraction as well. But unlike

the other methods, aggregations are normally defined manually. This work focuses on the automatic abstraction of data.

User interaction can mainly be found in combination with visualization methods from class 3. It often compensates the missing abstraction ability. The user normally either chooses parameters of the representation (e.g colors, icons etc.), influences the used projection (e.g. for scatterplots), changes the level of detail, or chooses a subset of the data. A more extensive overview of user-interaction techniques for algorithms from class 3 can be found in [91]. Different techniques for user-interaction are introduced in this work.

1.7 Evaluation

The next three chapters describe individual steps of the structure-based visualization methodology. The fourth chapter deals with the transformation of tabular data sets into visualization graphs. At the end of each chapter, a section called "Evaluation" checks whether the described methods follow the four features of visualizations from section 1.3.

Some general comments can now be given:

1. **Object-based Presentation:** All steps treat objects as atomic entities. The "Structure Envisioning" step especially depicts objects as geometric entities.

2. **Constant Dimension Presentation:** This feature only has to be followed by the final envisioning step (see section 4.4). It is therefore only examined there. The reader may note that several existing visualization methods do not use a constant dimension representation (see section 1.6).

3. **Abstraction:** "Structure Identification" and "Structure Classification" are the main abstraction steps. Details can be found in chapters 2.4 and 3.3. This feature is consequently not discussed in other chapters.

4. **User Adaption:** The question of user adaption, i.e. the integration of the user in the problem solving process, is subject to world-wide research (e.g. see [137, 73, 88]). Most approaches try to create a model of the user's behaviour[5] which is then used to adapt an algorithm. This work is neither an introduction to user adaption nor does it focus on the user adaption aspect. Nevertheless, user adaption is examined as far as the structure-based visualization method is concerned.

[5]This model is often implemented by means of Bayesian Networks, see [133].

Chapter 2

Structure Identification

As outlined in sections 1.3 and 1.4, the visualization graphs are clustered in order to identify their structure and to speed up the layout process.

Section 2.1 gives an overview of existing approaches. Two new methods applicable to general graphs are presented in section 2.2. An innovative approach integrating additional domain knowledge is given in section 2.3.

A clustering is now formally defined:

Definition 2.0.1 (Clustering)
A clustering of a visualization graph $G = (V, E_c)$ *is a set* $\{C_1, \ldots, C_n\}, C_i \subseteq V \; \forall 1 \leq i \leq n$ *with* $\bigcup_{1 \leq i \leq n} C_i = V$.

Definition 2.0.2 (Disjunctive Clustering)
A disjunctive clustering of a graph $G = (V, E_c)$ *is a clustering* $\{C_1, \ldots, C_n\}$ *with* $C_i \cap C_j = \emptyset \; \forall 1 \leq i, j \leq n, i \neq j$.

Definition 2.0.3 (Set of all Disjunctive Clusterings ($\mathcal{C}$))
$\mathcal{C}(G)$ *is defined as the set of all disjunctive clusterings of a graph* $G = (V, E_c)$*:* $\mathcal{C}(G) = \{\{C_1, \ldots, C_n\} \mid \{C_1, \ldots, C_n\}$ *is a disjunctive clustering of* $G\}$

Definition 2.0.4 (Induced Structure of a Graph (S))
Let $G = (V, E_c, w)$ *be a graph and* $C \in \mathcal{C}(G)$*. The graph* $S(G, C) = (V_s, E_s, w_s)$ *with* $V_s = C$*,* $E_s = \{\{C_i, C_j\} \mid \exists \{v_0, v_1\} \in E_c : \; v_0 \in C_i \wedge v_1 \in C_j\}$ *and* $w_s(\{C_i, C_j\}) = \sum_{e=\{v_0,v_1\} \in E_c:\; v_0 \in C_i \wedge v_1 \in C_j} w(e)$ *is called the structure induced by* C.

Figure 2.1: *A clustered graph (left side) and its structure*

An exemplary clustering and its induced structure can be seen in figure 2.1.

For some applications it is helpful to identify further subclusters within clusters. Such a clustering a called hierarchical clustering:

Definition 2.0.5 (Hierarchical Clusterings)
A hierarchical clustering of a graph $G = (V, E_c)$ *is defined as a triple* $< C, T, C_0 >$ *where*

- $C \in \mathcal{C}(G)$
- $T = (C_T, E_T), C \subsetneqq C_T$ *denotes the so-called clustering tree*[1] *and*
- $C_0 \in C_T$ *defines the root of* T

Figure 2.2: *An hierarchical cluster (left side) and the corresponding clustering tree (right hand side).*

An exemplary hierarchical clustering can be seen in figure 2.2: On the left side a hierarchical clustering of a graph $G = (V, E_c)$ can be seen (the top cluster node C_0, which comprises all clusters, is not shown). The figure on the right hand side shows the clustering tree $T = (C_T, E_c)$ with $C_T = \{C_1, C_2, C_3, C_4, C_5, C_6\}$, $E_c = \{\{C_1, C_2\}, \{C_3, C_2\}, \{C_4, C_5\}, \{C_6, C_5\}\}$, $C_1 = \{d, e\}$, $C_3 = \{i\}$, $C_4 = \{a, b, c\}$, and $C_6 = \{f, g, h\}$.
The reader may note that every disjunctive clustering can also be treated as a hierarchical clustering.

Parenthesis: For many graphs the clustering (and the layout step) can be improved by using only a subset of the edges E_c. E.g. a common pre-processing step is to delete all edges which have a weight below a given threshold.
Another popular pre-processing step is the k-nearest-neighbour method: Every node deletes all adjacent edges except the k edges with the highest weights.

[1] See definition B.0.8.

2.1 Existing Methods

Clustering is one of the oldest subjects investigated in the field of Machine Learning: Pearson introduced in 1894 ([134]) a statistical clustering method. Since then, a large number of methods and algorithms has been introduced, some generally applicable, others highly specialized. The following section tries to give an overview by introducing a new classification scheme for clustering algorithms.

This schemes classifies clustering methods according to two categories. Other schemes have been given in [111, 41, 160]:

1. **Additional Knowledge Necessary for the Algorithm**: Some clustering methods rely on graph-theoretical features only. These algorithms exploit the domain knowledge coded implicitly into the graph's structure. Clusters found by these approaches are sometimes called natural clusters. Other methods use additional knowledge, e.g. explicit node labels or an explicit clustering criterion.

   In most cases, natural clusters are desired, i.e. the given graph comprises all necessary information to find its structure. Thus, any additional knowledge needed should be seen as a weakness of the algorithm.

2. **Optimization Criterion**: All clustering techniques optimize some criterion. Some methods use a given explicit function, while others have the criterion coded into an algorithm.

   Most people agree on the right clustering of a given graph, but finding a generally approved criterion proves to be difficult (see also section 2.1.3 and 2.2). The variety of different criterions used by different algorithms does not mirror an uncertainty or arbitrariness inherent in the human understanding of correct clustering, but an inability to express this implicit human understanding in terms of a mathematical function. The reader may note that the author neither denies the existence of problematic graphs, where different experts see different clusters, nor does he question the necessity to sometimes develop specialized algorithms for special domains.

The clustering approaches presented in the following paragraphs are typical for a whole class of algorithms. Most methods used today are specialized versions of these algorithms. Section 2.1.5 presents some variations.

2.1.1 Agglomerative Approaches

Additional Knowledge:	extra parameters necessary
Criterion:	inter-cluster distances

Agglomerative methods try to minimize the inter-node distances within clusters. Initially, every node forms its own cluster. The two closest clusters are united until reasonable clusters are found. The definition of the closeness of two clusters and the definition of the term "reasonable clusters" are the main problems when applying this method. Besides the graph, additional explicit parameters are used to define when to stop the union of clusters. Therefore, the algorithm needs extra knowledge.

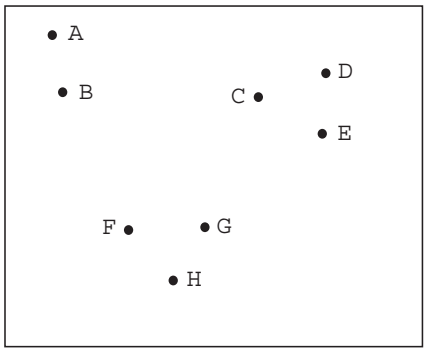

Figure 2.3: An Example Graph

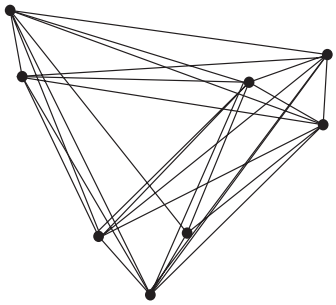

Figure 2.4: The corresponding edges

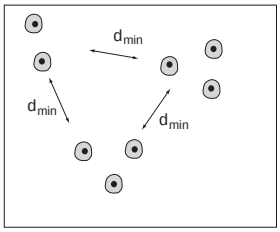

Figure 2.5: Nearest Neighbour: Step 1

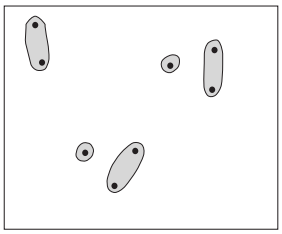

Figure 2.6: Nearest Neighbour: Step 2

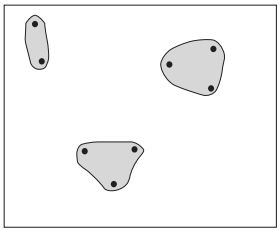

Figure 2.7: Nearest Neighbour: Step 3

One of the first implementations was the so-called single-linkage or nearest neighbour approach (see [45, 154, 78]). For this method, the distance between two clusters is defined by the distance between their closest nodes. Figure 2.3 gives an example: The graph $G = (V, E_c)$ already has a layout[2], it is fully connected, and all edges $e = (v_i, v_j)$ are weighted by the term $-dist(v_i, v_j)$, i.e. the closer the nodes are, the higher are the corresponding edge weight. For clarity sake, the edges are often not shown. Figure 2.4 shows the corresponding edges.

The iterative union of clusters normally stops when the distance between all clusters is larger then a given d_{min}. Figures 2.5, 2.6, and 2.7 show three steps of the algorithm. The algorithm stops at step 3, because all inter-cluster distance are large then d_{min} (d_{min} can be seen in figure 2.5).

Other agglomerative methods define the distance between two clusters differently:

- ☞ **Complete Linkage:** The distance is defined by the two furthest nodes.
- ☞ **Average Distance:** The distance is defined by the average distance between two nodes.
- ☞ **Centroid Distance:** For each cluster its centroid is calculated. The centroid of a cluster is the point with the smallest average distance to all other nodes within the cluster. The distance between clusters is then defined by the distance between their centroids.

[2] G comes with a layout in order to simplify the following explanations.

Figure 2.8: Chaining Effect: Step 1

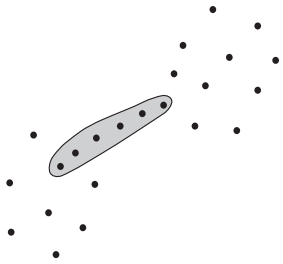

Figure 2.9: Chaining Effect: Step 2

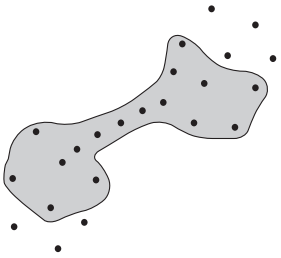

Figure 2.10: Chaining Effect: Step 3

A more extensive overview over agglomerative methods can be found in [41, 54].

Parenthesis: For some applications, agglomerative methods are used in a slightly different fashion: The algorithm does not stop, i.e. the method always ends with one cluster containing all nodes. A typical application is the creation of taxonomies in biology (e.g. in [154, 155]).

Agglomerative methods distinguish themself by some advantages:

⊕ They are fast: When the edge weights (i.e the node distances) are given, single-linkage algorithms need for a graph $G = (V, E_c)$ $O(|V|^2)$ or $O(|E|\log(|V|)$ steps. This can been seen by noting that single-linkage algorithms create iteratively a minimal spanning tree for G (see e.g. [188, 156]). Such an tree can be constructed in $O(|V|^2)$ or in $O(|E|\log(|V|)$ ([80]).

⊕ They can be easily implemented.

⊕ They are well understood and have been applied frequently.

Several disadvantages reduce the applicability of this method:

⊖ As outlined before, this approach relies solely on distance information. The so-called chaining effect may be used to illustrate that distance information is often insufficient: Figures 2.8, 2.9, and 2.10 show a situation where two distinct clusters are connected by a so-called bridge. The nodes forming the bridge are closer together than the nodes within the clusters. As can be seen in the figures, the clusters can not be found by using distance information only.

⊖ Being greedy algorithms, all agglomerative methods suffer from the disability to revise an earlier decision. Wrong early local decision, e.g. the union of two clusters, can not be changed. Therefore, agglomerative methods are a local heuristic approach.

⊖ The iterative union of clusters should stop when reasonable clusters are found. In order to define the term “reasonable clusters”, extra parameters are needed. These parameters are domain dependent and must be determined for each problem separately.

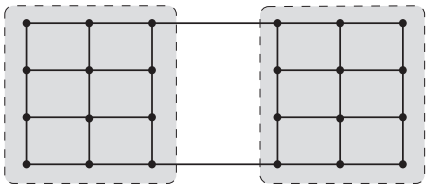

Figure 2.11: Mincut Clustering: Clustering 1

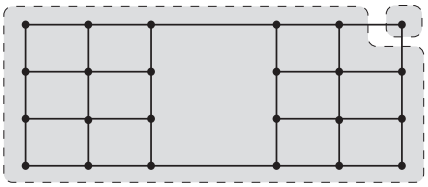

Figure 2.12: Mincut Clustering: Clustering 2

⊖ Graphs without weights can not be clustered.

2.1.2 Divisive Approaches

Additional Knowledge:	extra parameters necessary
Criterion:	minimum cut between clusters

Divisive approaches (also called MinCut clustering) try to minimize the cut of a clustering, i.e. the weight of the edges between clusters. Theses methods start with on cluster containing all nodes. The clusters are iteratively divided until reasonable clusters are found. Again, additional knowledge is necessary to define the term “reasonable clusters”.

The cut of a graph division plays an important role for these algorithms. The cut is defined as follows:

Definition 2.1.1 (Cut of a Clustering (cut))
The cut of a clustering $C = \{C_1, \ldots, C_n\}$ of a graph $G = (V, E_c, w)$ is defined as:
$cut(G, C) = \sum_{e=\{v_i,v_j\}\in E_c,\ \nexists C_k \in C:\ v_i \in C_k \wedge v_j \in C_k} w(e)$

Definition 2.1.2 (Minimum Cut of a Graph)
The minimum cut of a graph $G = (V, E_c, w)$ is defined as:
$\min_{C_1,C_2 \subseteq V, C_1 \cap C_2 = \emptyset, C_1 \cup C_2 = V} cut(G, \{C_1, C_2\})$

Divisive algorithms find a clustering $\{C_1, C_2\}$ with a minimum cut and subdivide C_1 and C_2 recursively. Since this method would always end with clusters containing one single node, it is necessary to define when to stop the partitioning. A common method uses a maximum cut value: A cluster is not subdivided if its maximum cut exceeds this given value.

More details can be found in [104, 185]. MinCut clustering has some interesting advantages:

⊕ Unlike Nearest-Neighbour approaches, not only distance information are taken into consideration, i.e. the chaining-effect (see section 2.1.1) is avoided.

⊕ It has got a good theoretical basis (see especially [185]).

The following disadvantages exist:

⊖ In some cases, the cut between the clusters is not sufficient as a cluster criterion: Figures 2.11 and 2.12 show two clusterings at the minimum cut; obviously only the left clustering is appropriate.

⊖ MinCut clustering is a greedy algorithm and cannot modify earlier decisions. The clustering in figure 2.12 is an example: It would make sense to revoke the decision and to choose a different minimum cut.

⊖ Extra parameters are necessary to define the optimal number of clusters.

⊖ Its run-time behaviour makes it inappropriate for larger graphs. In [79], an overview over the complexity of determining a minimum cut is presented. E.g. in [117] an efficient algorithm is given which needs $O(|V||E|+|V|^2 \log |V|)$ time ($O(|E|)$ space). A probabilistic algorithm needing $O(|V|^2 \log^3 |V|))$ steps ($O(|V|^2)$ space) has been introduced in [85].

For the following run-time estimations, we assume a run-time complexity of $O(|V|^3)$ for the minimum cut algorithm (dense graphs). If the partitions are equally sized, a maximum of $\log |V|$ calls to the minimum cut algorithm is necessary. By solving the resulting recursion formula $T(|V|) = |V|^3 + 2T(\frac{|V|}{2})$, we get an overall run-time of $O(|V|^3)$. If the partitions are not equally sized, a recursion formula $T(|V|) = |V|^3 + T(|V|-1)$ follows and an overall run-time of $O(|V|^4)$.

For non dense graphs, we assume a run-time complexity of $O(|V|^2 \cdot \log |V|)$. This results for equally sized partitions in a recursion formula $T(|V|) = |V|^2 \log |V| + 2T(\frac{|V|}{2})$ and in an overall complexity of $O(|V|^2 \log |V|)$. For arbitrary partition sizes, we get an overall run-time of $O(|V|^3 \log |V|)$

Remark 2.1.1 *In [111], Sven Meyer zu Eissen introduced a MinCut-Clustering method which uses the Λ^* value (see section 2.2.1) to determine when to stop further cluster divisions.*

2.1.3 Optimization Approaches

Additional Knowledge:	explicit criterion
Criterion:	all

Optimization approaches use a given function $f : \mathcal{C}(G) \rightarrow \mathbf{R}$ to rate the clustering of a graph $G = (V, E_c, w)$. By means of optimization methods, a clustering is identified which maximizes f. These methods rely on the given graph only, but they demand the explicit definition of the quality criterion.

Unfortunately, the sheer number of possible clusterings makes a simple "try all, keep best"-approach unfeasible. In [108], Liu found the number of possible partitions of $G = (V, E_c)$ into g clusters to be equal to:

$$N(|V|, g) = \frac{1}{g!} \sum_{i=0}^{g} (-1)^{g-i} \binom{g}{i} i^{|V|}$$

Since g is normally not known, the overall number of possible clusterings follows as:

$$N_{all}(|V|) = \sum_{j=1}^{|V|} N(|V|, j)$$

$\|V\|$	$N_{all}(\|V\|)$
10	115 975
15	1 382 958 545
20	51 724 158 235 372
25	4 638 590 332 229 999 353

Table 2.1: Complexity of the Optimization problem

Table 2.1 gives for some graph sizes $|V|$ the resulting $N(|V|)$. For complexity reasons, most methods start with an initial clustering and refine it using a hill-climbing technique. An overview can be found in [41, 54].

The quality functions used in most applications are mainly variations of the same idea: Maximize the edge density within clusters and minimize the edge density between clusters. The edge density of a graph $G = (V, E_c)$ is defined as $\frac{|E_c|}{|V|}$. Such approaches can be found in [41, 54, 7]. Most of these methods demand that the number of clusters is given.

In [143], a method combining the inter-cluster cut and the cluster sizes has been used.
The advantages of this method are:

- $\oplus$ The user can define the quality of a clustering explicitly.
- $\oplus$ All knowledge from the field of optimization can be applied.

Optimization methods are seldomly used, because of the following disadvantages:

- $\ominus$ For run-time reasons, the optimal solution is almost never found.
- $\ominus$ If fast optimization techniques as hill-climbing are used, the solution is often suboptimal.
- $\ominus$ Researchers do not agree on a good quality function.

2.1.4 Self-Organizing Approaches

Additional Knowledge:	none
Criterion:	inter-cluster distances

The best known self-organizing clustering approach is due to Kohonen ([9, 96]). The main idea is to mark a subset of the vertices as specimen vertices. These vertices define clusters implicitly: Each vertice belongs its closest specimen vertice.

Initially, a fraction of the vertice is marked randomly as specimen vertices. In each step, every specimen vertice is moved into the center of its cluster. The center of a cluster C_i is the node $v \in C_i$ which minimizes $\sum_{v' \in C_i} dist(v, v')$ where $dist$ denotes the graph-theoretic distance between v and v' (see definition B.0.4).

Figure 2.13 shows a typical start situation. The white vertices are the specimen vertices and their clusters are outlined. In the next step (figure 2.14), the specimen vertices are moved into the center of their cluster.

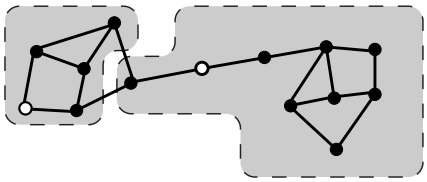

Figure 2.13: Kohonen Clustering: Step 1

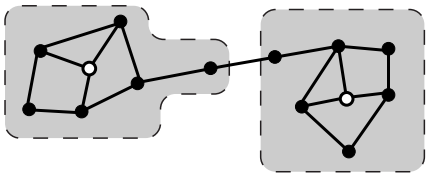

Figure 2.14: Kohonen Clustering: Step 2

When two specimen vertices move too close together, they are replaced by one single specimen vertice. Sometimes a new specimen vertice is added, if the distance between two neighbouring specimen vertices exceeds a given threshold. Therefore, this algorithm is able to find the optimal number of clusters.

Other self-organizing approaches are given in [44, 110, 121]. An extension to non-spherical clusters can be found in [56].

Sometimes the problem is stated as follows: Find k (k given) specimen vertices $S = \{s_1, \ldots, s_k\}$, so that the term $\sum_{s \in S} \sum_{v \in C_s} dist(s, v)$ where C_s denotes the cluster implied by s and $dist$ denotes the length of the shortest path from s to v is minimal. This problem is called k-means clustering (see [114] for a good introduction) and can also be solved using optimization methods (see section 2.1.3).

Kohonen-Clustering possesses some nice features:

⊕ The number of clusters can be found automatically.

⊕ The theoretical background of the method is well funded.

Some disadvantages reduce the applicability of this method:

⊖ The algorithm needs predefined threshold values in order to define when to split and when to unite two specimen vertices.

⊖ Only circular clusters can be found.

⊖ The run-time behaviour is unsatisfiable: Calculating the centroid of a cluster with m nodes takes $O(m^2)$ times if all nodes distances are known. Node distances can be computed (e.g. with the Floyd-Warshall algorithm) in $O(|V|^3)$. If the algorithm needs k iterations, an overall runtime complexity of $O(|V|^3 + k \cdot m^2)$ results. Most practical tests have shown that in most cases at least $|V|$ iterations were necessary.

2.1.5 Other Approaches

In the last decades, most of the basic algorithms presented in the last sections have been enhanced and extended. Furthermore, several different strategies have been developed. This section gives a brief overview of these methods.

In the last few years, three-way or multilevel approaches have been developed: First of all, a graph is coarsed, i.e. several nodes are merged into one. The resulting coarsed graph is clustered and the result is mapped back onto

the original graph. Such methods can be found e.g. in [86, 66, 87]. These methods normally apply standard techniques (e.g. nearest-neighbour or optimizing approaches) for the coarsening and the clustering step.

Non-disjunctive or fuzzy clustering finds overlapping clusters, examples of which are given in [52, 186, 21, 177].

In the field of statistics, several algorithms for clustering already laid out graphs exist. These approaches normally presume extra information about the clusters (distribution of the nodes, shape of the clusters, etc.). Most such systems combine statistical methods with algorithms presented in the previous sections. Details and overviews can be found in [41, 54, 142, 182, 6, 62].

Some applications constrain the cluster sizes. E.g. in VLSI or parallel computing clusters with similar sizes are needed. This version of the clustering problem is most times called "partitioning". Introductions to this field and to different methods can be found in [104, 93, 31, 66, 65, 126, 77].

2.2 Natural Clustering

In this work, clustering is applied first of all as a means of data mining, thus so called "Natural Clusters" are wanted. Such clusters are defined by the graph's structure only, i.e. no node or edge labels are used. Natural clusters correspond to clusters as identified by a human analyzer not using any additional domain knowledge.

The algorithms described in section 2.1 are in most cases not able to find natural clusters[3]. In this section, two new algorithms for detecting natural clusters are presented, one theoretical method (section 2.2.2) and one fast clustering algorithm (section 2.2.1).

Many attempts exist to define natural clusters; some formal definitions have been mentioned in section 2.1.3. Godehardt defines a cluster in [54] as follows:

> "Whatever the case, the clusters should be chosen so that the objects within a cluster are mutual similar (**homogeneity within the classes**) while the representatives of distinct classes are mutual dissimilar (**heterogeneity between the classes**)."

T. Roxborough and A. Sen outline in [143]:

> "In spite of the differences of opinions as to what constitutes a cluster, one idea is universally accepted: the nodes belonging to a cluster must have strong relationship between them in comparison with the nodes outside the cluster."

In [119] a similar statement can be found:

> "The goal of a clustering algorithm is to automatically identify a set of nodes / or edges that are more strongly connected with each other than with the rest of the graph."

Such statements can be found in a large number of work concerning clustering, all of which refer to the definition of natural clusters.

[3] Reasons for this are discussed in section 2.4.

2.2.1 MajorClust

Additional Knowledge:	none
Criterion:	inter-cluster connections

The Algorithm

In this section, a fast new clustering algorithm (see also [160]) is presented:

Initially, the algorithm assigns each node of a graph its own cluster. Within the following re-clustering steps, a node adopts the same cluster as the majority of its neighbors belong to. If there exist several such clusters, one of them is chosen randomly. If re-clustering comes to an end, the algorithm terminates.

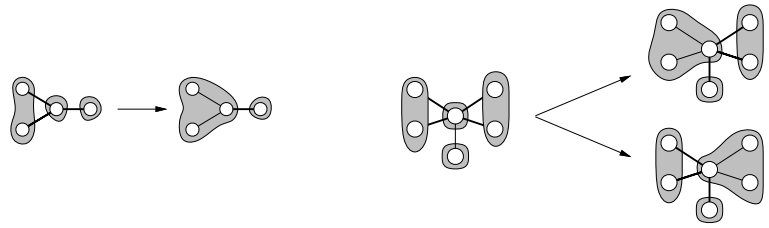

Figure 2.15: A definite majority clustering situation (left) and an undecided majority clustering situation (right).

The left hand side of Figure 2.15 shows the definite case: most of the neighbors of the central node belong to the left cluster, and the central node becomes a member of that cluster. In the situation depicted on the right hand side, the central node has the choice between two different clusters.

We now write down this algorithm formally.

Algorithm 2.2.1 (MAJORCLUST**)**
Input. A graph $G = \langle V, E_c \rangle$.
Output. A function $c : V \mapsto \mathbf{N}$ *which assigns a cluster number to each node.*

(1) $n = 0$, $t =$ false
(2) $\forall v \in V$ ***do*** $n = n + 1$, $c(v) = n$ **end**
(3) ***while*** $t =$ false ***do***
(4) $t =$ true
(5) $\forall v \in V$ ***do***
(6) $c^* = i$ ***if*** $\left|\left\{u : \{u, v\} \in E \wedge c(u) = i\right\}\right|$ *is max.*
(7) ***if*** c^* *is a unique solution* ***then*** $t =$ false
(8) ***if*** $c(v) \neq c^*$ ***then*** $c(v) = c^*$
(9) ***end***
(10) ***end***

Remark 2.2.1 *Choosing a node* $v \in V$ *in step* 5 *and choosing between clusters* i *with the same size of* $\left\{u \mid \{u, v\} \in E \wedge c(u) = i\right\}$ *in step* 6 *must be totally randomly.*

Theorem 2.2.1
MAJORCLUST *terminates after* $O(|E_c|)$ *definite majority decisions. The cluster change of a node is a definite majority decision if and only if it has happened in a definite majority situation (see figure 2.15).*

Proof.
Let $G = (V, E_c)$ *be a visualization graph.* $f(v), v \in V$ *is defined as the number of nodes adjacent to* v *with a different cluster membership:*
$f(v) = |\{\{v, w\} \mid \{v, w\} \in E_c, w \in V, c(v) \neq c(w)\}|$. $F(G)$ *is twice the overall number of inter-cluster edges:* $F(G) = \sum_{v \in V} f(v)$. *Obviously* $F(G) \leq 2|E_c|$ *holds for all visualization graphs* G.

Let $v \in V$ *be a node which is about to change its cluster membership in a definite majority decision. Without loss of generality let* $\{1, \ldots, k\}, k \in \mathbf{N}$ *denote the clusters adjacent to* v. $n_i, 1 \leq i \leq k$ *is defined as the number of nodes adjacent to* v *which belong to cluster* i. *Without loss of generality, we assume* $n_1 \geq n_2 \geq \ldots \geq n_k$. *Since* v *will change its cluster membership in a definite majority situation,* $c(v) = j, j \geq 2$ *and* $n_1 > n_j$ *follows.*

Let ΔF *denote the change of* F *caused by the cluster change of* v. F *is decreased by* $n_1 - n_j > 0$. n_1 *neighbours decrease their* f *value by 1, because* $c(v) = 1$ *after the cluster change of* v. n_j *neighbour increase their* f *value by 1. Thus:* $\Delta F = -(n_1 - n_j) - n_1 + n_j = 2n_j - 2n_1 < 0$.

The reader may note that $F(G)$ *does not change for non-definite majority situations in step* 6*: If* v *keeps its old cluster membership, no changes happen. In an undecided majority clustering (see figure 2.15),* $n_1 = n_j$ *holds and* $\Delta F = 0$ *follows.*

Since initially $F(G) \leq 2|E_c|$ *holds,* MAJORCLUST *terminates after* $O(|E_c|)$ *definite majority decisions.*

The following corollary analyses the run-time behaviour of algorithm 2.2.1.

Corollary 2.2.1 MAJORCLUST *terminates, when for* $|V|$ *steps (step 6 of algorithm 2.2.1) no definite majority decision has happened. Therefore,* MAJORCLUST *terminates after* $O(|V| \cdot |E_c|)$ *steps.*

Proof.
Follows directly from theorem 2.2.1.

A faster variation of MAJORCLUST is introduced by the next corollary.

Corollary 2.2.2 *If* MAJORCLUST *terminates, when for* $p, p \in \mathbf{N}$ *(*p *constant) steps (step 6 of algorithm 2.2.1) no definite majority decision has happened,* MAJORCLUST *terminates after* $O(|E_c|)$ *steps.*

Proof.
Follows directly from theorem 2.2.1.

Remark 2.2.2 MAJORCLUST *does stop, even when in the last* $|V|$ *steps an undecided majority clustering situation has happended (such situations are defined in figure 2.15). This problem may be solved by randomizing the edge weights, i.e. by adding a small, random value to the edge weights. For such randomized graphs, undecided majority clustering situation are very unprobable, i.e.* MAJORCLUST *becomes (almost) determininistic.*

Remark 2.2.3 MAJORCLUST *can also use the edge weights w: Each neighbour v_1 of a node v_0 is assessed using the weight of the edge $\{v_0, v_1\}$.*

Remark 2.2.4 *In order to find subclusters within clusters, most cluster algorithm (including* MAJORCLUST*) can be extended for weighted graphs as follows: After the clusters are identified, all edge weights are squared. This operation increases the differences between large and small edge weights. Now all subgraphs induced by the previous clustering are clustered again.* MAJORCLUST *is specially suited for this approach since is automatically detects when no more clusters can be found. This version of* MAJORCLUST *is called hierarchical* MAJORCLUST.

Properties of MAJORCLUST

In the rest of this section some hints will be given why MAJORCLUST finds reasonable clusters.

Definition 2.2.1 (Stable Regions of a Graph)
Let $G = (V, E_c, w)$ be a graph. A connected subgraph $G' = (V', E')$ of G induced by $V' \subseteq V, |V| > 1$ with $c(v) = a, v \in V', a \in \mathbf{N}$, where c denotes the cluster function from algorithm 2.2.1 is called a stable region iff no further iterations of MAJORCLUST *can change $c(v)$ for all $v \in V'$.*

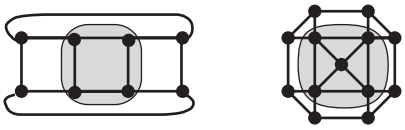

Figure 2.16: *Two Stable Regions*

Figure 2.16 shows two stable regions.

Remark 2.2.5 *Clusters found by* MAJORCLUST *are always stable regions.*

Remark 2.2.6 MAJORCLUST *may not find all stable regions.*

Definition 2.2.2 (Border of a Subgraph)
Let $G = (V, E_c, w)$ be a graph and $G' = (V', E')$ a stable region in G. The border of V' is defined as follows:
$border(V') = \{v' \in V' \mid \exists \{v', v_2\} \in E_c$ *with* $v_2 \notin V'\}$

Corollary 2.2.3 *Let $G = (V, E_c, w)$ be a graph and $G' = (V', E')$ a stable region in G. Furthermore let $cut(V')$ be the cut[4] of V' and let $\Psi \subseteq E_c$ be defined as follows: $\Psi = \{\{v_1, v_2\} \in E_c \mid v_1, v_2 \in border(V')\}$. Then*

$$cut(V') < cut(V' \setminus border(V')) + 2 \cdot |\Psi|$$

holds.

Proof.
$\sum_{v' \in border(V')} |\{v', v_2\} \in E_c, v_2 \in V'| = cut(V' \setminus border(V')) + 2 \cdot |\Psi|$
because in the sum above some edges are counted twice, furthermore:

[4]The definition of the term cut can be found in definition B.0.7

$\sum_{v' \in border(V')} |\{v', v_2\} \in E_c, v_2 \in V'| > cut(V') = \sum_{v' \in V'} |\{v', v_2\} \in E_c, v_2 \notin V'|$
(follows directly from the definition of a stable region)

The theorem above relates the cut at both sides of a border and the number of edge within the border (Ψ). Two situations are possible: (*i*) Only few edges are within the border, i.e. we can assume $cut(V') \lessapprox cut(V' \setminus border(V'))$. In that case, the stable area is a reasonable cluster (see beginning of section 2.2 for a definition of the term "reasonable cluster", similar cluster properties have been used in [188, 99, 98]). (*ii*) Ψ is relatively high, e.g. $\Psi > cut(V' \setminus border(V'))$. Then most edges connected to the border are between nodes within the border and the border itself forms a circular reasonable cluster area. The reader may note that in both cases stable areas are reasonable cluster candidates.

Lemma 2.2.1
Let $G = (V, E_c, w)$ be a connected graph and $G' = (V', E')$ a stable region in G with $G \neq G'$. Then G' contains a cycle[5].

Proof.
The following holds:
(i) $\forall v \in V'$: v has at least one edge to another node within V' (since G' is connected) $\Rightarrow |E'| \geq |V'| - 1$.
(ii) $v \in border(V')$: Such a node v exists, because $G \neq G'$ holds. Because of $|\{v, v_2\} \in E_c, v_2 \in V'| > |\{v, v_3\} \in E_c, v_3 \notin V'| \geq 1$, $|E'| \geq |V'|$ can be concluded. It follows directly, that G' contains a cycle (see [80], theorem 1.2.3).

Corollary 2.2.4 *From lemma 2.2.1 follows, that every stable region of a graph G contains at least as many nodes as the smallest cycle in G.*

The reader may note that corollary 2.2.4 makes sense: Graphs which contain only large cycles do not have the property of "strong local relations" between groups of nodes, i.e. they contain none or only few clusters. The corollary makes the same statement about stable areas.

At the beginning of section 2.2, the author has quoted some opinions about the features of optimal natural clusters. One possible conclusion is the demand of a relatively small cut between different clusters. It can be shown that the cut between two stable regions is also relatively small (see also corollary 2.2.3 and [188, 99, 98]).

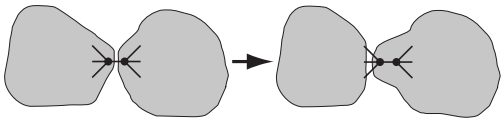

Figure 2.17: *Node exchange and cut.*

Theorem 2.2.2
Let $G = (V, E_c, w)$ be connected graph and be $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two neighboured stable regions in G. Then

$$cut(V_1, V_2) < cut(V_1 \setminus \{v\}, V_2 \cup \{v\}) \, \forall v \in V_1$$

[5] A cycle is defined in definition B.0.2

holds.

Proof.
For all $v \in V_1$ holds: $|\{v, v'\} \in E_c, v' \in V_2| > |\{v, v'\} \in E_c, v' \in V_1|$. The edges $\{v, v'\} \in E_c, v' \in V_2$ are part of $cut(V_1, V_2)$. If v is moved to cluster V_2, those edges are replaced by the edges $\{v, v'\} \in E_c, v' \in V_1$, i.e. the cut increases. This is also illustrated by figure 2.17.

A Statistical Model for MAJORCLUST

In this short section, a statistical model will be presented, that may help to explain the behaviour of MAJORCLUST. First of all a statistical model for the number of cluster will be given. In addition, an empirical evaluation will support this model.

Theorem 2.2.3
Let $G = (V, E_c, w), V = (v_1, \ldots, v_n)$ be a connected graph with a constant node degree of d. Then after one turn of MAJORCLUST *(one turn is finished when all node cluster memberships have been adjusted), $O(\frac{|V|}{d})$ expected clusters will be left.*

Proof.
The only possibility for a node to keep its cluster membership is to give its cluster membership first to a neighbour and then to get it back afterwards. The probability that the first node (v_1) keeps its cluster membership is 0. The probability that v_2 keeps its cluster membership is $\frac{d}{n-1} \cdot \frac{1}{d} \cdot \frac{1}{d} = \frac{1}{(n-1)d}$, because $\frac{d}{n-1}$ is the probability that v_2 is a neighbour of v_1 and $\frac{1}{d}$ is the probability that v_1 took the cluster membership of v_2 and $\frac{1}{d}$ is also the probability that v_2 got the same cluster membership back from v_1.

The probability that v_i keeps its cluster membership is then equal to $\sum_{1 \leq j \leq i} \frac{1}{(n-1)d} = i \cdot \frac{1}{(n-1)d}$. So it follows that after one turn $\sum_{1 \leq i \leq n} i \cdot \frac{1}{(n-1)d} = O(\frac{|V|}{d})$ expected clusters are left.

To verify this theorem empirically, MAJORCLUST has been applied to random graphs. These random graphs have to fulfill two conditions: (*i*) A constant node degree of d and (*ii*) they have to comprise structure, i.e. areas with a density above average. Meeting the last condition is a special challenge because normally structure is a property of graphs that have been used to model a real-world problem. The domain structure is then imprinted into these graphs.

The graphs used for the evaluation have been created using the following function:

Algorithm 2.2.2 (CreateRandomGraph)
Input *The number of nodes k^2, a node degree d*
Used Functions
1) $p(x) = 2\pi \cdot \varphi(\frac{x}{\frac{k}{5}})$
where φ denotes the standardized normal distribution
$\varphi(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$
2) *$dist(v_1, v_2)$ denotes the Euclidean distance between the nodes v_1, v_2*
Output *A graph $G = (V, E_c, w)$ with $|V| = k^2$.*

(1) *set* $G = (V, E_c)$ *with* $V = \{\}, E_c = \{\}$
(2) **for** $x = 1$ **to** k **do**
(3) **for** $y = 1$ **to** k **do**
(4) *create a new node* v
(5) $\rho(v) = (x, y)$ $/*$ $\rho(v)$ *denotes the position of node* v $*/$
(6) $V = V \cup \{v\}$
(7) **end**
(8) **end**

(9) **do**
(10) **for all** $v_1, v_2 \in V, v_1 \neq v2$ **do**
(11) **if** $degree(v_1) < d$ **and** $degree(v_2) < d$ **then**
(12) *With a probability* $p(dist(v_1, v_2))$ *add the edge* $\{v_1, v_2\}$ *to* E_c
(13) **end**
(14) **until** $\nexists v_1, v_2 \in V : p(dist(v_1, v_2)) > 0.01$

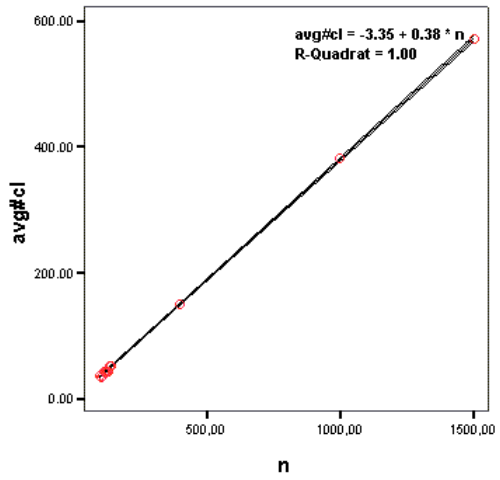

Figure 2.18: *The average number of clusters for growing* n

The algorithms above creates first of all a grid of k^2 nodes (steps 1-8), i.e. each node has position within a $k \times k$ grid. In addition, an edge $\{v_1, v_2\}$ is created with a probability which depends on the Euclidean distance between v_1 and v_2 (step 12). This guarantees for the creation of neighbourhoods which are connected higher-than-average. An edge is not created if it would result in a node degree higher than d (step 11).

Figure 2.18 shows the average number of clusters $(avg\#cl)$ after one turn of MAJORCLUST over the number of nodes (n) for a node degree of 5. Its linear nature is obvious. Figure 2.19 depicts the average number of clusters $(avg\#cl)$ over the node degree (deg) for 1000 nodes. The dependent variable is shown using a logarithmic axis. It can be seen that the function behaves like $\frac{1}{deg}$ for larger deg. The diagrams do not change for different n.

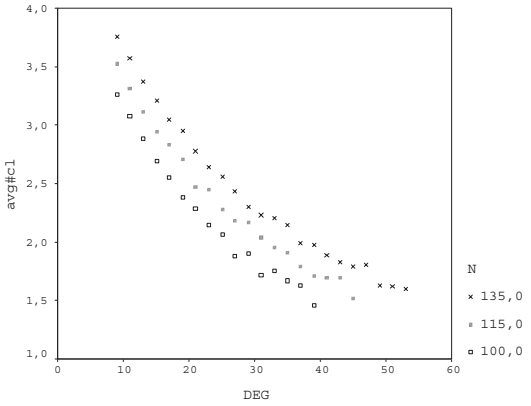

Figure 2.19: *The average number of clusters for growing node degrees*

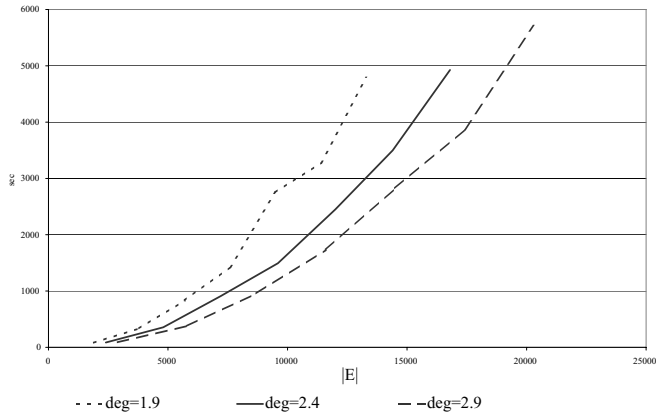

Figure 2.20: *Run-time behaviour of* MAJORCLUST *for different numbers of edges.*

These randomly created graphs also allow for an estimate of run-time behaviour: Figure 2.20 depicts for large artificial graphs the required run-time[6] over the number of edges. Three different average node degrees are shown. Since more clusters exist for graphs with lower average node degrees (see explanations above), it makes sense that for such graphs the run-time increases. Obviously MAJORCLUST behaves very nicely.

2.2.2 Λ-Maximization

Additional Knowledge:	none
Criterion:	Λ^*

Λ-Maximization is based on the idea of weighting each node by the connectivity of its appurtenant cluster. By maximizing the sum of node weights, the resulting clusters closely resemble the human idea of an ideal graph decompositon.

The *edge connectivity* $\lambda(G)$ of a graph $G = (V, E_c)$ denotes the minimum number of edges that must be removed to make G a not-connected graph: $\lambda(G) = \min\{|E^{'}| : E^{'} \subset E_c \text{ and } G^{'} = (V, E_c \setminus E') \text{ is not connected}\}$.

[6] Clisp on a Pentium II, 400 Mhz has been used.

Definition 2.2.3 (Λ)
Let $G = (V, E_c)$ be a graph, and let $C = \{C_1, \ldots, C_n\}$ be a disjunctive clustering of G. The weighted partial connectivity *of C, $\Lambda(C)$, is defined as*

$$\Lambda(C) := \sum_{i=1}^{n} |C_i| \cdot \lambda_i, \quad \textit{where}$$

$\lambda(C_i) \equiv \lambda_i$ designates the edge connectivity of $G(C_i)$, where $G(C_i)$ is the subgraph induced by C_i (see definition B.0.6).

Figure 2.21 illustrates the weighted partial connectivity Λ.

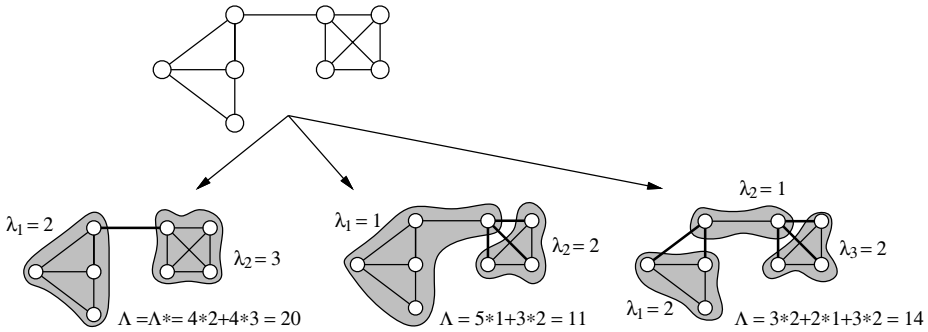

Figure 2.21: *Graph decompositions and related Λ values.*

Definition 2.2.4 (Λ-Structure)
Let $G = (V, E_c)$ be a graph, and let C^* be a disjunctive clustering of G that maximizes Λ:

$$\Lambda(C^*) \equiv \Lambda^* := \max\{\Lambda(C) \mid C \textit{ is a disjunctive clustering of } G\}$$

Then the structure $S(G, C^*)$ (see definition 2.0.4) is called Λ-structure *of the system represented by G.*

Figure 2.22 shows that Λ-maximization means structure identification.

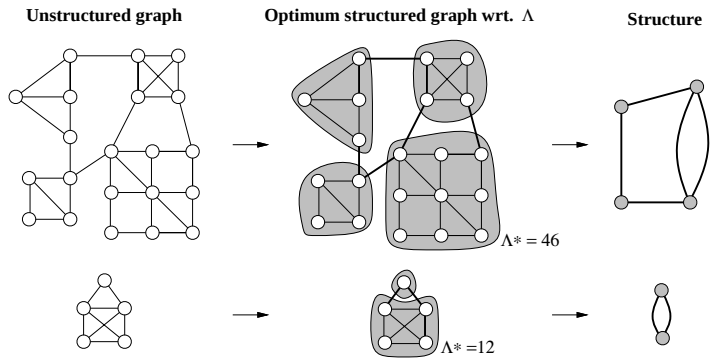

Figure 2.22: *Examples for decomposing a graph according to the structure definition.*

Remark 2.2.7 *A key feature of the above structure measure is its implicit definition of a structure's number of clusters.*

Two rules of decomposition, which are implied in the structure definition, are worth to be noted. (*i*) If for a (sub)graph $G = (V, E_c)$ and a disjunctive clustering $\{C_1, \ldots, C_n\}$ the *strong splitting condition*

$$\lambda(G) < \min\{\lambda_1, \ldots, \lambda_n\}$$

is fulfilled, G will be decomposed. Note that (*i*) the strong splitting condition is commensurate for decomposition. Obviously this splitting rule follows the human sense when identifying clusters in a graph.

(*ii*) If for no disjunctive clustering C the strong splitting condition holds, G will be decomposed only, if for some C the condition $|V| \cdot \lambda(G) < \Lambda(C)$ is fulfilled. This inequality forms a necessary condition for decomposition.

The weighted partial connectivity, Λ, can be made independent of the graph size by dividing it by the graph's node number $|V|$. The resulting normalized Λ value is designated by $\bar{\Lambda} \equiv \frac{1}{|V|} \cdot \Lambda$.

It is useful to extend the structure identification approach by integrating the edge weights w. For this a generalization of $\Lambda(C)$ is introducing: the *weighted edge connectivity* $\tilde{\lambda}$ of a graph. It is defined as follows:
$\tilde{\lambda}(G) = \min\{\sum_{e \in E'} w(e) : E' \subset E_c$ and $G' = \langle V, E_c \setminus E' \rangle$ is not connected$\}$.
Using this definition, all previous results can be directly extended to graphs with edge weights.

The following theorem relates Min-cut-clustering (see section 2.1.2) to clustering by means of Λ-maximization.

Theorem 2.2.4 (Strong Splitting Condition)
Let $G = (V, E_c)$ be a (sub)graph and $C = \{C_1, C_2\}$ a disjunctive clustering in two *clusters. Then applying the strong splitting condition ($\lambda(G) < \min\{\lambda_2, \lambda_2\}$) results in a decomposition at a minimum cut.*

Proof of Theorem. Every clustering $C' \in \mathcal{C}(G), C' \neq C$ would decompose C_1, C_2, or both. Since $\min\{\lambda_2, \lambda_2\} > \lambda(G)$, C' can not be a clustering at a minimum cut. Therefore only C can be a decomposition at a minimum cut.

Remark 2.2.8 *When the strong splitting condition does not hold, an optimum decomposition according to the structuring value need not be the same decomposition as found using the minimum cut. This is because of the latter's disregard for cluster sizes. Figure 2.23 is such an example. Here C_x refers to a clique with $x \geq 3$ nodes. An optimum solution according to the weighted partial connectivity Λ (which is also closer to human sense of esthetics) consists of one cluster $\{v_1, v_2, v_3, v_4\}$ and a second cluster C_x. An algorithm using the minimum cut would only separate v_1.*

The reader may also notice that, as mentioned before, maximizing the weighted partial connectivity implies an optimum number of clusters, while most other known clustering algorithms need additional parameters defining more or less explicitly the number of clusters.
The main advantages of this method are:

$\oplus$ The Λ-value strongly resembles the human assessments of clusterings.

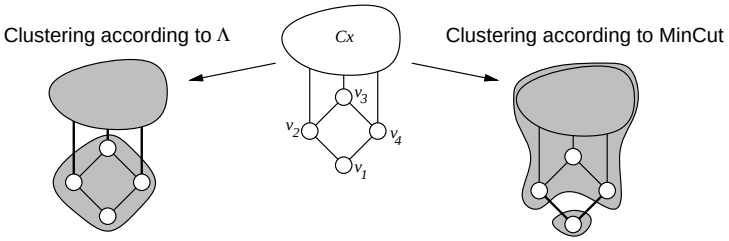

Figure 2.23: Weighted partial connectivity (Λ-) maximization versus Min-Cut-clustering.

Figure 2.24: *The problematic cases for the Λ-maximization.*

⊕ The number of clusters is defined implicitly.

The disadvantages of Λ-maximization are:

⊖ No fast method for finding the Λ-structure exists yet.

⊖ Λ-maximization fails to find the optimal clustering in some cases. Figure 2.24 shows two examples: While the optimal clustering on the left side can still be found by using the node connectivity instead of the edge connectivity[7], both variations fail for the graph on the right hand side.

2.3 Adaptive Clustering

In some domains, additional knowledge can be integrated into the clustering process. The most popular approach works by using node labels: Heuristic rules and algorithms are defined which compute clusters by means of graph-theoretic information and by using given node labels. An example can be found in [149, 163]: Special clusters (so-called hydraulic axis) are found by using graph grammars and path-search techniques, both of which rely heavily on node labels.

The main drawback of this approach is the time needed to define the rules and algorithms. In the example mentioned above, it took one master thesis and additional work of approximately 4 month to develop a working system. The main reason for this is that using node labels leads to a knowledge acquisition problem: The necessary knowledge has to be extracted from an expert, who in most cases is not explicitly aware of his knowledge.

One solution is to use Machine Learning. For this, the expert gives a typical set of clustered and labeled graphs and a system tries to conclude domain specific clustering knowledge from the examples.

[7] See [80] for an introduction to node connectivity

Such a system has been developed by the author. A description can be found in section 6.4.1[8].

2.4 Evaluation

As outlined in section 1.7, the methods described in this chapter are rated by the four features introduced at beginning of chapter 1. This allows for an assessment whether the AI-visualization paradigm has been followed.

1. **Object-based Presentation**: All clustering methods result in a clustering of objects. Even after the clustering step, the individual object do not vanish. Unlike methods where the cluster replaces the objects, clustering information is used here only as additional information.

3. **Abstraction:** Clustering is well suited to abstract knowledge coded into a graph. It is based on the assumption that edges connect nodes which somehow belong together, i.e. highly connected subgraphs form clusters.

   Clustering imitates the human approach of comprehending complex systems: find reasonable subsystems and work on different levels of abstractions. I.e. either subsystems are examined separately or information within subsystems is disregarded and only the global structure is analyzed. Therefore, the approach used in this work can be viewed as a typical AI-method: identify the human problem solving process and find an algorithm which emulates it.

   The reader may note that comparing different clustering algorithms is difficult since no generally accepted quality measure exists. The author personally prefers the Λ-value as a quality measure, but that quality function is neither generally accepted nor does it work in all cases (see section 2.2.2 for details). Therefore, only two methods remain to compare clustering algorithms:

   (a) Theoretical evaluations (as have been given in this chapter) may help to choose the most suitable algorithm. The abstraction quality of clustering algorithms depends mainly on the clustering criterion. Some criterions have obvious disadvantages, e.g. the distance information used for nearest-neighbour clustering. Optimization approaches suffer from the problem that no generally applicable quality measure exists. MAJORCLUST, Λ-Optimization, Divisive-Clustering, and Self-Organizing-Clustering are harder to compare; all algorithms have specific advantages and disadvantages. Those four algorithms must be compared using real-word problems (see below).

   Run-time considerations also influence the choice between different methods. From that point of view, MAJORCLUST and Nearest-Neighbour clustering should be prefered. Divisive and Self-Organizing-Clustering can still be applied to small problems, while Λ-Optimization and general optimization approaches are normally not suitable for larger graphs.

[8]The description has been placed in the chapter “Hydraulic Circuits” because it has mainly been applied to that domain.

(b) Clustering algorithms can also be compared using real-world problems: In chapters 6 and 7, six such applications are described.

MAJORCLUST is applied successfully to all six domains. Self-organizing approaches also prove helpful, but normally have problems finding the optimal number of clusters. Divisive algorithms suffer from an unsatisfiable run-time behaviour and from the problem described in figure 2.12. For some small graphs, this approach shows promising results (e.g. in section 6.4).

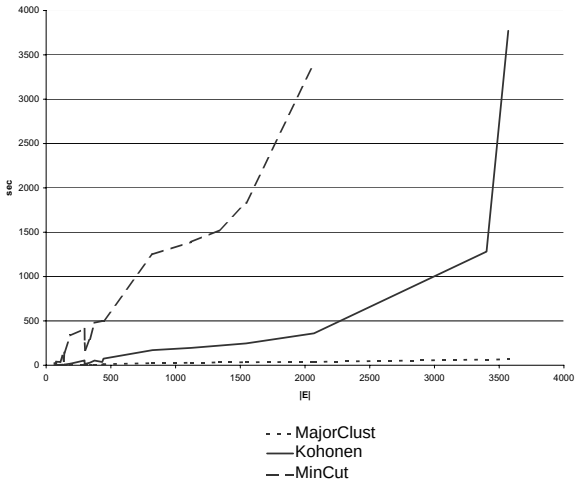

Figure 2.25: *Run-time behaviour of three clustering algorithms. Only clustering results have been used which passed a visual inspection.*

Some of the results from chapters 6 and 7 are summarized below: Figure 2.25 compares the run-time behaviour of MAJORCLUST, self-organizing clustering, and MinCut Clustering (according to remark 2.1.1). It shows the run-time[9] (in seconds) over the number of edges.

One of the main problems concerning the run-time comparison of clustering algorithms is that in many cases bad quality causes a fast performance. E.g. MinCut-clustering only needs few steps if it carries out only a small number of cluster divisions, but such clusterings are very often suboptimal. The fact that in this work clustering is applied to several domains allows for a different approach: 52 typical graphs from all six domains are clustered and the results are verified by a user. Only such graphs which pass this manual inspection are used for the run-time comparison. The reader may note that this procedure still causes some problems: (*i*) The quality of the results is disregarded. E.g. algorithm A may need 15% less time than algorithm B, but its quality might be 30% worse than the quality of algorithm B. (*ii*) The graphs from the six domains are not typical for all graph classes. (*iii*) Several versions and implementations of the same clustering algorithm exist. This test may disregard some improvements to the standard algorithms.

[9]The tests have been carried out on a Pentium II, 400Mhz using CLisp.

Nevertheless, figure 2.25 may help to give a notion of the different run-time behaviour. Obviously, MAJORCLUST shows the best performance while self-organizing clustering and especially MinCut Clustering are not suited for larger graphs. The run-time behaviour of MAJORCLUST is also evaluated using artificial graphs in section 2.17.

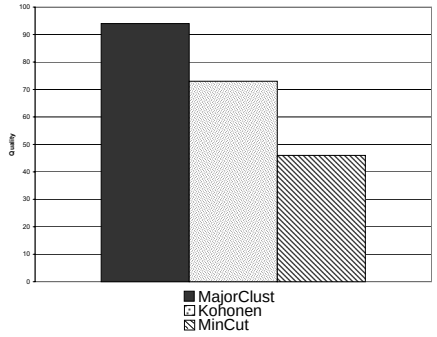

Figure 2.26: *The quality of the results from different clustering algorithms.*

Figure 2.26 presents the percentage of accepted clustering results. Obviously, MAJORCLUST is able to identify reasonable clusters in most cases while self-organizing clustering and especially MinCut-Clustering have more problems. The unsatisfactory quality of MinCut Clustering is mainly due to the problem described in figure 2.12. Kohonen clustering suffers from two main problems: (*i*) It often has problems finding the optimal number of clusters and (*ii*) non-circular clusters can not be detected.

Run-time behaviour and clustering quality are discussed in chapters 6 and 7 for each domain separately.

4. **User Adaption:** In most cases, all necessary knowledge for identifying natural[10] clusters is coded into the graph's structure. Therefore, any additional parameters should be viewed as a weakness of the clustering algorithms. In other words, the best user-adaption in clustering is no user-adaption. Thus, MAJORCLUST and Λ-optimization and (partially) some self-organizing approaches are optimal with respect to user-adaption. All other algorithms rely on extra information.

   In some cases, the information coded into the edges and edge weights in not sufficient, i.e. non-natural clusters are wanted. In that case either specialized algorithms or algorithms with parameters have to be used. When specialized methods have to be found, the application of cluster learning as described in section 2.3 and 6.4.1 allows for the definition of cluster detection methods on an abstract level, thus supporting the knowledge acquisition and user-adaption process.

[10] see section 2.2

Chapter 3

Structure Classification

In this chapter, two methods for the automatic classification of clusters are presented. Section 3.1 discusses direct classification methods, while section 3.2 deals with case-based approaches. Both section also describe algorithms for learning the necessary classification knowledge from given exemplary clusterings.

Cluster labels can support the human understand of complex graphs. E.g. a cluster in a network traffic graph may be identified as “AI Lab” or as “Human Resource Management”[1]. An example drawing can seen in figure 6.8.

Another application is knowledge-bases, e.g. a configuration knowledge-base modeling a technical system. Since such knowledge-bases form a graph, clustering can be used to identify technical subsystem. Labeling these clusters with the name of the corresponding technical subsystem increases the value of the clustering step. This application is described in detail in section 6.2.

Therefore, an automatic labeling or classification of clusters is desirable. This can be done by a classification function mapping clusters onto cluster labels. Manual definitions of such functions are highly complex. Therefore, this work will concentrate on the usage of machine learning algorithms to define classification functions. In this chapter, two classification approaches are presented: (*i*) Direct classification and (*ii*) Case-based Classification.

Typical data structures for graphs and clusters (e.g. adjacency lists or matrices) are not well suited as an input for classification functions for the following reasons:

- ☞ Similar (or even isomorph graphs) can result in different data structures.
- ☞ Users think about cluster similarities in terms of abstract graph features as the number of nodes, diameter, or domain dependent properties. Such features are not part of an adjacency list or matrix.

For these reasons, it makes sense to represent a cluster by a set of abstract graph features. Thus, for each cluster C a vector $\vec{f}(C) \in \mathbf{R}^p$ comprising several graph features is calculated. Table 3.1 shows typical graph features.

The cluster classification function c can now be formally defined:

[1] An extensive description of cluster classification in the domain of network traffic can be found in section 6.1.

Feature
number of connected components
edge connection
number of biconnected components
number of nodes
number of edges
minimum / average / maximum distance between nodes
diameter
minimum / average / maximum node degree
directed / undirected
number of clusters as found by MAJORCLUST

Table 3.1: Important graph features for drawing purposes

Definition 3.0.1 (Cluster Classification Function)
Let C be a cluster of a graph and let $\overrightarrow{f}(C)$ be the feature vector of C. A function $c : \mathbf{R}^* \rightarrow \{l_1, \ldots, l_k\}$, where l_i denotes a cluster label, is called a cluster classification function. $c(\overrightarrow{f}(C))$ denotes the label of cluster C.

In the definition above, only real numbers are allowed as cluster features. This is sufficient for the applications presented in this work. In general, nominal and ordinal features (see also section 5.1) could be used as well.

3.1 Direct Classification

Direct classification tries to map cluster features directly onto cluster labels. Since manually finding such a classification function is a difficult task, the following approach can be used: The user gives a set of clusters and the corresponding labels. By applying Machine Learning, the functional relation between cluster features and cluster labels is determined. An application is given in the next section. In section 6.2, clusters in knowledge-bases are classified.

3.1.1 Learning the optimal Layout Method

In many domains, no optimal graph-drawing method exists, but the user prefers different drawing methods for different graphs or even for different clusters. The idea presented in this section is to analyze the user's preferences (i.e. which method is used for which graph or cluster) and to learn a function which maps a graph or cluster onto a layout method.

The rest of this section elaborates on how the problem of finding a classification function c can be reduced to a standard regression problem, making the automatic learning of c possible.

The main idea is to have the user give some exemplary cluster labelings, learn a correlation between cluster features and labels, and use this learned knowledge to classify new clusters.

The learning process is quite simple: A set of typical clusters $\{C_1, \ldots, C_p\}$ has to be given. For each cluster C, the user chooses his favorite layout method $c(C)$ (see definition 3.0.1). This results in a database DB of classified feature vectors $DB = \{< \overrightarrow{f}(C_1), c(C_1) >, \ldots, < \overrightarrow{f}(C_p), c(C_p) >\}$.

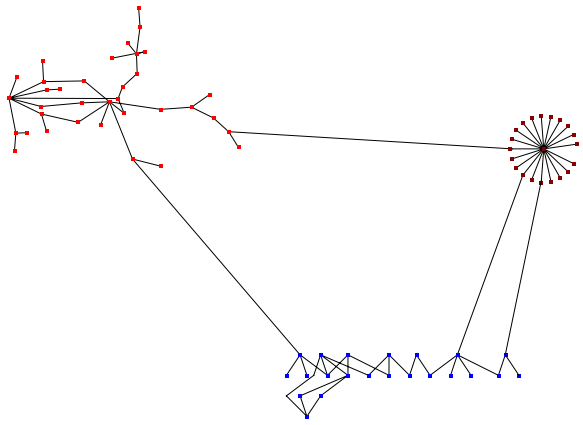

Figure 3.1: *Different graph drawing methods (spring embedder for the top-left cluster, circuit layout on the right, and a level-oriented approach for the bottom cluster) have been chosen automatically according to previous user preferences.*

The reader may note that such classified feature vectors can be obtained during the normal course of work: Every time the user chooses a special layout method, the system analyzes his behaviour. After a while, the system will be able to choose the layout method automatically.

Databases like DB are normal input for standard regression algorithms (see section A.2), i. e. c can be learned by applying regression to DB. c learns which features support or weaken the applicability of a graph drawing method. For runtime reasons, it may be reasonable for large databases and complex feature-vectors to use neural-networks as a heuristic method to solve the regression problem.

This method has been applied in section 6.2 to the visualization of knowledge-bases.

3.2 Case-Based Classification

Case-based Classification can be applied to the problem of cluster classification as follows: Old cluster classifications are saved together with the corresponding clusters. Again, this leads to a database DB of classified feature vectors $DB = \{< \overrightarrow{f}(C_1), c(C_1) >, \ldots, < \overrightarrow{f}(C_p), c(C_p) >\}$. For a new feature vector the most similar existing vector is retrieved and the classification of this vector is also used for the new cluster. In order to find a similar vector, a so-called similarity measure $sim : \mathbf{R}^* \times \mathbf{R}^* \rightarrow \mathbf{R}$ is used. $sim(\overrightarrow{f}(C_j), \overrightarrow{f}(C_j)$ calculates the similarity between two clusters C_i and C_j.

While similarity measures are used here in a different way than in section 5, all definitions, methods, and results from that section can nevertheless be applied to Case-based Classification. Obviously, similarity measures are an instrument used in different areas of computer science. The methods for learning similarity measures from chapter 5 can be especially applied.

Thus, the classification function is defined indirectly: Let $\tilde{C}$ be a new cluster and let DB be defined as above. $c(\tilde{C})$ is defined as $c(\tilde{C}) = \tilde{c}$, where $c(C_i) = \tilde{c}$

and C_i is the cluster in DB which maximizes the function $sim_{1 \leq j \leq p}(\tilde{C}, C_j)$

Case-based Classification is a special form of Case-based Reasoning (CBR). Good introductions to CBR can be found in [159, 97].

In section 6.1.2, Case-based Classification is applied to the identification of clusters in traffic graphs. For this, a manually defined similarity measure has been developed.

3.3 Evaluation

As outlined in section 1.7, the methods described in this chapter are rated by the four features introduced at the beginning of chapter 1. This allows for an assessment of whether or not the AI-visualization paradigm has been followed.

1. **Object-based Presentation:** Since clusters and not objects are the subject of this chapter, the object-oriented concept can not be violated.

3. **Abstraction:** Cluster classification is another step of abstraction: After a graph is abstracted into a system of clusters, each cluster is further abstracted into a single symbol, i.e. its label. Again, this approach is borrowed from human problem solving methods: As many author have stated before (e.g. see [109, 120]), human thinking often happens in terms of symbols. It therefore makes sense to associate clusters with labels.

   The abstraction power of these methods mainly depends on the underlying classification or similarity function (see appendix A for regression and neural networks and chapter 5 for the learning of similarity functions). All methods are also evaluated using real-world problems in chapters 6 and 7.

4. **User Adaption:** Cluster classification adapts the visualization methodology to the user by labeling identified clusters with meaningful symbols. Three ways of user-adaption have been introduced and assessed in this section:

   (a) **Manual Definition of a Classification Function**: This method demands the formulation of knowledge in the form of a mathematical function and is therefore not applicable in most cases.

   (b) **Learning of a Direct Classification Function:** In section 3.1, the cluster classification process is adapted to the user by applying Machine Learning: Exemplary labelings are analyzed and a classification function is abstracted. This classification method is used for the identification of optimal clustering methods and cluster labels in section 6.2. Users find the notion of automatically chosen layout methods to be interesting and promising (see e.g. [125]).

   This approach is almost optimal from a knowledge acquisition point of view; the user behaviour is observed and compiled into classification knowledge. The following comparison to case-based classification reveals some disadvantages.

   (c) **Case-based Classification:** Case-based classification has, compared to direct classification, interesting advantages: (*i*) New labels are

"learned" simply by remembering the corresponding cases[2]. The reader may note that by integrating new cases, the classification is adapted to the user. (*ii*) Sometimes finding (or learning) a similarity measure is easier than finding a function for the direct classification. E.g. only experts are able to classify wine, but most people can assess the similarity between two given wines. (*iii*) In some domains, e.g. in medical science, cases are a natural form of knowledge representation.

The following two methods for finding similarity measures differ by the explicity of the demanded knowledge:

i. **Manual Definition of a Similarity Measure:** Explicit knowledge about object similarities is needed for this method. An example is given in section 6.1.2.

ii. **Learning of a Similarity Measure:** Abstract and user-friendly methods for the definition of similarity measures are given in section 5.2.2. An examples is given in section 7.1.1.

These methods are also discussed for each domain separately in chapters 6 and 7.

[2] Assuming that the similarity measure is still correct.

Chapter 4

Structure Envisioning

In this chapter, methods are presented which compute a graph layout (see definition 1.4.2) for a given (clustered) visualization graph (see definition 1.4.1). First, existing algorithms for drawing graphs are described in section 4.1. New graph drawing methods are described in section 4.2. All algorithms are extended in section 4.3 to allow for the integration of cluster knowledge.

In this chapter, only theoretical evaluations of the presented methods are given. An evaluation using graphs from various realistic domains can be found in chapters 6 and 7.

Two categories are used to classify graph drawing techniques:

I. **Quality Criterion:** Some drawing methods use local criterions, e.g. edges, to define a good layout. Other techniques are based on global features of the graph, e.g. graph diameters. This criterion is called quality criterion.

II. **Graph Class:** While some graph drawing algorithms can be used with all graphs, other methods are mainly suited for special graph classes. Typical graph classes are “Directed Graphs”, “Weighted Graphs”, “Trees”, or “Bipartite Graphs”.

4.1 Existing Methods

Graph drawing is the area of computer science dealing with the problem of finding two- or three-dimensional[1] positions for the nodes of a graph. Normally, an esthetic graph layout is wanted. Though, the visualization method presented in this work emphasizes the data-mining aspect: Graphs are visualized to support the analysis of large and complex data. Hence, similarity definitions, clustering, and cluster classification have been used. This work is intended as a contribution to the field of visual data-mining, not mainly to the field of graph drawing.

Much research has been done recently in the relatively young area of graph drawing; good overviews can be found in [49, 8] or in the proceedings of the annual conference “Graph Drawing” (published by Springer). In the rest of this section, the most popular classes of graph drawing algorithms are presented.

[1] 3-dimensional drawings are not dealt with in this work.

This short introduction focuses on algorithms suited for general (directed) graphs. Methods developed for special graph classes (trees, acyclic graph) are mentioned in section 4.1.4.

4.1.1 Force-Directed Drawing

Criterion:	local: edge lengths
Class:	general graphs

Force-directed drawing methods rely on knowledge about optimal edge lengths. Wrong edge lengths are corrected by moving the corresponding nodes into a better position.

This is often illustrated using an analogy from the field of physics: Nodes correspond to small bodies, while edges correspond to springs connecting them. Starting from a random layout, the spring forces move the bodies into such positions that leave the springs as relaxed as possible. In order to simulate such a system on a computer, all forces affecting a node must be calculated and iteratively applied to the bodies.

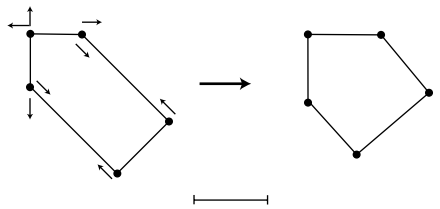

Figure 4.1: *One step of a force-directed layout, forces are depicted as arrows, the optimal edge length can be seen at the bottom of the figure.*

Figure 4.1 shows an example: On the left side a randomly laid out graph can be seen; edge lengths are suboptimal. The optimal edge length is shown at the bottom of the figure. Arrows depict the forces on the nodes; these forces move the nodes into the positions shown on the right side of the figure.

An alternative view of such approaches uses energy functions: The sum of deviations between optimal edge lengths and actual edge length is called the energy or the overall-error of the graph layout. Force-directed methods try to find a layout minimizing that energy. From this point of view, calculating forces on nodes and moving them respectively may be seen as a local search heuristics for finding a minimum of the energy function. The reader may note that by using forces, only a local minimum can be found.

Several realizations of this algorithm exist, two typical implementations are presented in more detail:

- **Eades:** P. Eeades introduced the idea of force-directed drawing to the field of graph-drawing ([33]) and named his approach spring-embedder. Two connected nodes v_1, v_2 placed at a distance of $dist_{v_1,v_2}$ are attracted with a force proportional to $\log(\frac{dist_{v_1,v_2}}{\delta_{v_1,v_2}})$ where δ_{v_1,v_2} denotes the optimal distance between v_1 and v_2. This force obviously attracts the nodes only if $dist_{v_1,v_2} > \delta_{v_1,v_2}$, otherwise is repels them. Not connected nodes are repelled by a force proportional to $\frac{1}{\delta^2_{v_1,v_2}}$. Eades moves iteratively one node after the other until an equilibrium as achieved.

- **Kamada and Kawai**: This approach has been developed in [83] and, independently, in [101]. The optical distance between nodes is defined by their graph-theoretical distance[2] in the graph. Applying the energy view onto the graph layout, Kamada and Kawai assume the energy in the edge between v_1 and v_2 to be $k \cdot \frac{(dist_{v_1,v_2} - \delta_{v_1,v_2})^2}{\delta^2_{v_1,v_2}}$, where $dist$ again denotes the actual (Euclidean) distance and δ the optimal distance.

  The solution is found iteratively, Kamada and Kawai always move that nodes first, whose movement results in the highest improvement of the energy function. The direction of the movement is defined by the local gradient of the energy function.

Other variations have been presented e.g. by Fruchtermann and Reingold in [47], who used different forces to improve the run-time behaviour. Sugiyama and Misue introduced in [167, 168] an algorithm which incorporated new constraints by adding magnetic forces. Tunkelang applied in [175] numerical analytic methods to improve the optimization process, another method has been proposed in [128]. Some author, e.g. in [19, 29], have used different optimization method. A good overview of force-directed drawing can be found in [8].

In this work, two versions of force-directed layout algorithms have been used: Tunkelang's method[3] and Eades's spring-embedder.

Force-directed method offer many advantages:

⊕ It is intuitively appealing.

⊕ The results often resemble the human esthetic notion of nice graph drawings.

It also has disadvantages:

⊖ The run-time behaviour allows only for the visualization of small graphs: In every step $O(n^2)$ pairs of nodes have to be evaluated. Since for most graphs at least $O(n)$ steps are necessary for a good layout, a complexity of $\Omega(n^3)$ results. Tunkelang's methods needs only $\Theta(|E| \cdot |V| + |V|^2 \log |V|)$ time (assuming $|V|$ iterations). Clustering provides a way to reduce the complexity, e.g. in section 2 or in [57, 61].

⊖ Edge crossings and other constraints are not taken into consideration in the basic version of the algorithm; the only optimization criterion is the given optimal edge length. These problems have been the subject of several improvements to the basic method, e.g. in [167, 35, 19].

Two examples can be seen in figure 4.2, a protein-network on the left (section 6.3) and a knowledge-base for the configuration of telephone systems (see section 6.2).

[2] see definition B.0.4
[3] Implemented in [178].

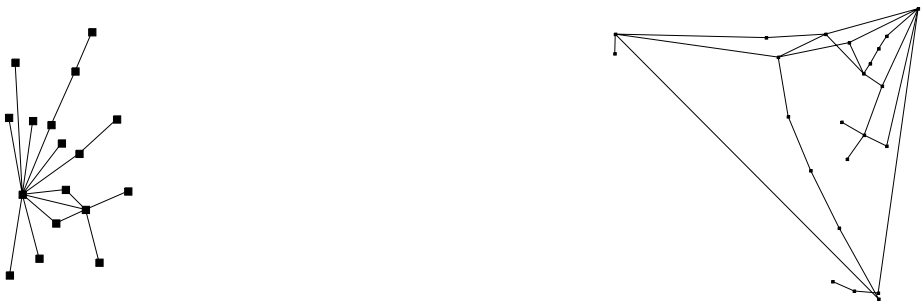

Figure 4.2: *A Protein-network (left side) and a knowledge-base of a telephone system (right side) visualized by a spring-embedder*

4.1.2 Multidimensional Scaling

Criterion:	local: node distances
Class:	general graphs

A similar or even identical graph visualization technique to force-directed drawing (section 4.1.1) is multidimensional scaling. This approach stems from the field of statistics and descriptive data analysis (see e.g. [172]). Good overviews can be found in [17, 6]. A good comparison to force-directed drawing and to other graph layout methods is given in [30].

Generally speaking, Multidimensional Scaling defines energy functions similar to force-directed drawing (see section 4.1.1) and applies gradient methods to find a local minimum. In addition to the different history, the main differences to force-directed drawing are (*i*) modified energy functions, (*ii*) the usage of a totally connected graph (the force-directed layout method due to Kamada and Kawai [83] can be seen as a multidimensional scaling algorithm) and (*iii*) different optimization algorithms.

4.1.3 Level-Oriented Drawing

Criterion:	global: graph inherent hierarchy
Class:	directed graphs with few cycles

Sugiyama, Tagawa and Toda introduced in [81] a new layout approach comprising three steps. These steps are also shown in figure 4.3:

1. **Layer Assignment:** Nodes are assigned to horizontal layers.

2. **Crossing Reduction:** Nodes within a layer are ordered to minimize the number of edge crossings.

3. **Horizontal Coordinate Assignment:** Nodes are assigned x-coordinates.

This method normally assumes directed graphs. The first step associates nodes with (horizontal) layers in such a way that nodes within a layer are not connected. For acyclic graphs, this can be done by a topological sorting of the nodes. Cyclic graphs are transformed into acyclic graphs (see [135, 8]) by reversing the direction of some edges. For complexity reasons (the so-called feedback arc set problem is NP-complete, see [51]), heuristics are applied.

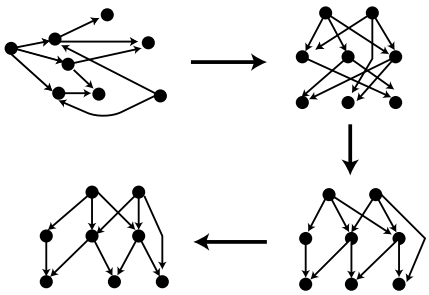

Figure 4.3: *The three steps of the level-oriented layout approach.*

If connected nodes are not in neighboured layers, edges have to cross several layers. To improve the optical appearance of such edges, so-called dummy nodes are inserted. E.g. for an edge between a node v_1 in layer 1 and a node v_2 in layer 3, a dummy node d in layer 2 is created. The original edge $v_1 \rightarrow v_2$ is replaced by the edges $v_1 \rightarrow d \rightarrow v_2$. This can also be seen in figure 4.3.

Within the second step, an ordering of the nodes is found that minimizes the edges crossings between layers. Since this problem is also NP-complete, heuristics (e.g. barycenter method, see [81]) are used. An overview of existing heuristics can be found in [8].

In a third step a balanced layout is found. For this, positions for the nodes are found without changing the ordering from step 2. The balanced layout is found by minimizing the distances between connected nodes. Details can be found in [50].

Several improvements to this algorithm exist, e.g. the inclusion of compound or cluster information (see [148, 166, 147, 39]). A general overview can be found in [147, 8].

The main advantages are:

⊕ Graphs which comprise a general flow, i.e. a general direction of the edges, can be drawn nicely.

⊕ Graphs which model a given hierarchy, e.g. the positions in a company, are drawn in a natural way.

⊕ Bipartite graphs (for a definition refer [80]) are laid out nicely, since each layer can only comprise nodes of the same type.

Disadvantages are:

⊖ The method is not well suited to undirected graphs. Of course, undirected graphs may be transformed into directed graphs, but randomly chosen edge directions do not carry any real information.

⊖ Graphs which do not have a hierarchical structure are drawn rather poorly.

An example of a level-oriented drawing can be seen on the left side of figure 4.4.

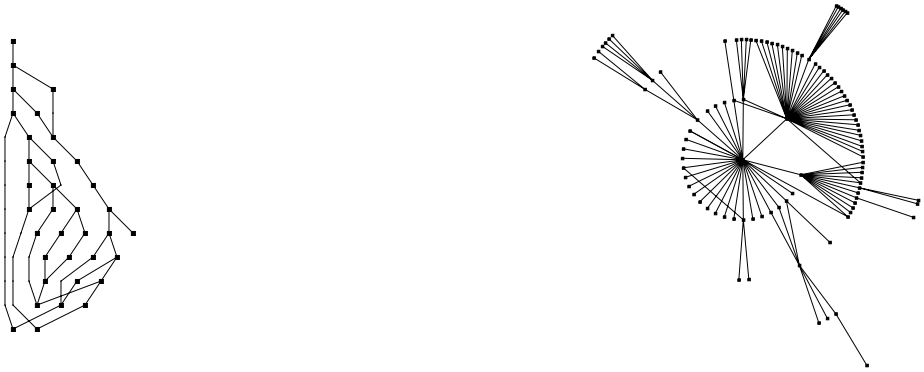

Figure 4.4: *A fluidic circuit visualized by a level-oriented algorithm (left side) and network traffic visualized by a circular layout algorithm (right side).*

4.1.4 Other Methods

Several other methods for laying out a graph exist. A classification can be seen in table 4.1.

Name	Criterion	Class
Circular Layout	global: minimum spanning tree	all
Flow-based Drawing	local: edges	planar graphs
Self-Organizing	local: edge lengths	all

Table 4.1: Classification of some Graph Drawing Methods

Tree Layout algorithms are specialized in laying out trees; examples can be found in [139, 169]. A popular approach is the circular layout ([34, 12]) : The root of the tree is placed in the center of the plane and all descendants are positioned on concentric circle around the root node.

The reader may note that tree drawing algorithms can be applied to all graphs by computing the minimum spanning tree first and using only edges for the visualization process included in the spanning tree. The root of the spanning tree is chosen in such a way that the height of the tree is minimized, the reader may refer to [115] for details. An example can be seen in Figure 4.4 (right side).

Flow-based drawings form a popular class of layout algorithms for planar graphs. The drawings are orthogonal, i.e. edges are drawn rectangular and edge bends are minimized. An overview can be found in [8]. The reader may note that though most algorithms compute global features of the graph (e.g. minimum cost flow), the criterion itself relies mainly on local graph features. Orthogonal drawing algorithms exist for general graphs as well: Some author planarize a graph and apply layout techniques for planar graphs afterwards (e.g. in [76, 129, 8]), while others construct a orthogonal drawing directly (e.g. in [131, 8]).

Self-organizing layouts using Kohonen maps have been introduced in [110].

Several authors have combined clustering knowledge and graph drawing: In [32], a method for clustering already laid out graphs is presented. This method subdivides the graph into almost equally sized partitions. A classical optimization approach to clustering has been combined with a spring-

embedder in [145]. In [143], a divisive clustering technique has been used which also tries to generate equally sized clusters. A force-directed layout method has been applied to visualize the resulting clusters.

Several papers exists which introduce layout techniques for already clustered graphs: In [179, 70, 107] clustered graphs are visualized by means of force-directed layout algorithms. 3-dimensional drawing of clustered graphs are described in [36]. Orthogonal drawings of clustered graphs are dealt with in [37]. In [166, 148], level-oriented layout algorithms are extended to allow for an integration of clustering knowledge.

4.2 Factor-Based Drawing

Criterion:	global: variances
Class:	all

Most graph drawing methods presented in section 4.1 can be ranked into two types:

- Some methods use local information to find a layout (e.g. force-directed drawing, self-organizing layout, etc.). Since they disregard global information about the graph, the resulting layout is often locally pleasing but fails to emphasize the global graph structure.

- Other algorithms use a global quality criterion which exploits a-priori knowledge about the graph's structure (e.g. level-oriented layout, circular layout, etc.). Of course, these algorithms can only be applied to specific graph classes.

This section introduces a new method which is applicable to general graphs and which uses a global optimization criterion.

In the field of statistics, principle component analysis is a popular data analysis technique[4] (e.g. see [11, 6, 105, 141]). Given a tabular data set G with n features, principle component analysis reduces G to a new data set G' with only $m, m < n$ features, whereby as much information as possible is preserved.

So far, while being a popular method for visualizing tabular data sets in the area of statistics, principle component analysis has not been applied to graph drawing. The main idea presented here is as follows: The adjacency matrix of a given graph $G = (V, E_c)$ is interpreted as a tabular data set. Each row is the description of a node, i.e. nodes are described by their edges. This $|V|$-dimensional tabular data set is reduced to a 2-dimensional data set using principle component analysis. The new 2-dimensional features of a node are used as visualization coordinates.

The basic idea of principle component analysis is now described using an example: Figure 4.5 shows a 2-dimensional data set on the left side. This data set is transfered into a new 2-dimensional data set. First of all, an ellipse is placed around the nodes. The first new dimension runs from one tip of the ellipse to the other (i.e. the first axis of the ellipse). This axis is depicted in the figure as a solid line. The second new dimension (see the dotted line in the

[4] Some authors classify it as a special factor analysis method.

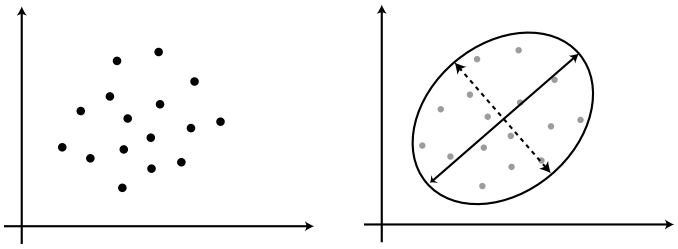

Figure 4.5: *The identification of the new dimensions by principle component analysis*

figure) is defined by the second largest distance within the ellipse and must be orthogonal to the first dimension (i.e. the second axis of the ellipse). The data is now transfered into the new coordinate system. Obviously, global features of the graph are used for this layout technique.

More formally, the algorithm first finds a new x-axis in such a way that the variance of this new feature is maximized. In the second step, the second feature (y-value) is computed, using the same method as before, from that part of the original matrix which has not been explained by the first axis.

Since principle component analysis has not been described so far in the context of graph drawing, a formal description of a graph drawing by means of principle component analysis will be given now:

Algorithm 4.2.1 (Factor-based Layout)
Input: *A visualization graph* $G = (V, E_c, E_v, \delta, w, \Delta)$
Output: *A function* $\rho : V \to \mathbf{N} \times \mathbf{N}$ *defining two-dimensional positions*

1. *The* $|V| \times |V|$ *adjacency matrix* A *for the graph* G *is constructed as follows: Let* a_{ij} *denote the element of* A *in the* i*th row and* j*th column.* a_{ij} *is set to* $w(\{v_i, v_j\})$*. We presume that* A *is standardised: The average value of each column of* A *is* 0 *and the variances are* 1*.*

2. *The correlation matrix* C *of* A *is constructed as follows: Let* c_{ij} *denote the element of* C *in the* i*th row and* j*th column.* c_{ij} *is the (linear) correlation between the* i*th and the* j*th column of* A*. The reader may note that only linear correlations between columns are used.* c_{ij} *is computed as follows:*

$$c_{ij} = \frac{\sum_k (a_{ki} - \bar{a_{\cdot i}})(a_{kj} - \bar{a_{\cdot j}})}{\sqrt{\sum_k (a_{ki} - \bar{a_{\cdot i}})^2 \cdot \sum_k (a_{kj} - \bar{a_{\cdot j}})^2}},$$

   where $\bar{a_{\cdot i}}$ *denotes the average value of the* i*th column. Since* $\bar{a_{\cdot i}}$ *is always* 0 *and the variances* $\frac{\sum_k (a_{ki} - \bar{a_{\cdot i}})^2}{|V|-1}$ *are* 1*,* C *can also be computed by* $C = \frac{1}{|V|-1} A \cdot A^T$*.*

3. *Let* λ_1 *and* λ_2 *be the first two eigenvalues*[5] *of* C *and let* x_1, x_2 *be the corresponding eigenvectors. The first new axis* n_1 *is then defined by* $n_1 = \sqrt{\lambda_1} x_1$ *and the second axis* n_2 *by* $n_2 = \sqrt{\lambda_2} x_2$*.*

4. *The only problem left is the calculation of the new coordinates for the objects according to the 2-dimensional coordinate system* $\{n_1, n_2\}$*. In [82]*

[5] For an introduction to eigenvalues and eigenvectors, the reader may refer to [100, 181].

a fast method which, unlike many other methods, does not rely on all eigenvectors has been introduced: The matrix $R = (A^T \cdot A)^{-1} \cdot C^T \cdot A$ *defines the new coordinates of the objects and thus the graph layout* ρ*.*

In the implementation used for the STRUCTUREMINER system, the power method (see [100, 181, 68]) has been applied to compute the first two eigenvalues and eigenvectors. This approximative algorithm identifies one eigenvalue / vector pair after the other. Since only the first two eigenvalues / vectors are needed in the last step of the algorithm, the usage of such algorithms is recommended.

Figure 4.6: *A grid laid out by factor-based drawing (left side) and by all-pair factor-based drawing.*

One typical problem of factor-based layout is shown on the left side of figure 4.6: The layout of the 15×10 grid seems to be crumpled. This is mainly because the original matrix A only comprises local information. No global information keeps the algorithm from placing two unneighboured nodes too close to each other. Factor-based drawing now tries to find a compact layout, thus leading to a "crumbled" layout. Most humans want a different layout, i.e. they want the original distances between nodes in the graph to be preserved. This can be carried out by computing the matrix A in the first step of the algorithm 4.2.1 differently: a_{ij} is set to $dist(v_1, v_2)$ ($dist$ is defined in definition B.0.4). The result can be seen on the right side of figure 4.6. This modified version of algorithm 4.2.1 is called all-pair factor-based drawing.

The main advantages of this method are:

- ⊕ No assumptions about the graphs are made.
- ⊕ Since global features are used, the layout emphasises the global structure of the graphs very well. Because of the maximization of feature variances, distance proportions in the graph are emphasised.

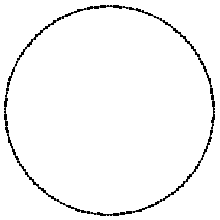

Figure 4.7: *A cycle visualized using all-pair factor-drawing*

This is a major advantage compared to force-directed layout methods: Force-directed drawing uses only local information to find an optimum

layout, i.e. it uses edges and edge lengths. This works well for symmetric graphs (e.g. grids) since the global structure of the graph is mirrored in its local structure. Most graphs from realistic domains as used in chapters 6 and 7 are non-symmetric.

A simple example can be seen in figure 4.7: A cycle with 200 nodes is visualized using all-pair factor-drawing. This method always finds the optimum layout.

Force-directed methods have problems laying out cycles, because the local structure is different from the global structure. Tunkelang's version of force-directed drawing only finds in 20 of 50 tries a planar layout; in most cases extra loops were generated (see left side of figure 4.8). Such loops correspond to local minima of the error function optimized by force-directed layout (see also section 4.1.1). Even if a planar layout is found, the layout is suboptimal; an example can be see on the right side of figure 4.8.

Figure 4.8: *A cycle visualized using Tunkelang's version of force-directed layout*

⊕ The computed solution is always optimal with regard to the explained variances.

Some disadvantages are:

⊖ It is not very fast: Step 1 of algorithm 4.2.1 needs $O(|E|)$ time and step 2 and step 4 have a complexity of $O(|V|^3)$. The power-method used in this implementation for the computation of eigenvalues and eigenvectors converges with the same rate as $(\frac{\lambda_2}{\lambda_1})^s \to 0$, where s denotes the number of iterations (see [181]). The reader may note that this method is at least as fast as typical force-directed approaches.

The following theorem ranges the run-time behaviour in:

Theorem 4.2.1
If the layout criterion "Node distances in the final layout should resemble the original node distances in the graph" is reasonable, then any satisfying graph layout technique must have at least the same run-time complexity as the all-pair-shortest-pathes problem[6]*.*

Proof.
Since these distances are initially unknown, any graph drawing method must compute them. Algorithms which rely on local information only (e.g. force-directed methods) can not abide by the criterion from above. Algorithms as the level-oriented methods rely on special knowledge about node distances.

[6]The well-known Floyd-Warshall algorithm needs $O(|V|^3)$ time.

- Edges have no uniform lengths and edge crossings are not minimized.
- Since rows do not need to be different, nodes may be placed at the same position.

The problems of equally placed nodes and of non uniform edge length can be overcome by using an incremental spring-embedder (see section 4.1.1) as a post-processing step: This special spring-embedder uses the node positions computed by the factor-based layout algorithm as starting positions. Just like a normal spring-embedder, nodes are moved according to attraction and repulsion forces. Unlike a normal spring-embedder, the amount of node movement is restricted, i.e. nodes are only allowed small position changes. Furthermore, the amount of movement allowed decreases over time. Such an algorithm results in small local position changes while the overall layout remains unaltered; hence combining the advantages of spring-embedder (local optimization criterion) and factor-based drawing (global optimization criterion). Figure 4.9 gives an example. The reader may note that (*i*) the drawing on the left side (pure factor-based layout) shows precisely the tube's two most important dimensions and (*ii*) the drawing on the right side (factor-based layout combined with an incremental spring-embedder) is a 2-dimensional drawing. No 3-dimensional information has been used.

Figure 4.9: *An all-pair factor-based layout of a 3-dimensional tube (left side), obviously nodes have been placed very closely together. The drawing on the right side shows the same graph after an incremental spring-embedder has been applied as a post-processing step.*

4.3 Combining Clustering and Graph Drawing

The cluster information from section 2 still has to be integrated into the graph drawing process in order to implement the methodology of structure-based visualization from section 1.3. The following detailed description of the structure-based visualization process illustrates the exploitation of the computed struc-

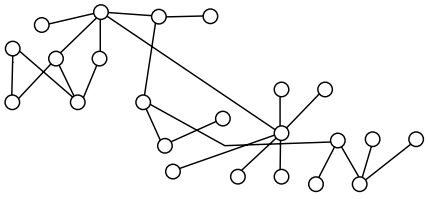

Figure 4.10: Example Step 1

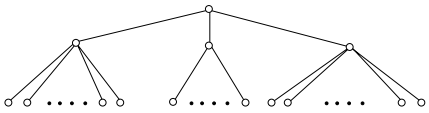

Figure 4.11: Example Step 2

ture information for the graph layout process. Later on, necessary extensions to the layout methods are explained.

1. **Structure Identification:** As described in chapter 2, the graph's inherent structure is identified, resulting in a (hierarchically) clustered graph. This is illustrated using a small exemplary graph: The graph in figure 4.10 depicts a small traffic graph. This graph is clustered (e.g. using MAJORCLUST, see section 2.2.1 for details). The result can be seen in figure 4.11, i.e. the clustering tree from definition 2.0.5 is depicted. Three clusters have been identified.

2. **Structure Classification:** The cluster are classified, i.e. each cluster is described by a label. Figure 4.13 shows the classified clusters.

3. **Structure Envisioning:** The graph is laid out with respect to the structure information. For this, three steps are necessary:

   (a) *Size Estimation:* For each cluster a bounding box is estimated. Clusters are laid out within their bounding boxes. The size of the bounding box is estimated from the bottom up: For each node, the size is given. This size may be seen as the bounding box of the node.
   Clusters comprising only clusters or nodes, whose bounding boxes are already known, compute their bounding box size by merging the individual bounding boxes of their subclusters. Normally, the resulting box size is multiplied by a constant value to allow for an uncongested layout and to support the layout process. This step can be seen in figure 4.14.

   (b) *Layout:* The clustered graph is laid out from the top down: The graph comprising the top clusters (i.e. the graph's induced structure, see definition 2.0.4) is laid out using a given graph drawing method. The bounding boxes of the clusters are the nodes of this graph. Laid out bounding boxes (i.e. clusters) can be seen in figure 4.14.

   (c) *Substructure Layout:* In the next step, the substructures of a cluster are laid out. For this, the substructure becomes the new input for the layout algorithm and the process recursively goes to step 3b. In our example, the substructures consist of the original nodes, the laid out substructures of the three top clusters are shown in figure 4.15.
   One problem remains, namely that substructures can not be totally independently laid out. If two connected nodes belong to different clusters, they should nevertheless be placed together as close as possible. I.e. if a substructure is laid out, edges to other substructures have to be considered. An example can be seen in figure 4.12 where the nodes A and B are in different clusters. A should be placed on the right side of the left cluster and B on the left side of the right cluster.

   (d) *Post-Optimization:* In some cases, a computed graph layout does not fit into the given bounding box. In these cases, two post-optimizations are possible: (*i*) The bounding box uses free space in its neighbourhood for enlargement. (*ii*) If no free space is available, the graph layout is scaled to fit into the bounding box.

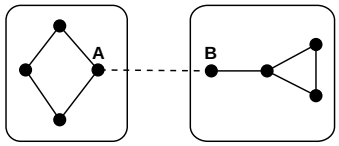

Figure 4.12: *The dependency between substructures*

In order to implement the methodology described above, the graph layout methods have to be extended in three aspects:

1. **Bounding Box:** Since each layout happens within the bounding box of the respective parent cluster, the layout algorithms have to incorporate the possibility to define maximum and minimum x- and y-coordinates.
2. **Integration of external Attractions:** As described in step 3c, some nodes are connected to nodes in other clusters. These connections result in an attraction to other clusters. In order to simplify this problem, only four external attraction are allows for a node: to the left, to the right, to the top, and to bottom of its bounding box.
3. **Spatial nodes:** Nodes are now defined by a bounding box, i.e. they are not dimensionless quantities anymore, but geometric entities. The graph drawing algorithms have to be able to work with such nodes.

These extensions have been implemented for STRUCTUREMINER as follows:

- **Level-oriented Layout:**
  1. **Bounding Box:** Only the methods from step 3d are used.
  2. **Integration of external Attractions:** Attractions to clusters above or below the current cluster can be implemented by assigning a node to a top or bottom level. Horizontal attractions are taken into consideration by moving a node to the left or right of its level.
  3. **Spatial nodes:** Spatial nodes can be integrated by using appropriate distances between layers (each layer has the height of its highest node) and by leaving enough space between nodes within a layer.
- **Force-directed Layout:**
  1. **Bounding Box:** Additionally to the methods from step 3d, nodes which leave the bounding box are placed in the next step randomly within the bounding box again.
  2. **Integration of external Attractions:** Attractions to nodes within other clusters are implemented by additional attraction forces.

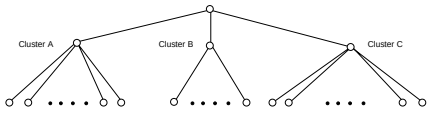

Figure 4.13: Example Step 3

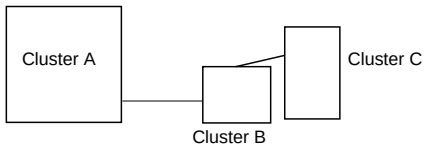

Figure 4.14: Example Step 4

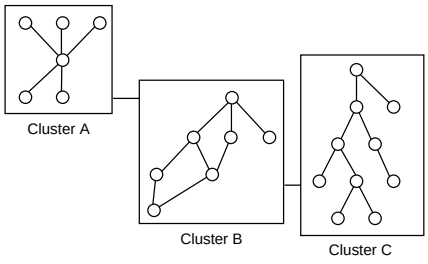

Figure 4.15: Example Step 5

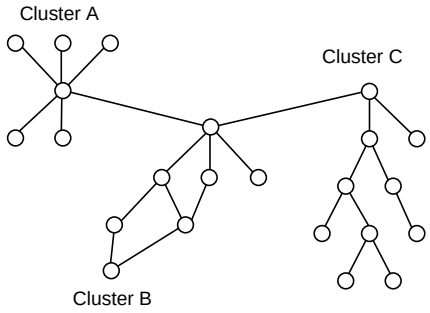

Figure 4.16: Example Step 6

3. **Spatial nodes:** Spatial nodes can be implemented by modifying the optimal edge length: Bounding box diameters are added to the edge lengths.

- **Circular Layout:**

1. **Bounding Box:** Only the methods from step 3d are used.

2. **Integration of external Attractions:** External attractions can be integrated into the circular layout. The general idea is as follows: Each node in the tree summarizes its own attractions and the attractions of all its descendants. Since attractions can be seen as vectors (e.g. $(1, 0)^T$ for an attraction to the right), summarizing attraction means summarizing vectors. The root node is always placed in the center of the plane, its descendants are positioned on the inner circle according to their summarized attractions. Nodes deeper in the tree are also placed according to their attractions. Since they also should be placed close to their parent node, a trade-off between external attractions and attraction to their parent node has to be found.

3. **Spatial nodes:** This can be implemented by choosing appropriate diameters for the concentric circles.

- **Factor-based Layout:**

1. **Bounding Box:** Only the methods from step 3d are used.

2. **Integration of external Attractions and Spatial nodes:** This demands are met by applying a force-directed algorithms as a post-processing step. The force-directed algorithm simply uses the result of the factor-based layout as its initial starting point. Details can also be found in section 4.2.

4.4 Evaluation

As outlined in section 1.7, the methods described in this chapter are rated by the four features introduced at beginning of chapter 1. This allows for an assessment of whether or not the AI-visualization paradigm has been followed.

1. **Object-based Presentation:** All layout techniques presented in this chapter depict objects as clearly recognizable geometric entities.

2. **Constant Dimension Presentation:** All drawing methods reduce a graph to a 2-dimensional layout. Only the techniques of force-directed drawing, multidimensional scaling, and factor-based drawing are suited for general graphs. The new technique of factor-based drawing especially supports a fast and deterministic drawing of graphs. Other layout methods, e.g. level-oriented drawing, orthogonal drawing, or tree drawing have been developed mainly for special graph classes.

   Common weaknesses of all algorithms, especially run-time problems and an insufficient emphasis of the graph's structure, have been partially overcome by the combination with clustering methods. For this, extensions of all drawing methods have been described in section 4.3.

   Comparing layout methods leads to the same problems as the comparison of clustering algorithms (see section 2.4): No generally accepted quality measure exists. Thus, only two ways remain to carry out such a comparison: (*i*) Theoretical features can be compared which has been done in this chapter. (*ii*) The layout methods can been evaluated using real-world problems: chapters 6 and 7 describe six applications.

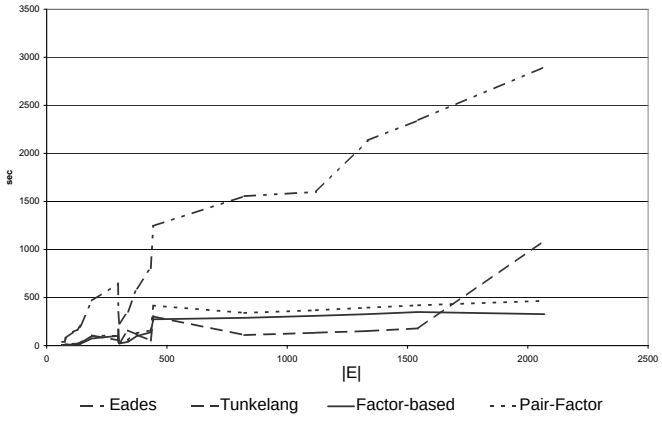

Figure 4.17: *A run-time comparison of different layout methods.*

The same method as in section 2.4 is used here to compare the run-time behaviour[7] of layout algorithms: 52 typical graphs from all six domains are visualized and rated. Figure 4.17 depicts the run-time of acceptable layouts over the number of edges. The reader may note that (*i*) As expected, Eade's force-directed layout algorithm is much slower than Tunkelang's. (*ii*) The run-time of the factor-based algorithms is less influenced by the number of edges than the other methods. (*iii*) Circular and

[7] The tests have been carried out on a Pentium II, 400Mhz using CLisp.

level-oriented layout methods are not shown since their run-time never exceeds one minute.

The layout quality is compared for each domain separately in chapters 6 and 7.

3. **Abstraction:** In most cases, a 2-dimensional layout of a graph comprises less information than the original graph. This is mainly because the Euclidean distances on the plane differ from the edge weights and the layout now defines distances between previously not connected nodes. The loss of information can be seen clearly when layout techniques such as multidimensional scaling or factor-based drawing are used. Both methods use an explicit loss function (the optimization function for multidimensional scaling or the percentage of explained variances for factor-based drawing).

   Loss of information is connected to data abstraction: Important information is emphazised while less important information is lost. In this sense, graph layout may be seen as an abstraction technique. The general information about node relations is preserved while the precise information about edge weights is lost. Factor-based drawing controls the loss of information by preserving as much variance as possible. It therefore provides a theoretically well-founded means of data abstraction. Force-directed methods only use local information, therefore disregard the loss of global information.

4. **User Adaption:** User adaption in the field of graph drawing mainly means the choice of the proper drawing method. For some graphs, a level-oriented technique might be optimal while other graphs or subgraphs can best be laid out by a force-directed method or even by a specially developed algorithm. Section 3.1.1 elaborated on a method for the automatic choice between drawing methods.

   The classification of drawing methods given in this work may also help in manually choosing the correct layout algorithm.

Chapter 5

Creating Graphs from Tabular Data

In this chapter, methods are presented which take a tabular data set (see definition 1.5.1) and compute a corresponding visualization graph (see definition 1.4.1). First, general concepts for defining object similarities are described. Existing algorithms for learning similarity functions are the subject of section 5.2.1. Section 5.2.2 introduces new methods for learning object similarities. These methods use abstract user interfaces and machine learning to bridge the gap between implicit expert knowledge and explicit knowledge as necessary for the usage by a computer system.

Data mining or knowledge discovery tries to find coherencies and unknown relationships between objects in large sets of data. In this sense, the graph analysis techniques introduced and described in this work should be looked at as data mining methods. Nevertheless, in many cases, the data is not given as a graph, but as a tabular data set (see definition 1.5.1).

For many data mining tasks, the general goal of the analysis is known, e.g. when investigating the influence of people's youth on their later lives. More precisely, users normally have an object similarity category (people's youth) and an investigation goal (an assessment of people's later life, e.g. their income). The underlying visualization assumption is that the object similarity category influences the investigation goal.

The object similarity category defines the similarity between objects, e.g. people with similar youths are similar. The visualization should express this similarity between objects by means of spatial relations in the visualization, e.g. people with similar education, citizenship, and gender should be placed closely to each other, since their youths were probably similar.

Using such a visualization, the user is often able to recognize structures in the data, i.e. if the visualization assumption holds, spatial patterns of objects should correlate with the investigation goal. The reader may note that the task of identifying *unknown* patterns in the visualization is left the the user. Thus, the distinctive human ability of visual pattern recognition is exploited. This general methodology can also be seen in figure 5.1.

Many existing visualization and data analysis systems do not support the definition of object similarities. Instead, given object features are directly in-

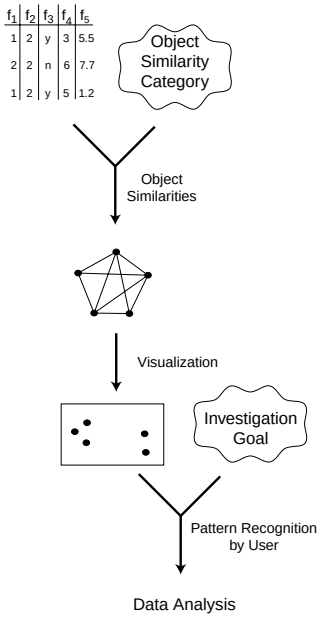

Figure 5.1: *The general methodology for the visual analysis of tabular data: Using a given object similarity category, object similarities are found and a graph is constructed. This graph is visualized. The user can now try to find coherencies between the visualization result and the investigation goal.*

terpreted as coordinates in an Euclidean space[1]. Thus, object similarities are defined implicitly by the corresponding Euclidean distances. Such techniques must not be understood as different from the method described above; the responsibility for the definition of object similarities is shifted only to a data preprocessing step. The given Euclidean distances implicitly define object similarity categories.

Therefore, no visualization technique for tabular data is possible where the user has not defined object similarities. Either he scales the data in such a way that the resulting features may be viewed as Euclidean coordinates (implicitly defining object distances), or he specifies the object similarities explicitly. Euclidean similarity definitions are highly sensible to feature scaling, e.g. choosing meters or feet as a unit for height influences the object distances. For these reasons, this sections deals with the direct definition of object similarities by means of functions.

Humans normally find it hard to define object similarities explicitly in the form of a mathematical function for most datasets and analysis tasks. In simple cases, e.g. when the similarity between people equals the difference between their incomes, a similarity function can be stated easily. But for most cases, the process of transferring a fuzzy understanding of “similar” into an explicit mathematical function overtaxes the capabilities of most users, hence making the application of graph-based analysis and visualization methods described in this work impossible. Therefore, this section introduces knowledge acquisition methods for learning a similarity measure which are based on a combination of high-level user interfaces and Machine Learning.

[1]For this, non metric feature are transformed into metric feature.

Similarity functions are now defined formally:

Definition 5.0.1
Let $G = (V, E_c = \emptyset, E_v, \delta, w, \Delta)$ be a tabular data set. As defined in definition 1.5.1, $\delta(v_i) = (f_1^{(i)}, \ldots, f_p^{(i)}), v_i \in V, p \in \mathbf{N}$ denotes an object feature list, and $f_j^{(i)} \in (\Delta \setminus \{,(,)\})^*$ denotes the feature j of node v_i. A function $sim : \Delta^* \times \Delta^* \to \mathbf{R}$ is called a similarity function for G if and only if[2] $sim(\delta(v_1), \delta(v_2)) = sim(\delta(v_2), \delta(v_1))$.

Sometimes similarity functions are defined more restrictively (e.g. in [41, 75]). Such functions will be called strict similarity functions:

Definition 5.0.2
Let $G = (V, E_c = \emptyset, E_v, \delta, w, \Delta)$ be a tabular data set and let sim be a similarity function for G. sim is called a strict similarity function if and only if $sim(\delta(v_1), \delta(v_2)) = \max\{sim(\delta(v_3), \delta(v_4)) \mid v_3, v_4 \in V\} \Rightarrow \delta(v_1) = \delta(v_2),\ v_1, v_2 \in V$.

For reasonable similarity functions, the following holds: Let $G = (V, E_c = \emptyset, E_v, \delta, w, \Delta)$ be a tabular data set, let $v_1, v_2, v_3, v_4 \in V$ and let sim be a similarity function for G. Then the statements "$sim(\delta(v_1), \delta(v_2)) > sim(\delta(v_3), \delta(v_4))$" and "$v_1$ is more similar to v_2 than v_3 to v_4" are equivalent.

> **Parenthesis:** Sometimes not the similarity but the dissimilarity between two objects is defined. Such a function is called dissimilarity-function. If for a dissimilarity-function $dist$ the property $dist(a, c) \leq dist(a, b) + dist(b, c)$ holds for all objects a, b, c, it is called a distance function. Dissimilarity functions and similarity functions can be transformed into each other (e.g. by the operations $-x$ or $\frac{1}{x}$). Details can be found in [41].

When the similarity function is known, the corresponding visualization graph can be constructed: For this, all nodes are connected by an edge and each edge is weighted by the objects similarities.

Let $G = (V, E_c = \emptyset, E_v, \delta, w, \Delta)$ be a tabular data set and let sim be a similarity function for G. A corresponding visualization graph $G' = (V', E_c', E_v', \delta', w', \Delta')$ can be constructed as follows:

- $V' = V$
- $E_c = \{\ \{v_1, v_2\} \mid v_1, v_2 \in V'\}$
- $E_v' = E_v$
- $\delta' = delta$
- $w' = sim$
- $\Delta' = \Delta$

[2] For asymmetric similarity functions, i.e. functions where $sim(\delta(v_1), \delta(v_2)) \neq sim(\delta(v_2), \delta(v_1))$, see [26]

This graph can now be visualized using the method of structure-based visualization (see section 1.3).

In the rest of this section, the problem of defining a similarity function is addressed. Since, as mentioned before, a manual definition is in most cases not possible, knowledge acquisition methods are used to make the user's implicit understanding of the domain explicit. For this, machine learning is used to narrow the gap between abstract and fuzzy expert knowledge and explicit and precise similarity functions.

5.1 Similarity Functions

Much work has been done in the last decades on similarity functions; good overviews can be found in [138, 41, 75, 180]. This section will introduce some basic concepts from similarity functions and will concentrate on one class of similarity functions which is suited especially for the applications presented in section 7: weighted similarity functions. The next section will focus on the automatic learning of these similarity functions.

In section 5, tabular data sets have been defined formally; each node (or object) v has been described by a set of features $\delta(v) = (f_1, \ldots, f_p)$. Nodes correspond to rows and features correspond to columns in a typical tabular representation of tabular data sets (see table 1.1). Similarity functions measure the similarity between the features of two nodes. These functions are often classified according to the type of features they can handle:

- **Cardinal:** We call a features cardinal if and only if all values of the feature are real numbers. Typical cardinal features are height, temperature, or distances. Values of cardinal features can be added, subtracted, multiplied and divided and the result is still a reasonable value for the feature.
- **Ordinal:** The only operation on values of ordinal features are the functions $<, =, >$. School-grades or positions in a company hierarchy are typical examples. Adding or subtracted values of ordinal feature does not make sense.
- **Nominal:** Nominal values can only be compared for equality. Name or profession of a person are nominal features. If a nominal feature has only two possible values (e.g. gender of a person), it is called a **binary** feature.

Nominal Features

If all features of two given nodes are binary, only four combination of possible value comparisons exist: $0/0, 0/1, 1/0$, and $1/1$. The number of value pair occurrences are summarized, a denotes the number of $1/1$ combinations, b the number of $1/0$ combinations, c the number of $0/1$ combinations, and d the number of $0/0$ combinations (see also table 5.1).

Several authors have combined the values a, b, c and e and formed similarity functions, e.g. the Jaccard coefficient $\frac{a}{a+b+c}$ or the matching coefficient $\frac{a+d}{a+b+c+d}$. Details and further similarity functions for binary features can be found in [155, 25, 55].

Nominal features with more than two values can be reduced to a set of new binary features: Let f be a nominal features with possible values $\{a_1, \ldots, a_k\}$.

values	1	0	
1	a	c	$a+c$
0	b	d	$b+d$
	$a+b$	$c+d$	

Table 5.1: Possible combinations for binary feature values

Then k new features $f_{a_1}, \ldots, f_{a_k}$ are created and the following relation between f and f_{a_i} exists: $\forall 1 \leq i \leq k$: $f_{a_i} = 1 \Leftrightarrow f = a_i$.

For some applications (see section 1.6), it may make sense to treat binary features as metric features with only two values.

Ordinal Features
Ordinal features are normally either transformed into cardinal or nominal features, details can be found in [41, 62, 138].

Cardinal Features
Similarity between cardinal features are often defined by the so-called $L-$ or Minkovski-Metrics:
Let v_i and v_j be two objects with corresponding features $(f_1^{(i)}, \ldots, f_k^{(i)})$ and $(f_1^{(j)}, \ldots, f_k^{(j)})$. Then the Minkovski-Metric is defined as: $-(\sum_{1 \leq q \leq k} |f_q^{(i)} - f_q^{(j)}|^r)^{\frac{1}{r}}$. For $r = 2$, we get the well known Euclidean distance (only with an extra "$-$"). Other cardinal similarity functions and information about mixing different types of features can be found in [41, 62, 138, 6].

Weighted Similarity
In most cases, weighted similarities are used. These functions first calculate the difference between individual pairs of features. The overall similarity is created by weighting the feature differences and by summarizing the resulting weighted differences. A typical example is the following functions:

Example 5.1.1 (Weighted Linear Similarity Function of Interaction Level 1)
Let $(f_1^{(1)}, \ldots, f_p^{(1)}), (f_1^{(2)}, \ldots, f_p^{(2)})$ be two feature vectors and let all features be cardinal. Then the weighted linear similarity function of interaction level 1 is defined by:

$$\sum_{i \leq i \leq p} w_i \cdot |f_i^{(1)} - f_i^{(2)}|, w_i \in \mathbf{R}$$

This is one of the most commonly used similarity functions.

Since all the functions in this section use individual feature differences, it makes sense to define a so-called difference vector:

Definition 5.1.1 (Difference Vector $\vec{d}$)
Let $(f_1^{(1)}, \ldots, f_p^{(1)}), (f_1^{(2)}, \ldots, f_p^{(2)}), p \in \mathbf{N}$ be two feature vectors. The vector

$$\vec{d}(\delta(v_1), \delta(v_2)) = (f_1^{(1)} \ominus f_1^{(2)}, \ldots, f_p^{(1)} \ominus f_p^{(2)}), \vec{d} \in \mathbf{R}^p$$

where $\ominus$ denotes the appropriate difference operator between the features (depending on their type, e.g. for cardinal features $a \ominus b \equiv |a - b|$), is called the **difference vector**.

Since in many cases, dependencies between elements of the difference vector exist, it makes sense to introduce the following class of similarity functions. These functions create additional features by multiplying the original features, e.g. a combined feature $\langle income \times gender \rangle$ is created by multiplying the original features income and gender.

Definition 5.1.2
(Weighted Linear Similarity Function of Interaction Level ρ)
Let $G = (V, E_c = \emptyset, E_v, \delta, w, \Delta)$ be a tabular data set and let sim be a similarity function for G. sim is called a weighted linear similarity function of interaction level ρ ($\rho \in \mathbf{N}_0$) if and only if it has the form:

$$sim(\delta(v_1), \delta(v_2)) = \sum_{1 \leq i \leq \rho} \sum_{A \in \mathcal{P}(\{d_l | 1 \leq l \leq p\}), |A| = i} w_A \cdot \prod_{\alpha \in A} \alpha$$

where $\delta(v_1) = (f_1^{(1)}, \ldots, f_p^{(1)}), \delta(v_2) = (f_1^{(2)}, \ldots, f_p^{(2)}), v_1, v_2 \in V, p \in \mathbf{N}$ and $\vec{d}(\delta(v_1), \delta(v_2)) = (d_1, \ldots, d_p)$ denotes the difference vector of v_1 and v_2.

Remark 5.1.1 *ρ is called the interaction level because a weighted linear similarity function of interaction level ρ takes interactions between $\leq \rho$ features of the difference vector into consideration.*

Example 5.1.2 (Weighted Linear Similarity Function of Interaction Level 2)
Let $(f_1^{(1)}, \ldots, f_p^{(1)}), (f_1^{(2)}, \ldots, f_p^{(2)})$ be two feature vectors and let all features be cardinal. Then the weighted linear similarity function of interaction level 2 is defined by:

$$\sum_{i \leq i \leq p} w_i \cdot |f_i^{(1)} - f_i^{(2)}| + \sum_{1 \leq i \leq p} \sum_{1 \leq j \leq p} w_{ij} \cdot |f_i^{(1)} - f_i^{(2)}|$$

Such a function has been used in section 7.1.1.

Remark 5.1.2 *The reader may note that learning such weighted functions means finding values for the parameter w_χ.*

5.2 Learning Similarity Functions

Especially in the field of Case-Based Reasoning (see section 3.2), several methods for learning weighted similarity measures have been developed. A short overview is given in section 5.2.1. New methods developed by the author are presented in section 5.2.2. These methods provide high level interfaces to support the knowledge-acquisition process.

The methods introduced in section 5.2.2 are applied to the problem of document retrieval and document management in section 7.1.1.

> **Parenthesis:** All methods for learning similarity measures can also be used in the field of Case-Based Reasoning (see chapter 3.2). Similarity measure are there used to find already known solutions for new problem descriptions.

Name	Type	Remarks	Literature
EACH	reinforcement-learning	extra parameters needed	[146]
RELIEF	reinforcement-learning	binary weights	[95]
CCF	statistical	only binary features	[28]
GM-CDW	statistical		[69]

Table 5.2: Some existing methods for learning similarity measures

5.2.1 Existing Methods

Methods for learning similarity measures can be divided into two main classes: (*i*) Methods using Reinforcement-Learning and (*ii*) Algorithms relying mainly on statistical analysis. Reinforcement-Learning methods normally predict a similarity and the user or a different system rates the prediction. Based on the rating, the weights are adjusted. Statistical methods analyze given examples and deduce appropriate weights. Table 5.2 gives some examples of well known methods. Other examples can be found in [16, 4, 157].

All these methods have in common that the knowledge acquisition step (identifying exemplar feature vectors including similarities) and the learning step (using the examples to learn weights) are not treated separately. The new method described in the next section differentiates between the two steps.

Not separating the knowledge-acquisition step from the learning step yields several problems:

- Since the expert is integrated into such methods in a predefined way, no flexibility is possible in the way in which the knowledge is obtained. Hence, most well-known knowledge-acquisition procedures cannot be applied.
- Most existing methods rely on well-known basic learning paradigms, e.g. on reinforcement-learning. Nevertheless, they do not reduce the learning problem to well-known learning algorithms (e.g. regression, neural networks) which realize those paradigms. Instead, they use specially developed proprietary algorithms. While for most known learning algorithms advantages and disadvantages have been examined, almost nothing is known about the applied proprietary algorithms.
- Verifying such algorithms is difficult, because learning problems cannot be distinguished from knowledge-acquisition problems.

5.2.2 New Methods

Stein and Niggemann presented in [162] a general methodology for learning similarity measures. Figure 5.2 shows this framework: Two main steps are identified, a knowledge acquisition step and a learning step.

As mentioned before, the definition of object similarities relies mainly on subjective matters: (*i*) What is the purpose of the visualization? Often the same

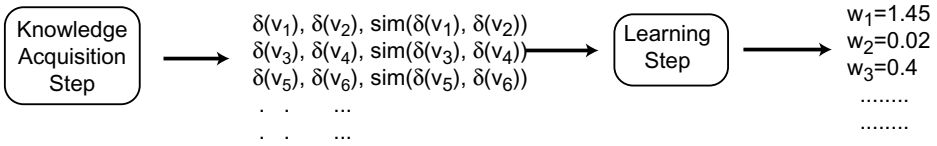

Figure 5.2: *A General Methodology for Learning Similarity Measures*

data has to be visualized in different ways in order to help with different questions. (*ii*) Who is using the visualization tool? Different experts often have different views of a domain. (*iii*) Different data sets often need different similarity measures. It is therefore desirable to use knowledge acquisition methods which allow for a fast and reliable definition of similarity measures.

The knowledge acquisition step uses existing knowledge sources (experts, databases, etc.) to get the necessary knowledge. This step always results in a set of feature vector pairs whose similarity is known (see also figure 5.2), e.g. $\{(\delta(v_1), \delta(v_2), sim(\delta(v_1), \delta(v_2)), (\delta(v_3), \delta(v_4), sim(\delta(v_3), \delta(v_4)), \ldots\}$

The second steps uses these rated vector pairs and applies a supervised learning strategy to find values for the weights w_i. In this work, regression and neural networks have been mainly used (see appendix A). The introduced general framework allows for the application of standard learning techniques.

Normally only a small (but typical) subset of the data set, called a learning set, is used to learn the similarity measure. The computed similarity measure is then used for a larger set of data. By using separate learning sets, two main problems arise: (*i*) The question of whether or not a given learning set is typical for a domain has to be answered for each domain separately. (*ii*) All applied learning algorithms have to abstract the small learning set in a reasonable way. The reader may note that the first problem concerns all learning processes; the data used for the learning must be correlated to the data used later on. The second problem is discussed in section A.

The next three sections introduce techniques for the knowledge acquisition step. All these methods have been applied in section 7.1.1 to the complex problem of document visualization and retrieval. For simplicity sake, a less complex problem is used here to illustrate the example: Finding similarities between dogs. Table 5.3 shows information about 8 dogs; each dog is described by 8 features (2 binary feature, 4 nominal features, 2 cardinal features).

Learning by Means of Given Similarities

This technique is quite trivial: The user provides the similarity for pairs of objects. This can be done by presenting two objects to the expert. The expert then assesses the similarity between this pair of objects, e.g. on a scale from 0 to 10.

Such methods have several disadvantages:

- ⊖ If m objects are used for the learning process, the expert has to rate $\frac{m^2}{2} - m$ object pairs.

- ⊖ The similarities have to be comparable, i.e. the expert has to decide whether object A is more similar to object B then object C to D. Such decisions can be quite difficult.

Name	Race	Age	Colour	Sex	Height	Exam.	Character
Murmel	Briard	1	black	f	64	no	friendly
Roma	crossbreed	1	gold	f	50	no	friendly
Carlo	Shepard	6	gray	m	62	yes	dominant
Keddy	Terrier	7	wheat	m	48	yes	unfriendly
Aiko	Briard	1	fauve	m	62	no	friendly
Mücke	Briard	2	black	f	64	yes	touchy
Bajaro	Shepard	2	gray	m	68	yes	friendly
Henry	crossbreed	1	brown	m	62	no	sensible

Table 5.3: Example: Dog Data

In the field of statistics, the problem of assessing object similarity is well known. Most methods[3] (e.g. ranking method, free sorting, anchor stimulus) rely on an user who defines all object similarities, i.e. no learning or abstraction mechanisms are applied. The reader may note that this method is used here in a different way: The expert assess the similarity of m object pairs. These assessments are used to learn a similarity measure. This similarity measure is then used to compute the similarity between $n, n \gg m$ pairs of objects.

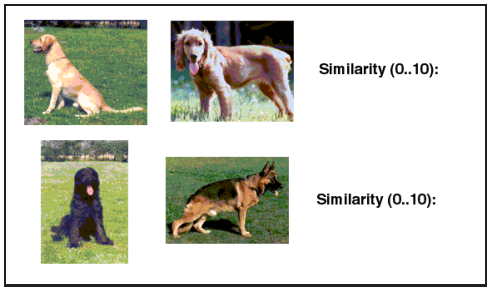

Figure 5.3: *Example: Learning by Means of given Similarities*

Normally, the user does not see the feature vector but the objects itself. In our dogs example, the user would be confronted with a questionnaire similar to figure 5.3.

Learning by Classification

For this method, the expert classifies the objects. The main assumption is that two objects are similar if and only if they belong to the same class. Let $G = (V, E_c = \emptyset, E_v, \delta, w, \Delta)$ be a tabular data set and $c : V \to \mathcal{K}$ be the classification function. $\mathcal{K}$ is the set of all possible classes. The function c is normally given by the user. The similarity sim is then defined by:

$$sim(\delta(v_1), \delta(v_2)) = \begin{cases} 1, & \text{if } c(v_1) = c(v_2) \\ 0, & \text{otherwise} \end{cases} ,$$

with $v_1, v_2 \in V$.

For our dogs example, reasonable classes would be {dangerous, unknown, harmless} or {hunting hound, tracking hound, family dog}. A possible questionnaire can be seen in figure 5.4.

[3] Good overviews can be found in [6, 17].

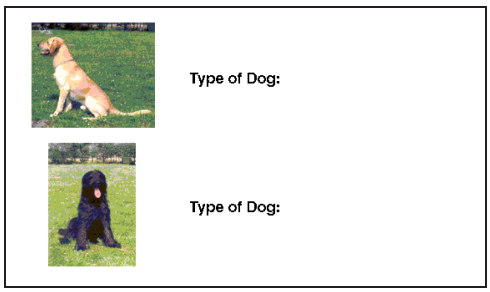

Figure 5.4: *Example: Learning by Classification*

This method has some interesting advantages:

⊕ m classifications define $\frac{m^2}{2}$ similarities.

⊕ Most experts have few problems classifying objects.

⊕ Several data sets come with a classification, e.g. the characters of the dogs are part of the given data set.

The main disadvantages are:

⊖ A disjunctive classification is sometimes difficult, e.g. not all dogs can be associated with a race.

While in the learning set only similarities 0 or 1 exist, learning algorithms as regression result in similarity measures which can yield arbitrary similarities. This is because learning algorithms abstract the given data.

Learning by Examples

The method presented now demands the least explicit knowledge from the expert and was first introduced by the author and Stein in [122]. The main idea is to have the expert give some exemplary visualizations, analyze them, and abstract a similarity measure. Since the visualization methodology used in this work requires the user to know the investigation goal (see beginning of this chapter), no additional explicit knowledge is demanded from the expert, i.e. this method can always be applied.

Again let $G = (V, E_c = \emptyset, E_v, \delta, w, \Delta)$ be a tabular data set (the learning set). The expert now manually defines a graph layout (see definition 1.4.2), i.e. he specifies a function $\rho : V \to \mathbf{N} \times \mathbf{N}$, which defines a two-dimensional position for each object. Figure 5.5 shows an exemplary drawing for our dogs example.

The similarity of two objects $v_1, v_2 \in V$ is now defined by:

$$sim(v_1, v_2) = -||v_1, v_2||_2,$$

where $||\ ||_2$ denotes the Euclidean distance between the position of v_1 and v_2. The following points distinguish this method:

⊕ The graphical definition of similarities is close to the mental model of the user.

⊕ By placing m objects, $\frac{m^2}{2}$ similarities are defined.

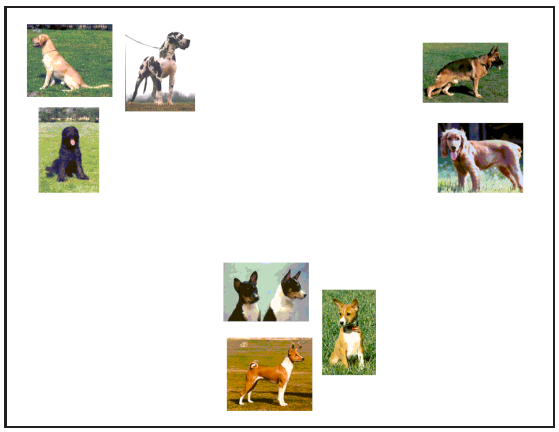

Figure 5.5: *Example: Learning by Example*

Some disadvantages exist:

⊖ Finding a good layout may overstrain the expert. This is mainly because by placing one object, the distances to $m-1$ other objects are defined. To solve this problem, only distances between close objects could be used. For this, the layout is first clustered, i.e. groups of closely related objects are identified (see section 2 for an introduction to clustering techniques). Three clusters can be seen in figure 5.5. Next, only distances to objects within the same cluster are considered. The distances between all other objects are defined by the cluster distances.

⊖ Implementing the necessary user interface may require some work.

5.3 Evaluation

As outlined in section 1.7, the methods described in this chapter are rated by the four features introduced at beginning of chapter 1. This allows for an assessment of whether or not the AI-visualization paradigm has been followed.

1. **Object-based Presentation**: All methods presented treat objects as atomic entities.

4. **User Adaption:** The three new knowledge acquisition methods presented above differ by the explicity of the knowledge demanded from the user.

   Method 1 (Learning by Means of Given Similarities) requires relatively complex knowledge about object similarities and the similarity of given object pairs has to be correctly assessed.

   Classifying objects (Method 2: Learning by Classification) is less demanding; instead of object pairs, only single objects have to be rated.

   Method 3 (Learning by Examples) hardly uses any explicit knowledge; the graphical definition used is very close to the mental model of the user.

   Experts rate the algorithms presented here as helpful and promising (e.g. in [162, 122]). In section 7.1.1, these methods are evaluated using a real-world problem.

At the end of part I of this work, it can be said that the methodology of structure-based visualization from section 1.3 can be implemented by a combination of existing algorithms, new methods, and extensions to existing techniques.

Part II

Implementing Visual Data Mining

The methods presented in the first part of this work have all been implemented and applied to various problem domains. In chapter 6, applications are presented where the data is given in the form of graphs. Chapter 7 concentrates on domains which use tabular data, i.e. data where the similarities between objects is unknown.

Chapter 6

Visualizing Models given as Graphs

All four applications presented in this chapter have in common that the to-be-visualized data is given in the form of a graph. Chapter 6.1 deals with the visualizing of traffic in computer networks. Visualizing knowledge-bases for resource-based configuration is the subject of chapter 6.2. In chapter 6.3, the methodology of structure-based visualization is applied to visualizing protein interaction graphs. Visualizing these graphs is useful in the context of the Human Genome Project. Chapter 6.4 focuses on the visualization of technical drawings, especially hydraulic circuits.

The software system StructureMiner has been applied to all four domains. It is described in detail in chapter 6.1, i.e. chapter 6.1 also serves as an introduction to the capabilities and to the usage of StructureMiner. The chapters 6.2, 6.3, and 6.4 focus on the aspects peculiar to the respective domains.

6.1 Network Traffic

Modern computer networks are designed to allow for a fast and unimpeded communication between computers. To guarantee a problem free operation, the administrator is faced with the task of understanding the traffic. If the traffic is well understood, the administrator can identify network components constituting bottlenecks for the traffic (see section 6.1.2) or even detect critical network situations (see [125]).

For the following reasons, understanding network traffic is difficult:

- ☞ Even small local networks (LANs) consist of several hundred computers. Large company or university networks often comprise several thousand computers. These computers communicate with each other and with computers outside the LAN. Furthermore, the amount of traffic between two computers, its temporal distribution, and the used protocols vary greatly. In order to help the administrator understand the traffic, automatic abstraction of the data and a high-level visualization is necessary.

- ☞ Almost nothing is known about the structure of the traffic beforehand. Terms like “strong traffic,” “weak related subnets,” or “strong internet user” have a totally different meaning in different contexts.
- ☞ Much information about the traffic can not be concluded from static information (e.g. the traffic of an hour or a day), but the temporal variation of the traffic structure must be examined.
- ☞ Because of new users, new technologies, and new scopes of duties within the organization, the overall structure of the traffic can rapidly change.

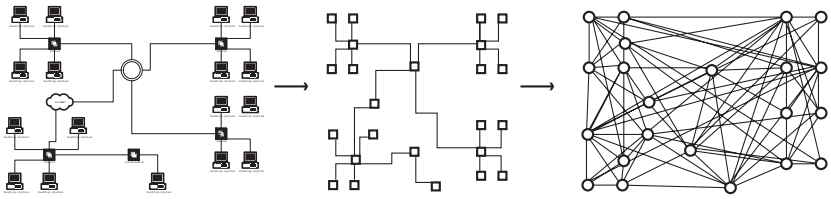

Figure 6.1: *Transforming network traffic into a graph*

To address these problems, an abstract communication model is formed automatically: Every active network component (PC, router, switch etc.) forms a node in a so-called traffic graph. If and only if two nodes (i.e. network components) communicate with each, they are connected with each other by an edge which is weighted by the amount of corresponding traffic. Figure 6.1 illustrates this: On the left side, the network topology can be seen; each edge models a direct communication link. In the middle figure, network components are understood as nodes of a graph. The edges of the right-handed graph are communication links and this graph must be visualized in order to enhance the user’s understanding of the network traffic.

Formally, a network is mapped onto a visualization graph $G = (V, E_c, E_v, \delta, w, \Delta)$ (see definition 1.4.1) as follows:

- V is the set of network components.
- E_c is the set of communication pairs in the network, i.e. every edge in E_c connects two components which communicate with one another..
- E_v is normally identical to E_c. Sometimes E_c represents the network topology, i.e. each edge in E_c represents a direct connection (e.g. a cable) between two network components.
- $\delta(v), v \in V$ is the IP address of node v.
- $\Delta = \{0, \ldots, 9, '.'\} \cup \{',', '(', ')'\}$
- $w(e), e = \{v_1, v_2\} \in E_c$ is the amount of traffic between v_1 and v_2.

This graph is now visualized using the method of structure-based visualization presented in chapter 1.3.

The user can choose between several clustering methods for step 1 (Structure Identification), allowing him/her to evaluate the applicability of cluster methods to given graphs.

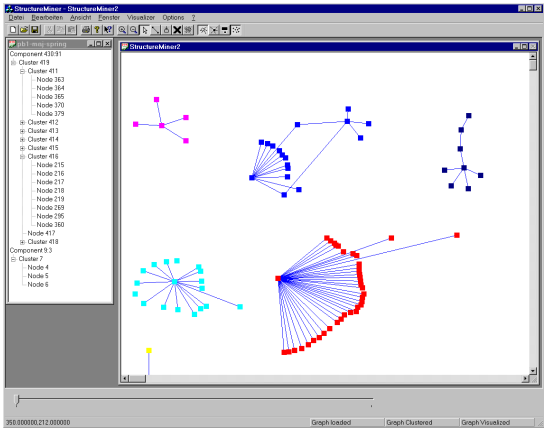

Figure 6.2: *Using* StructureMiner *for Visualizing Network Traffic*

Step 2 (Structure Classification) is implemented as follows: The user can label clusters. When later on, a new cluster in the same network as before is found, the label of the most similar known cluster is used. Typical clusters are “Team 2”, “Department C”, or “Project A”. In most computer networks, the clusters do only change slightly over a period of time; normally only a few computers change their cluster membership. This classification approach may be viewed as a form of case-based classification (see section 3.2). The classification of traffic clusters is the subject of section 6.1.2.

For step 3 (Structure Envisioning), various graph drawing methods (see chapter 4) can be applied. This allows the user to choose the most suitable layout method for each graph or cluster separately.

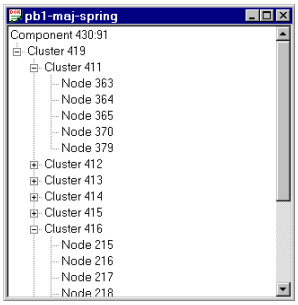

Figure 6.3: *The Tree View of* StructureMiner

The analysis and visualization approach presented here is totally different from the methods used so far: Most existing systems mainly show statistical analysis of traffic data (e.g. average traffic on line A, lost packets between router 1 and switch 2 etc.). Typical academic research can be found in [72, 71, 10], established commercial systems are e.g. CINeMA (Hirschmann), TrendReporter (NetScout Systems), NetworkAudit (Kaspia Systems), Optivity (Bay Networks), Observer (Network Instruments), or Surveyer (Shomiti).

6.1.1 Using StructureMiner

Figure 6.2 shows the graphical user interface of StructureMiner: Each project consists of a tree view (on the left side) and of several graph views. The tree view shows the structure of the graph. On the top level, the connected components of the graph can be seen. The clusters are arranged below the respective connected component. If subclusters exist, they can be found below their parent cluster. In figure 6.3, a tree view can be seen in detail.

For each connected component, a graph view exist which is used to visualize the graph. Figure 6.4 shows a graph view: The separate clusters are emphasized by not showing edges between clusters and by bounding boxes. Figure 6.4 depicts the same graph, but now the edges between clusters (sometimes called external edges) are drawn.

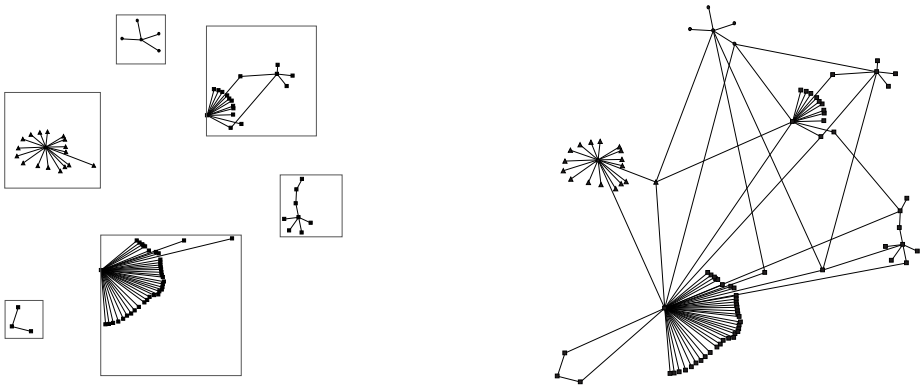

Figure 6.4: *Left side: The* StructureMiner *Graph View. Right side: Showing External Edges*

The user is often mainly interested in the graph's structure: Figure 6.5 shows only the structure. Figure 6.5 depicts a random layout; the reader may note that the structured approach improves the comprehensibility of the graph layout.

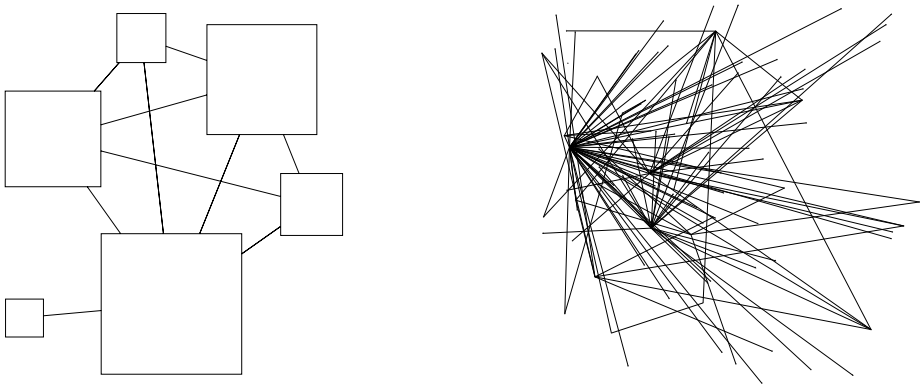

Figure 6.5: *Left side: The Graph's Structure. Right side: A Random Layout*

The recursive structure of the graph is not only visible in the tree view; but also appears in the graph view: Figure 6.6 shows a top-level cluster: By choosing "Fold-in" from its context menu, the subclusters are shown (figure 6.6).

In STRUCTUREMINER, different cluster algorithms can be applied to each connected component:

- ☞ MAJORCLUST (see section 2.2.1)
- ☞ Hierarchical MAJORCLUST (see remark 2.2.4)
- ☞ Kohonen (see section 2.1.4)
- ☞ MinCut (see section 2.1.2). This version of MinCut Clustering has been developed and implemented in [111]: The iterative division of clusters at their minimum cut stops, if the λ-value (see section 2.2.2) of the clustering worsens.

The step of cluster classification is explained in section 6.1.2.

Figure 6.6: *Left side: Before the "Fold-in" action, Right side: After the "Fold-in" action*

After the clustering step, the graph can be visualized by applying one of six layout methods:

- ☞ Force-directed Layout (see section 4.1.1): STRUCTUREMINER offers two versions of force-directed layout, Tunkelang's method[1] and Eades's spring-embedder.
- ☞ Level-oriented Layout (see section 4.1.3)
- ☞ Circular Layout (see section 4.1.4)
- ☞ Factor-based Drawing (see section 4.2). An optional incremental spring-embedder (see section 4.2) can be applied as a post-processing step.
- ☞ All-Pair Factor-based Drawing (see section 4.2).
- ☞ Random Layout

It is possible to choose the drawing method for each cluster individually. This can be seen in figure 6.7 (the actual drawing method is marked). The new drawing method can be applied recursively to the subclusters as well.

Further Features of StructureMiner

Further features of STRUCTUREMINER are:

- ☞ Using a slider control, only edges with weights above a given threshold γ are shown, i.e. $E_v = \{e \in E_c \mid w(e) > \gamma\}$. This allows for the concentration on important edges.

[1] Implemented in [178].

- ☞ In a separate file, the user can associate name templates with colors, i.e. node names matching the regular expression “controller*.pb*” (e.g. controller2.pb3) may be associated with the color red. Nodes are then visualized using the appropriate color.
- ☞ The user can zoom into (and out of) the graph.
- ☞ Several graphs can be visualized at the same time.
- ☞ Graphs can be printed and exported (e.g. into the wmf-format).
- ☞ Nodes and edges can be added, deleted and moved, i.e. StructureMiner works also as a graph editor.

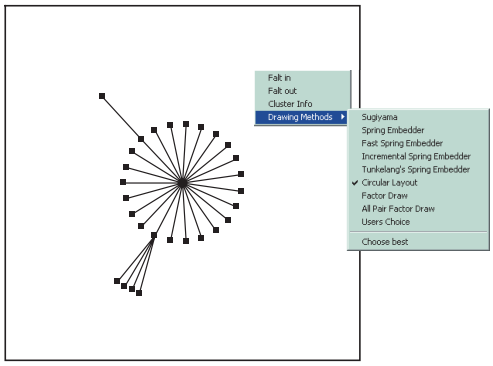

Figure 6.7: *Individual Drawing Methods for each Cluster*

Features especially developed for network administration are described in section 6.1.2. J. Tölle from the Network Institute of Prof. Martini at the University of Bonn served as a network expert for section 6.1.2. He also provided most of the knowledge concerning intrusion detection in [124]. The author concentrated on the machine learning and visualization aspects.

6.1.2 Administration Tasks

The three following extensions to StructureMiner became necessary in order to agree with demands from network administrators:

Classification

The administrator should be able to label clusters, typical labels are e.g. “Project A” or “AI Lab”. When a new cluster is identified, it should automatically be given the same label as the most similar cluster seen before, i.e. the clusters are classified using a case-based approach (see section 3.2).

In order to find a similar cluster, in case-based classification a so-called similarity measure is used. Here the similarity measure is a function which measures the similarity between two clusters (see 3.2 for details). When a new cluster is found, it is compared to all previous clusters using the similarity function. The label of the previous cluster which yields the highest similarity (i.e. the highest value of the similarity function), is also used for the new cluster. An automatically labeled traffic graph can be seen in figure 6.8.

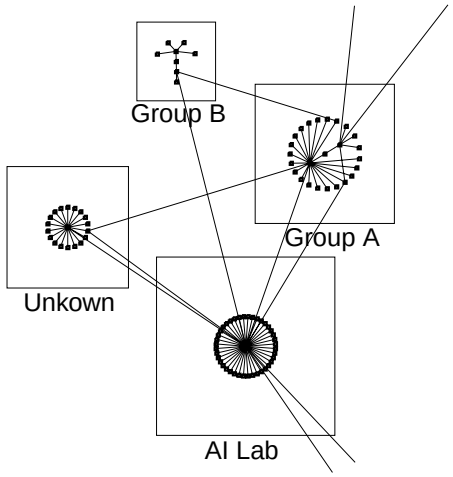

Figure 6.8: *An automatically classified traffic graph*

In the rest of this section, let $G = (V, E_c, \sigma, \Sigma)$ be a traffic graph and let the node label function σ be a one-to-one function, i.e. the node labels are unique. All node labels have a length of $k = \lceil \log |V| \rceil$, i.e. $\forall v \in V : \sigma(v) = l_1 \dots l_k$ with $l_i \in \Sigma, 1 \leq i \leq k$. Furthermore let $num : \Sigma \to \{1, \dots, |\Sigma|\}$ be an one-to-one function mapping letters in Σ onto numbers.

For network administration purposes the following similarity measure makes sense:

Definition 6.1.1 (Cluster Similarity)
Let C_1 and C_2 be two clusters of the same graph $G = (V, E_c)$, i.e. $C_1, C_2 \subseteq V$. C_1 and C_2 need neither be disjunctive nor belong to the same clustering (see definition 2.0.1). Normally C_1 denotes an already labeled cluster created by a previous clustering, while C_2 denotes a cluster found by a later clustering. The similarity between C_1 and C_2 is used to decide whether the label of C_1 should also be used for C_2. Then the similarity function sim between two clusters is defined as:

$$sim(C_1, C_2) = \frac{|C_1 \cap C_2|}{\max(|C_1|, |C_2|)}$$

Remark 6.1.1 *For administration purposes, clusters or groups of users are normally identified by their nodes. Therefore its makes sense not to use the edges to define cluster similarities.*

Other functions are possible, e.g. functions that also take the non-similar nodes into consideration.

Theorem 6.1.1
For this theorem, we assume that all clusters are subgraphs of a given graph $G = (V, E_c)$ (the computer network). Let $C = \{C_1, \dots, C_q\}$ denote a set of already labeled traffic clusters. A new cluster C_0 can be labeled using the similarity measure sim from definition 6.1.1 in $O(|C_0| + |V| \cdot \mu)$ time where $\mu = \max_{v \in V}(|\{C_i \mid 1 \leq i \leq q \wedge v \in C_i\}|)$.

Proof. An algorithm is given which labels clusters in $O(|C_0| + |V| \cdot \mu)$ time:

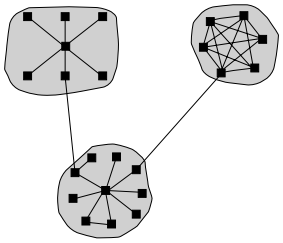

Figure 6.9: The Animation Graph

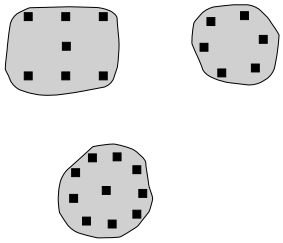

Figure 6.10: Animation Step 1

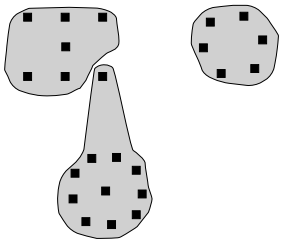

Figure 6.11: Animation Step 2

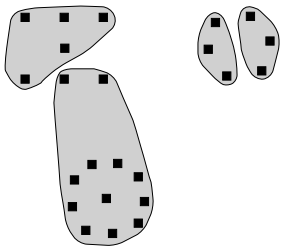

Figure 6.12: Animation Step 3

(1) It is assumed, that every node $v \in V$ is linked to a set $\rho(v)$ of clusters which comprise v. ρ can be computed in a preprocessing step.
μ is equal to $\max_{v \in V} |\rho(v)|$.
(2) $\forall C_i \in C$ **do** $no(C_i) = 0$ **od**
(3) $\forall v \in V$ **do**
(4) **if** $v \in C_0$ **then**
(5) $\forall C_i \in \rho(v)$ **do**
(6) $no(C_i) = no(C_i) + 1$
(7) *remember the cluster C_{best} with the highest value of no*
(8) **od**
(9) **fi**
(10) **od**

Initially, $O(|C_0|)$ time is needed to inform a node $v \in V$ whether it belongs to C_0. Since C_0 is a subgraph of G, ever node can be informed in $O(1)$ time. This information is needed in step 4.

Step 3 causes $|V|$ iterations. In each iteration, step 5 causes $O(\mu)$ executions of steps 6 and 7. Since step 6 and 7 need $O(1)$ time, the overall run-time is $O(|C_0| + |V| \cdot \mu)$.

Animation

The administrator is also highly interested in understanding the change in traffic over a day, a week, or a year. Since the amount of changes makes a quantitative presentation impossible, a qualitative visualization of traffic change has to be found. For this, the technique of structure-based visualization (see section 1.3) can be extended as follows:

The network traffic is now defined by a set of traffic graphs which share the

same nodes: $\mathcal{G} = \{G_1, \ldots, G_p\}$ with $\forall 1 \leq i \leq p : \; G_i = (V, E_c^{(i)}, w^{(i)})$. Each graph represents the traffic over a given period of time (e.g. the graph G_1 may represent the traffic from 8am to 9am, G_2 is the traffic from 9am to 10am etc.).

The method of structure-based visualization is now applied to the graph G_1, i.e. it is clustered and visualized. The edges are normally not shown. Colors are used to highlight clusters. The user may note that since structure-based visualization has been used, clusters are emphasized by the spatial distribution of nodes. An example can be seen in figures 6.9 and 6.10 where the clusters are represented by borderlines. The first figure shows the visualized graph G_1, the second figure shows the same graph layout, but without the edges.

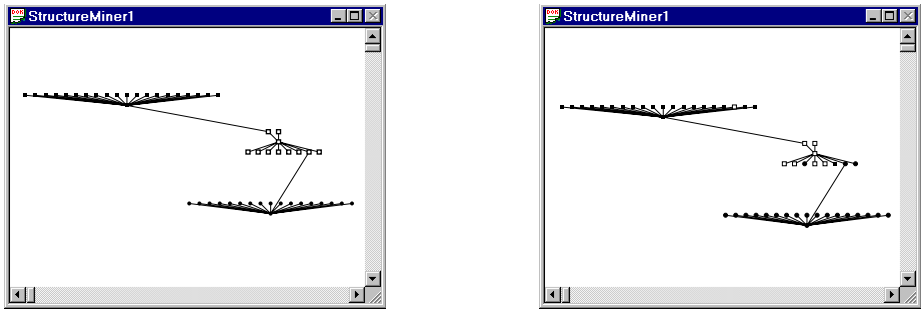

Figure 6.13: *Traffic animation with* STRUCTUREMINER*: Three clusters are visualized, the clustering changes over time.*

Next the graph G_2 is clustered. This new clustering is applied to the old layout, i.e. all nodes keep their positions, but they may change their cluster membership. A new cluster membership results in a different node color / shape. This process continues for the graphs $G_i, i > 2$. Figures 6.11 and 6.12 depict typical cluster changes.

Figures 6.13 and 6.13 show an example: Traffic changes recorded at the University of Bonn throughout the course of a day.

> **Parenthesis:** The reader may note that it is helpful for the user if clusters keep their color / shape during the animation. This may look like a simple problem, but it is essentially the cluster classification problem described earlier in this section. Because clusters change during the animation, it is necessary to identify the cluster most similar to a new cluster from the previous animation step.

Analyzing the Network Topology

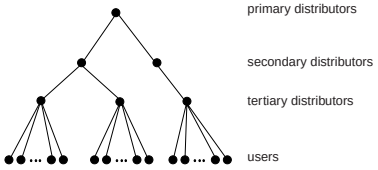

Figure 6.14: *Structured Cabling*

Three different views of a computer network exist. Each view is defined by a graph:

1. **Cabling Graph G_C:** Edges of this graph model cables. Usually, every computer is connected to a so-called tertiary distributor by a single cable. The tertiary distributors are again connected to secondary distributors which are connected to a single primary distributor. Therefore, the cabling forms a tree[2] (see figure 6.14). The reader may note that (*i*) distributor nodes may be connected by several cables and (*ii*) the purpose of a cable is unspecified.

   Computers and distributors form the nodes of G_C and the edges are defined by the tree structure. This view onto the network is called the cabling or the physical topology of the network.

2. **Logical Topology G_L:** The logical topology defines which computers can communicate directly with each other. This view of a network may differ from the cabling, e.g. by connecting two cables to each other, tree-like cabling can be used to form a ring. Unlike the cabling, the logical topology also specifies which active network components (e.g. switches, routers, computers) exist and how they are connected to each other. E.g. a tertiary distributor node in G_C may comprise m switches and n hubs. Active network components form the nodes of G_L and edges represent direct communication links[3].

3. **Traffic Graph G_T:** As defined above, nodes of traffic graphs are active network components while edges model network traffic. Logical topology and traffic graphs are also explained in figure 6.1.

Figure 6.15: *A visualized physical topology G_C (left side) and a superimposed clustering of the traffic graph G_T.*

Structure-based visualization can now be used to solve two typical network problems:

1. **Network Planning:** Choosing a network topology G_L which takes the given cabling G_C and the expected traffic graph G_T into consideration is a major network planning task. Structure-based visualization is applied as follows:

   (a) Visualize G_C

   (b) Identify a clustering C_T of G_T

[2] This type of cabling has been standardized in EN 50173 and ISO / IEC DIS 11801.
[3] I.e. direct links on layer 2 of the OSI model.

(c) Use node colors / shapes to superimpose C_T over the visualization of G_C (while keeping all node positions). An example can be seen in figure 6.15.

The planer can now try to find a logical topology which allows for a fast communication between nodes within the same cluster.

One typical problem consists of grouping the set of computers connected to a tertiary distributor. Computers in the same group are connected using a single hub or a switch. Because of technical restrictions, only a limited number of computers can be connected to a switch or hub. Since communication within a group is faster than the communication between groups, the administrator wants computers which communicate with each other quite frequently (i.e. computers in the same cluster) to be in the same group.

Parenthesis: Finding an optimal grouping of computers can also be viewed as a k-means clustering problem (see section 2.1.4). An alternative is to apply MinCut Clustering (or Nearest-Neighbour clustering) and stop the division (or union) step when the given number of clusters is reached.

2. **Analysis of an Existing Logical Topology:** StructureMiner can help the network administrator to examine an existing logical topology. This can be done as follows:

   (a) Visualize G_L

   (b) Identify a clustering C_T of G_T

   (c) Node colors / shapes are used to superimpose C_T over the visualization of G_L (while keeping all node positions).

   By comparing the structure of G_L (represented by the layout) with the clustering C_T of G_T (represented by node colors / shapes), the administrator can see whether or not the existing logical topology still makes sense. The more C_T corresponds to the layout, the more appropriate the logical topology is.

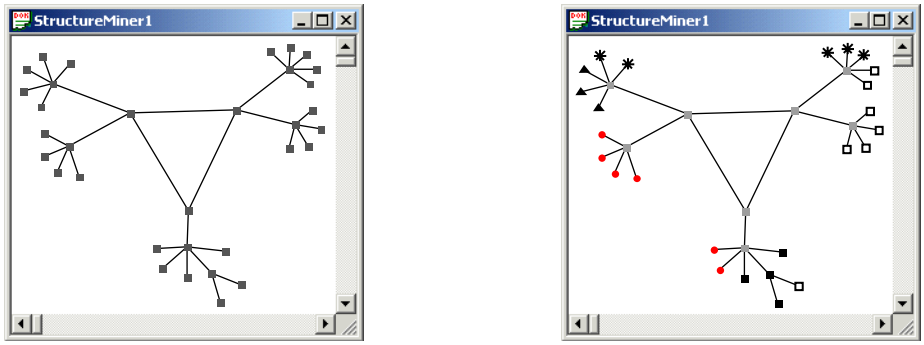

Figure 6.16: *A network topology (left hand side) and the superimposed traffic clusters (right hand side). The clusters are represented by different node shapes.*

Figure 6.16 shows a typical example: The node groups at the top and at the bottom of the screenshot are cut open by the traffic clusters (MAJORCLUST has been used as a clustering algorithm).

6.1.3 Simulation

Regarding visualization and simulation as a symbiotic combination makes sense for several domains: Whenever a simulation is done in order to present the results to a user, both visualization and simulation must be examined together. Therefore, it is helpful to differentiate between two classes of simulation tasks:

1. **Internal Simulations:** The results of these simulations are only used within a larger algorithm and they are never shown to the user. A typical example is the field of configuration: Technical systems consist of several small components. The configuration task is to combine components in such a way that the system implements a given function. Many systems propose a configuration and test it by using a simulation (see e.g. [164, 158]). When the simulation shows a discrepancy between the wanted and simulated behaviour of the system, the configuration is changed. Another example is the model-based diagnosis of technical systems (e.g. in [165, 46]): A diagnosis is constructed by comparing the normal and observed behaviours of a technical system. The normal behaviour is found by means of simulation.

2. **External Simulation:** These simulations are done with the sole purpose of showing the results to the user. An example will be given in this section.

External simulation has two natural connections to the field of visualization:

- ☞ The simulation results must be visualized.

- ☞ We presume that a user wants to simulate a given system and that the simulation results are presented to the user. The system is modeled on three different levels (see also figure 6.17): The user has a mental model M of the given system, i.e. the user thinks about the system on a specific abstraction level. This system is formally defined on a modeling level M', which should be as close as possible to M. M' is used as a modeling level for the computer.

  For many domains, the same modeling level is used for the presentation of the simulation results. This makes sense, because M' is an abstract level close to M, but is also manageable by a computer. For some domains, different modeling levels for the input and for the visualization are possible.

  The simulation is done on a modeling level M''. The simulation result R is then transformed back onto the input and visualization level, resulting in R'. The user sees and comprehends R' by transforming it into a mental model R''.

Choosing a modeling level for M'' is normally a difficult problem, since a simulation should be fast, simulate all important variables of the system, should often be able to work with unprecise input (e.g. qualitative simulation in [102]), and should also result in easily comprehendable data. The examination of the simulation / visualization combination leads to some hints for the modeling level M'': Since the input is transformed from the modeling level M' into M'' and the simulation result R is transformed back into R' on the visualization level, it can be concluded that visualization and simulation levels must be chosen in such a way, that model transformations can be easily carried out. A possible solution would be to choose two similar or equal modeling levels.

In many cases, users carry out manual simulations on the level M'. Therefore, it often possible to choose M'' close to M'.

These thoughts can be summarized as follows: Whenever possible, a simulation level close to a reasonable visualization level should be chosen.

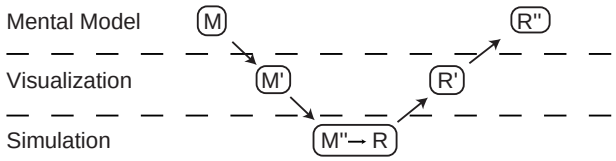

Figure 6.17: *Model-transformations for the Visualization of Simulation Results*

The rest of this section presents a system for the simulation of network traffic. This system has been developed and implemented by the author. While the main focus of the description lies on the connection between visualization and simulation, this section may also be worth reading for someone interested in the field of network simulation.

The simulation system presented here has been developed to support the planning of networks. In order to choose appropriate components for a computer network, the planer has to know what the demands on the components are. The demands are mainly defined by the expected traffic in the network, i.e. the more precise the traffic is known beforehand, the better the planning can be. The subject of this section is the network simulation only; further aspects of network planning and configuration are not described here.

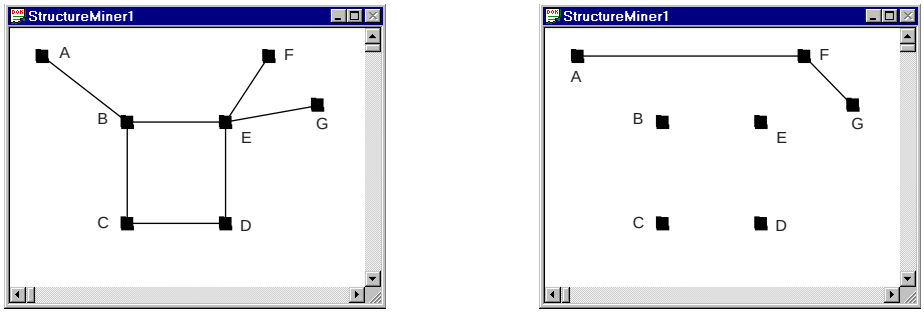

Figure 6.18: *Left side: The topology of an example network, every edge represents a physical connections between network components. Right side: The traffic connections of the example network, each edge represents network traffic.*

Existing simulation algorithms for computer-networks were mainly developed for an in-depth analysis of protocols and network-configurations. Typical systems employ discrete-event-simulation, e.g. in [127, 53, 144, 151]. However, this work concentrates on the demands of a network planner or administrator: a fast and understandable simulation of large networks. Furthermore, since future network traffic as caused by the users is only vaguely known, in-depth analysis takes second place to a reasonable processing of vague and abstract terms of traffic-load.

Most existing algorithms pay for their accuracy with a slow run-time behaviour and with a dependency on precise inputs, making them inappropriate for planning and administration tasks. Therefore, the system presented here uses fast algorithms which can work with imprecise input.

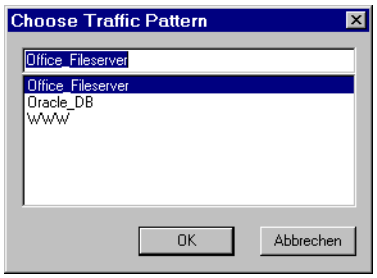

Figure 6.19: *Choosing a Traffic Pattern*

The simulation system is based on StructureMiner. One main difference is that now two sets of edge are used. One set E_v represents the logical structure of the network (i.e. each edge connects two components which can communicate directly with each other) and another set E_c represents the communication links (i.e. each edge connects two components which communicate with each other). Figure 6.18 shows the logical view of a small example network. Figure 6.18 depicts the communication links of the network, $A - F$ and $F - G$ are the only communication edges.

The following input and visualization modeling level has been chosen: Each communication pattern between two nodes v_1, v_2 is modeled as a function $p_{v_1,v_2} : \mathbf{N} \rightarrow [0, 1]$. p_{v_1,v_2} maps the amount of traffic per second onto a probability, i.e. $p_{v_1,v_2}(1000) = 0.1$ means that with a probability of 0.1 1000 bytes per second are send from v_1 to v_2. The reader may note that traffic is not modelled as a time-dependent function; only a static view of the traffic is used.

Since the user does not want to define such a function for each communication link, standard functions can be used. The user simply chooses for each communication link respective traffic patterns from a set of predefined functions (see figure 6.19), i.e. a communication link is defined as a typical WWW communication of a secretary.

> **Parenthesis:** Typical communication patterns have been found by analyzing network traffic. For this, traffic has been recorded using a software probe and analyzed statistically.

The simulation itself is first explained using the example from figures 6.18 and 6.18, a formal algorithm is presented afterwards: Figure 6.20 shows the

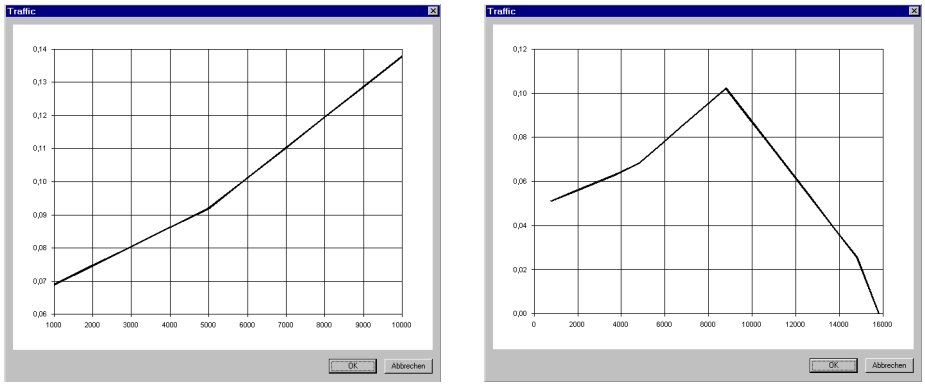

Figure 6.20: *Traffic Pattern 1 (left side) and Traffic Pattern 2 (right side)*

communication pattern $p_{A,F}$ for the traffic from node A to node F, figure 6.20 shows the pattern $p_{F,G}$. The traffic for each communication pair is routed through the (physical) network, e.g. the traffic from A to F uses the way $A - B - E - F$.

This way is normally defined by routing schemes which are known for most networks. Here the shortest way routing has been used (see [58, 170] for details and for other routing schemes). The simulation method presented here also allows for the comparison of different routing schemes.

For each physical edge, the communication patterns using this edge are summarized. Since the patterns p_{v_1,v_2} are probability density functions, two patterns can be summarized using the method of convolution (see [62]). The result for the logical edge $E - F$ can be seen in figure 6.21; the functions $p_{A,F}$ and $p_{F,G}$ have been summarized. The traffic on the edges $A - B, B - E$ is defined only by $p_{A,F}$ (since only this communication pair uses those edges) and the traffic of $E - G$ corresponds to $p_{F,G}$.

The implementation uses discrete functions to approximates the continues functions p_{v_1,v_2}.

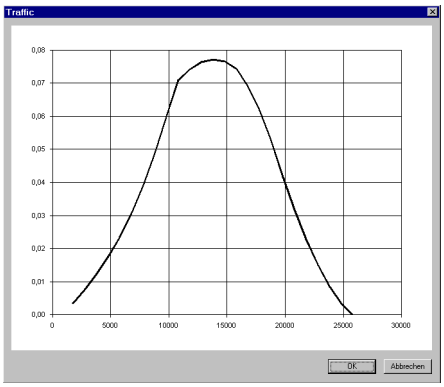

Figure 6.21: *The resulting Superimposition of Traffic Patterns*

The algorithm for the simulation can be stated formally as follows:

Algorithm 6.1.1 (Statistical Simulation)
Input. A graph $G = (V, E_l)$,

where E_l defines the logical topology of the network,
a set $S = \{p_{v_{i_1^{(1)}}, v_{v_{i_1^{(2)}}}}, \ldots, p_{v_{i_k^{(1)}}, v_{v_{i_k^{(2)}}}}\}$ of
probability density functions with $v_{i_r^{(q)}} \in V$
Output. A probability function f_e for each $e \in E_l$

(1) $\forall e \in E_l$ ***do*** $f_e(x) = \begin{cases} 1, & \text{if } x = 0 \\ 0, & \text{otherwise} \end{cases}$ **end**
(2) $\forall p_{v,w} \in S$ ***do***
(3) *let $w = (e_{j_1}, \ldots, e_{j_g}), e_{j_t} \in E_l$ be the way*
of the traffic from v to w using the edges E_l
(4) $\forall e \in w$ ***do***
(5) $f_e(x) = \int_{\infty}^{\infty} f_e(x-y) p_{v,w}(y) dy$
// convolution

This simulation approach has been chosen for the following reasons:

⊕ The simulation meets the demands of a network planner. A network planner is mainly interested in the amount of traffic which the network components have to cope with. A pure worst-case scenario, i.e. adding the maximum possible traffic for each logical edge, does not help the planner, because he/she has to decide whether better components are worth the money. I.e. he/she must know how often the cheaper components would cause traffic delays or network problems. By modeling the traffic with probability density functions, not only the possible amount of traffic, but also the probability for a special traffic situation is simulated.

⊕ The approach is also abstract enough to allow for the processing of only vaguely known traffic patterns. The exact future behaviour of a user is unknown, but typical statistical characteristics of his/her behaviour are normally known (e.g. the typical traffic patterns caused by a secretary accessing the WWW). The change of traffic over time is especially unknown for future traffic.

⊕ The statistical simulation presented here is fast enough to simulate even large networks. The reader may note that calculating the convolution requires $O(k^2)$ time (k being the number of traffic intervals used for the discrete probability functions). The number of convolutions is bounded by the overall length of paths used by traffic in the physical network.

⊕ The simulation does not have to take all technical restrictions (e.g. maximum traffic allowed a special switch) into consideration. This makes sense, since we are not simulating a real network, but trying to finds demands on a new network. E.g. the user wants to identify an appropriate switch, but is not interested in evaluating an existing one.

The main disadvantages are:

⊖ Typical traffic patterns have to be known.

⊖ Unlike discrete-event-simulations, temporal change of the traffic can only be modelled by a series of different simulations.

⊖ Some network events (buffer overflows) cannot be precisely predicted.

The simulation method presented here allows for the simulation of even large networks (about $500 - 1000$ nodes) within a few seconds. It has been tested with several realistic networks (30-40) and the results proved helpful for network planners and administrators. Simulation results have been compared to measurements in real networks (traces with software-probes). In order to verify the simulation results, a test laboratory has been used. In this laboratory, different network topologies can be created and software systems on the computer (PCs and SUNs) generate communication patterns. The resulting traffic has been recorded and has been compared to the predictions of the simulation method presented in this paper. The simulation results resembled reality closely.

When asked about the optimal method for the presentation of recorded network traffic, most network administrators prefered probability density functions. Therefore, input/visualization modeling level and simulation level are identical. Many of the advantages of the simulation approach, are also applicable to a visualization using probability density functions: (*i*) The visualization is abstract enough to allow for an overview of large amounts of traffic. (*ii*) The probability of problematic traffic situation is visible. (*iii*) The precision of the visualization corresponds to the precision of the knowledge about future traffic and to the reliability of the knowledge about user behaviour. These correspondences are due to the unique mental model of the user.

Furthermore, the simulation results can be visualized using the visualization method for network traffic described earlier in section 6.1.1. The only remaining question is how the probability function p_e of an edge e can be transfered into an edge weight. A good solution is to use the centroid of the function as the corresponding edge weight:

$$w(e) = \frac{\int_{-\infty}^{\infty} x p_e(x) dx}{\int_{-\infty}^{\infty} p_e(x) dx}$$

The simulation system presented here is, besides being beneficial to the network planner, a good example of the close relationship between simulation and visualization. It can be seen that, when developing a system for external simulation, it makes sense to regard visualization and simulation as a whole and to choose similar or even identical modeling levels.

6.1.4 Discussion

In order to evaluate the analysis and visualization approach of structure-based visualization, traffic recorded at the universities of Paderborn and Bonn is used. Furthermore, ≈ 100 artificial traffic situations have been generated in the laboratory described in section 6.1.3. For these artificial situations, the correct clustering[4] is known. Hierarchical MAJORCLUST is able to identify 87% of the given traffic clusters. MinCut-Clustering[5] finds 53% and Kohonen-Clustering finds 76% of these artificial clusters. MinCut-Clustering mainly

[4]Since the laboratory comprises only ≈ 10 computers, only divisions into 2 clusters have been generated.

[5]The implementation from [111] has been used.

suffers from the problem described in figure 2.12, while MAJORCLUST and Kohonen-Clustering sometimes fails to differentiate between the two given clusters. As mentioned before, MAJORCLUST has problem finding small clusters ($|V| < 10$).

The size of real traffic graphs (300-600 nodes) makes an application of Min-Cut-Clustering hardly possible (see also figure 2.25). For most graphs (44 out of 51 traffic graphs) Kohonen-Clustering finds reasonable clusters, but needs additional parameters in order to detect the correct number of clusters (see section 2.1.4). In all cases, users find the results produced by hierarchical MAJORCLUST quite adequate. MAJORCLUST needs only a few seconds to compute the clustering, while Kohonen clustering needs $\approx 1-2$ minutes.

Because traffic graphs normally have an underlying star-like structure[6], circular layout (see section 4.1.4) yields the best results. Even the usually good factor-based layout can not compete in this domain with circular layout. The figures 6.22 and 6.22 show the same traffic graph visualized using circular and factor-based layout.

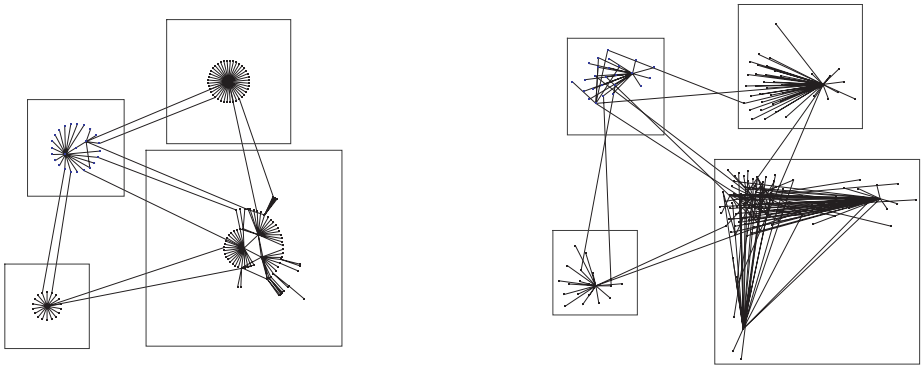

Figure 6.22: *Left side: A traffic visualization using* MAJORCLUST *and circular layout. Right side: A traffic visualization using* MAJORCLUST *and factor-based drawing*

Experts rate the results of structure-based visualization as very helpful (see e.g. [124, 125, 160]). The combination of clustering, classification, and graph drawing allows for the analysis of otherwise incomprehensible traffic situations.

Some open problems remain:

- It would be helpful if data could be read directly from network probes.
- Much research is needed to apply the clustering and visualization methods to the problem of intrusion detection. Some questions concern the usage of rule learning paradigms, appropriate visualization methods, and user modeling (hacker modeling). Early answers to these questions can be found in [124].

In Conclusion, it can be said that the methodology of structure-based visualization allows for a decent visualization of network traffic. Users are able to identify unknown patterns and information in the data. The application of clustering and classification proves especially helpful.

[6] This is caused by the typical client-server relations.

6.2 Configuration Knowledge-Bases

The configuration of technical systems is one of the oldest fields of Artificial Intelligence. Configuration denotes the task of assembling a system from given atomic components. In most cases, the system has to fulfill given demands and has to minimize a cost function.

Resource-based configuration is one of many existing configuration methods[7]. Components are described by means of functionalities which are offered or demanded. Initially, a set of wanted functionalities is given. A solution is the cheapest set of components which leaves no demands unfulfilled.

Let $C = \{c_1, \ldots, c_k\}$ denote the set of components and $F = \{f_1, \ldots, f_h\}$ the set of functionalities. Each component $c_i \in C$ demands functionalities $\text{demand}(c_i) \subseteq F$ and offers functionalities $\text{offer}(c_i) \subseteq F$.

Knowledge-bases for resource-based configuration can be treated as a directed visualization graph $G^{(K)} = (V^{(K)}, E_c^{(K)}, \delta^{(K)})$ (see definition 1.4):

- $V^{(K)} = C \cup F$
- $E_c^{(K)} = \{(c, f) \mid c \in C, f \in F, f \in \text{offer}(c)\} \cup$
  $\{(f, c) \mid c \in C, f \in F, f \in \text{demand}(c)\}$
- $\delta^{(K)} : V^{(K)} \to \{\text{“}plugin\text{”}, \text{“}slot\text{”}, \text{“}virtual\text{”}, \text{“}presumed\text{”}, \text{“ ”}\}$
  where $\delta^{(K)}(v) = \text{“ ”} \Leftrightarrow v \in C$. $\delta^{(K)}$ provides additional information about functionalities: “Virtual” and “presumed” functionalities do not, like “slot” or “plugin” functionalities, correspond to physical connections between components.

Obviously, $G^{(K)}$ is bipartite.

Experts are mainly interested in causal effects between components, i.e. they want to know if the existence of a component c_i in S directly causes the existence of a different component c_j in S. Therefore it makes sense, not to visualize $G^{(K)}$ but a graph $G^{(C)}$ which models components and the causal effects between them.

$G^C = (V^{(C)}, E_c^{(C)}, w^{(C)})$ is constructed as follows:

- $V^{(C)} = C$
- $E_c^{(C)} = \{\ (c_i, c_j) \mid c_i, c_j \in C, \exists f \in F : (c_i, f), (f, c_j) \in E_c^{(K)}\}$
- $w^{(C)} : E_c^{(C)} \to \mathbf{R}$. Let $(c_i, f), (f, c_j) \in E_c^{(K)}$ be the edges which cause an edge $e \in E_c^{(C)}$. $w(e)$ depends on the importance of the type of the functionality f: $w(e) = 4$ for “slot” functionalities, $w(e) = 2$ for “plugin” and “virtual” functionalities, and $w(e) = 1$ for “presumed” functionalities.

  An additional function

  $$\begin{aligned} \kappa : E_c^{(C)} &\to \{\text{“}plugin\text{”}, \text{“}slot\text{”}, \text{“}virtual\text{”}, \text{“}presumed\text{”}, \text{“”}\}, \\ e &\mapsto \delta^{(K)}(f) \end{aligned}$$

  is used to label edges according to the type of the corresponding functionality.

[7] Introduction can be found in [64, 123].

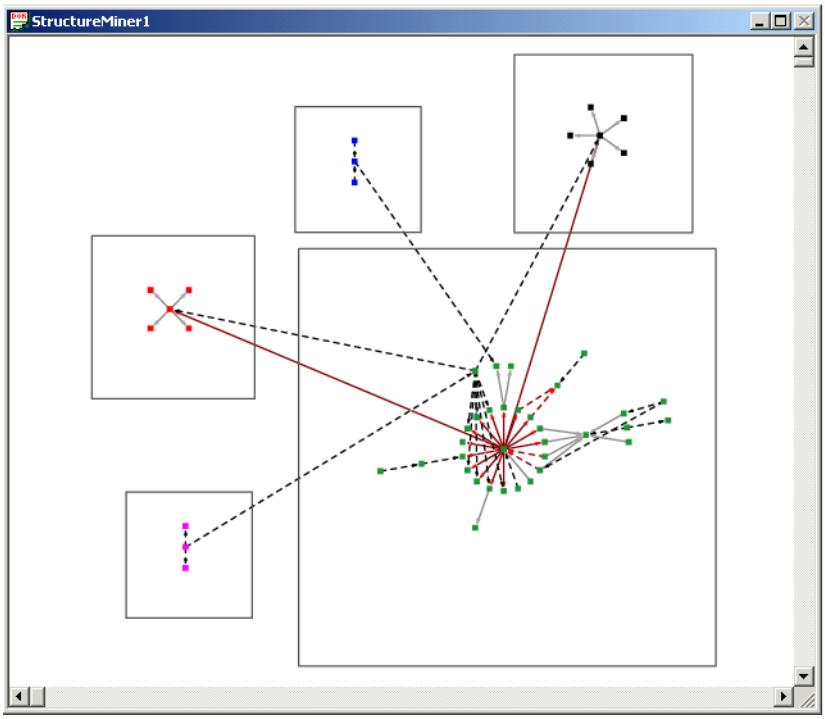

Figure 6.23: *A visualized knowledge-base for resource-based configuration.*

G^C is visualized using the methodology of structure-based visualization. Edges are displayed differently according to their κ value:

- "Slot" edges are represented by red and solid lines.
- "Plugin" edges use grey, thick, and solid lines.
- "Virtual" edges are displayed using red and dotted lines.
- "Presumed" edges are drawn using a dotted black pen.

STRUCTUREMINER is used to visualize such graphs. In this work, a knowledge base modeling a telecommunication system is used. This knowledge base comprises components such as cables, boxes, computer plug-in boards, telephone exchange systems, adapters, etc.

The system Pre-AKON, which has been developed in 1994 for the company Tenovis (formerly Bosch-Telenorma), uses this knowledge bases to support the synthesis of telecommunication system.

Figure 6.23 gives an example: First, MAJORCLUST is used to cluster the graph G^C. The clusters and nodes are visualized by means of a circuit layout method (see section 4.1.4 for details).

STRUCTUREMINER can blend-out edge types. Figure 6.24 gives an example: In the left screenshot all edges are shown; in the screenshot on the right, only "slot" and "plugin" edges are displayed.

Other graph drawing methods have also been applied: Figure 6.25 shows the result of a force-directed layout method (left side, see section 4.1.1 for details) and a random layout (right side).

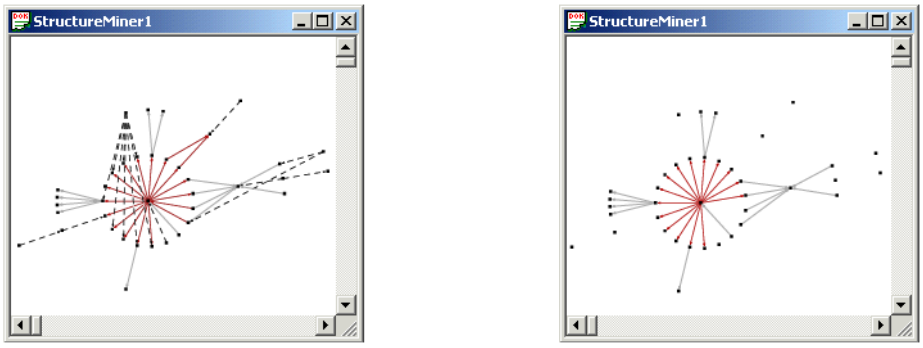

Figure 6.24: *Left side: One cluster of the knowledge-base, all edges are shown. Right side: Only the edge types "SLOT" and "PLUGIN" are shown.*

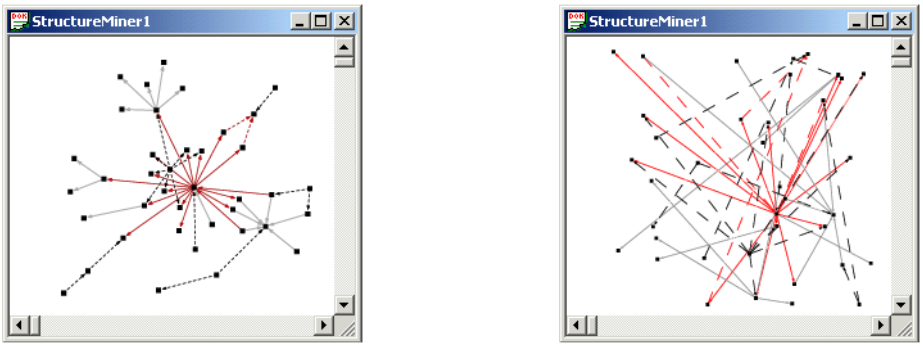

Figure 6.25: *The same cluster as in figure 6.24 visualized using a force-directed layout method (left side) and using a random layout (right side).*

6.2.1 Identifying Technical Subsystems

As described in chapter 3, it is often helpful to label clusters. For the knowledge-base used in this work, clusters should correspond to technical subsystems.

Knowledge-bases often come in different variations. E.g. for different telephone systems, slightly different knowledge-bases are used. Such knowledge bases share most of their components and functionalities. However, a few components normally offer or demand different functionalities. In such cases, it is helpful to recognize subsystems which are similar to already known subsystems. For this work, variations of the original knowledge base have been generated artificially. The variations are similar to variations as found between real problem instances. The original knowledge base comprises 5 technical subsystems.

As described in section 3.1, a classification function c is used to map for a cluster C_i from cluster features $\overrightarrow{f}(C_i)$ onto the cluster labels $\{l_1, \ldots, l_5\}$. Here, l_i denotes the name of a technical subsystem. I.e. $c(\overrightarrow{f}(C_i))$ denotes the correct cluster label for C_i. c is learned by analyzing sample cluster labels, details can be found in section 3.1 and in appendix A.

184 cluster are examined manually. 136 of these clusters correspond to one of original 5 subsystems. These 136 clusters are labeled using one of the 5 different labels (i.e. technical subsystems). 42 labeled clusters are chosen ran-

domly and used to learn[8] a classification function c which maps from clusters onto labels. Details concerning the learning process can be found in appendix A. The learned function c classifies incorrectly 16.7% of the clusters in the learning set.

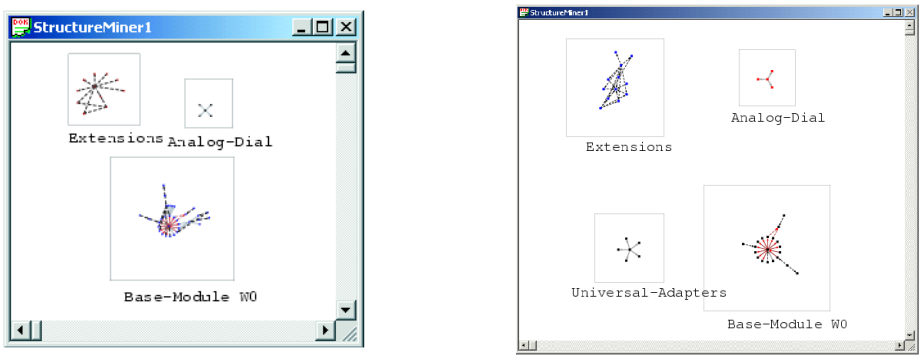

Figure 6.26: *Two different knowledge-bases; clusters are labeled, i.e. technical subsystem are identified*

Next, the other 94 labeled clusters are used to verify the learning results, i.e. these clusters form a test set: 15.8% of the clusters in the test are labeled incorrectly by c. Obviously, a classification function is learned by analyzing sample labelings. Figure 6.26 gives two examples: The clusters correspond to technical subsystems and are labeled automatically.

6.2.2 Classifying Clusters according to the Optimal Layout Algorithm

No single graph layout method proved optimal for all knowledge-bases or knowledge-base clusters. Therefore, STRUCTUREMINER supports the choice of individual layout methods for each graph and each cluster. In section 3.1.1, a method has been introduced that analyzes such choices and chooses appropriate layout methods for new graphs or clusters automatically.

This method calculates for each cluster C_i a set of features $\vec{f}(C_i)$ (see table 3.1). In this work, cardinal features are used, i.e. $\vec{f}(C_i) \in \mathbf{R}^*$. As described in appendix A, regression can be use to learn a classification function $c : \mathbf{R}^* \rightarrow \{l_1, \ldots, l_p\}$ where $l_i, 1 \leq i \leq p$ denotes a layout method. Thus, c maps from features onto graph drawing algorithms.

An example: The author has chosen his favorite graph drawing method for 75 clusters. 48 of these clusters has been used to learn a classification function[9] The set of classified clusters is divided into a learning set comprising 48 clusters and a test set of 27 clusters[10]. The learning set is used to learn c. The purpose of the test set is to evaluate the generalization capabilities of c. In this example, 4.7% of the cases in the learning set and 11.9% of the clusters in the test set are

[8] Regression is used as a learning method.

[9] The feature x from appendix A correspond the features $\vec{f}(C_i)$.

[10] As usually, several such apportionments are used; the variance of the error rate has been 0.0076 for the test set. The layout methods have been represented in each apportionments by an equal number of cases.

classified incorrectly. Obviously, it was possible to learn the author's layout preferences.

Examples can be seen in figures 3.1 and 6.27.

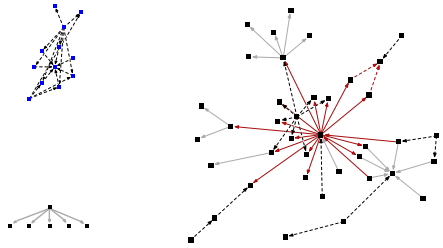

Figure 6.27: *Different layout methods have been chosen automatically for different clusters (circular layout for the top-left cluster, level-oriented layout for the bottom-left cluster, and force-directed layout for the right cluster).*

6.2.3 Discussion

The visualization method presented in this section helps users to comprehend complex knowledge bases quickly. This is done by means of the following visualization features:

- MAJORCLUST is able to find technical subsystems.
- Labels identify clusters as special technical subsystems. By analyzing sample labelings, a function, which labels clusters automatically, is learned successfully.
- Individual layout methods are applied to different clusters. This is done by analyzing previous user preferences. Users mainly chose force-directed layout methods, all-pair factor-drawing, and circular drawing.

6.3 Protein Interaction Graphs

After the completion of the human genome project and because the complete sequences of many other organisms are available these days, it becomes increasingly important to identify the biological function of the novel genes found. Since proteins are the products of genes, researchers work on the characterisation of the Proteome, the set of all proteins in a living cell. In order to understand how proteins work together and build the complex machinery of living cells, experimental and computational studies of Protein-Protein Interactions are central to enhance our understanding of cellular processes as a whole.

The reader may note that in this work, using publicly available sources, protein interactions are taken for granted. The various underlying biological principles are not examined (e.g. enzymatic reactions, protein complex assembly, gene regulation, signal transduction, see [84]). When protein interactions are understood, it is often possible to prevent or influence a sequence of biological events (e.g. disease states).

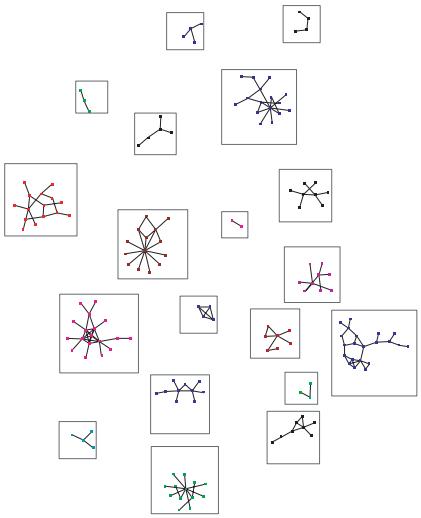

Figure 6.28: *A typical protein-network visualized using* MAJORCLUST *and factor-based layout, no inter-cluster connections are shown.*

Introducing even the basic underlying concepts of protein analysis is beyond the scope of this work. Biological questions have not been the focus of research of the author[11]. Introductions to protein analysis can be found in [84, 132, 60]. This section therefore treats proteins as nodes of a so-called protein network. Edges represent interactions between proteins. In some cases, nodes do not model a single protein, but a whole family of proteins. In this case, edges between protein families summarize all interactions between single proteins and are therefore weighted by the amount of original interactions. For such analyses, existing protein taxonomies (e.g. the SCOP hierarchy, see also [150]) are used.

The methodology of structure-based visualization is applied to protein networks. The results are presented and discussed at the end of this section.

As mentioned before, biologists classify proteins by means of taxonomies. Taxonomies are hierarchical clusterings (see definition 2.0.5). In this work, the SCOP hierarchy is used which describes proteins by the categories “Class,” “Fold,” “Superfamily,” “Family,” and “Protein”. At the top level of the hierarchical clustering, proteins are differentiate by their membership to a special class, the category “Fold” is used on the second level, etc.

In order to obtain a notion of the degree of interactions within and between clusters, it is helpful to compare the clusterings as defined by the SCOP hierarchy (clustering using domain knowledge) and the clusterings computed by one of the methods from chapter 2 (graph-theoretic clustering). The reader may note that graph-theoretic clustering tries to find a clustering which maximizes the number of edges within clusters and which minimizes the number of edges between clusters. The following text defines a so-called clustering comparison coefficient χ which can be used for arbitrary clustering comparisons. χ is used later on to compare the results of SCOP-clustering and MAJORCLUST respectively.

[11] Those merits must be assigned to Michael Lappe from the Structural Genomics Group of the European Molecular Biology Laboratory — European Bioinformatics Institute in Cambridge (UK).

Definition 6.3.1 (Routing Distance in a Hierarchical Clustering)
Let $G = (V, E_c)$ be a visualization graph. $T = (C, E_T, C_0)$ denotes a hierarchical clustering of G (see definition 2.0.5). The routing distance $dist_r$ between two nodes $v_1, v_2 \in V$ is defined as follows:

$$dist_r(v_1, v_2) = \begin{cases} 0 & v_1, v_2 \text{ are in the same cluster } C_i \in C \\ \frac{dist(C_i, C_j)}{2} & v_1 \in C_i, v_2 \in C_j, i \neq j \end{cases}$$

where $dist$ denotes the length of the shortest path between two nodes in the tree T and is formally defined by definition B.0.4.

Definition 6.3.2 (Clustering Comparison Coefficient χ)
Let $G = (V, E_c)$ be a visualization graph. $T^{(1)} = (C^{(1)}, E^{(1)}, C_0^{(1)})$ and $T^{(2)} = (C^{(2)}, E^{(2)}, C_0^{(2)})$ denote two hierarchical clusterings of G (see definition 2.0.5). $\chi(G, T^{(1)}, T^{(2)})$ is defined as follows:

$$\chi(G, T^{(1)}, T^{(2)}) = \frac{\sum_{v_1, v_2 \in V} |dist_r^{(1)}(v_1, v_2) - dist_r^{(2)}(v_1, v_2)|}{\mu},$$

where $\mu = (|V|^2/2 - |V|) \cdot h$ denotes a normalization factor. h stands for the maximum of the heights of the trees $T^{(1)}$ and $T^{(2)}$. $dist_r^{(i)}(v_1, v_2)$ denotes the routing distance of v_1, v_2 in $T^{(i)}$ (see definition 6.3.1).

$\chi(G, T^{(1)}, T^{(2)})$ examines all node pairs of V. Some node pairs may be neighbours in $T^{(1)}$, but belong to different clusters of $T^{(2)}$. χ assesses the differences of distances in $T^{(1)}$ and $T^{(2)}$ respectively. χ is normalized, i.e. all values are in the interval $[0, 1]$. A value of 0 means that the clusterings are equal.

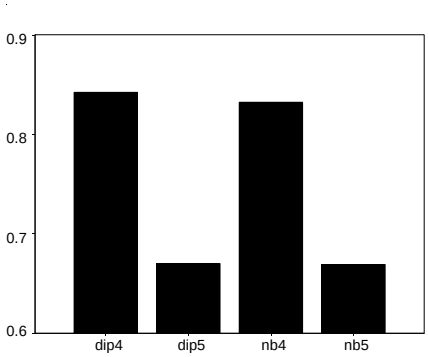

Figure 6.29: *The clustering comparison coefficient χ for different protein networks.*

Figure 6.29 shows $\chi(G, T^{(1)}, T^{(2)})$-values for 4 protein networks. $T^{(2)}$ is defined by the SCOP-hierarchy, $T^{(1)}$ has been computed using MAJORCLUST (see section 2.2.1). All values are between 0.66 and 0.85. Thus, the clusters defined by MAJORCLUST differ significantly from the SCOP-hierarchy. With other words, protein interactions happen mostly between SCOP-clusters.

6.3.1 Discussion

The system STRUCTUREMINER is used to visualize 20 different protein networks[12], 60% of which represent protein families. Those graphs comprise 300-

[12] Because protein networks are rather valuable, only a small number of such graphs is available.

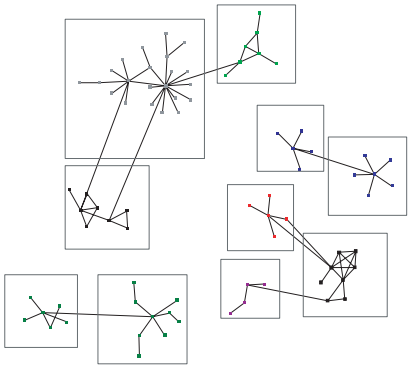

Figure 6.30: *A visualized protein network including inter-cluster connections.*

500 proteins; most graphs have an edge density (see section 2.1.3) of $\approx 5 - 12$. Figure 6.28 shows a part of a typical protein network: MAJORCLUST has been used as a clustering algorithm. Nodes and clusters have been visualized by the all-pair factor-based drawing method (combined with an incremental spring-embedder). Figure 6.30 shows some inter-cluster interactions.

Only MAJORCLUST is used as a cluster algorithm since evaluating cluster qualities is time-consuming and can only be done by experts who have limited time.

Figure 6.31: *The same cluster of a protein network visualized by a spring-embedder (left side, Tunkelang's spring-embedder has been used, see 4.1.1 for details) and by all-pair factor-based drawing (right hand side).*

Since no information about a graph's structure is known beforehand, only the following two layout methods are applied: (*i*) Force-directed drawing (section 4.1.1) and (*ii*) a combination of all-pair factor-based drawing and an incremental spring-embedder (see section 4.2.1). The second method normally produces for many cases the better results. Figures 6.31 and 6.32 show two examples; the reader may note that while Tunkelang's method produces more esthetic layouts, all-pair factor-based drawing leads to easier analyzable layouts. Force-directed drawing often fails to identify the underlying global graph structure.

Even for larger graphs ($\approx 400 - 500$ nodes, $\approx 4000 - 5000$ edges), MAJORCLUST is able to compute the clusters within ≈ 1 minute.

Experts find the combination of clustering methods and graph drawing techniques helpful for the analysis of protein networks. By analyzing inner cluster interactions first and by considering edges between clusters in a second step, hardly comprehendable graphs can be examined faster and more reliably.

The structure-based visualization of protein networks is still in its infancy.

Figure 6.32: *The same cluster of a protein network visualized by a spring-embedder (left side, Tunkelang's spring-embedder has been used, see 4.1.1 for details) and by all-pair factor-based drawing (right hand side).*

Nevertheless, the few results which have already been evaluated by experts, suggest the continuation of this research. Some open research issues are:

- ☞ Existing protein taxonomies should be integrated in structure-based visualization using node colors or node shapes. This would allow for a visual comparison of graph-theoretic and biological clusterings.
- ☞ Identified clusters should be labeled automatically. Even the manual classification of clusters (according to their biological function) currently constitutes a difficult problem. No automatic methods for such a cluster analysis are known so far.
- ☞ The visualization tool should be customized according to the special demands of biologists. This may include node shapes, edge thicknesses, or node and edge labels.

6.4 Fluidic Circuits

This section elaborates on the visualization of fluidic circuits. The main focus lies on the clustering step. Since fluidic circuits often have an almost serial-parallel structure, special layout algorithms can be applied (see [8] for details). These serial-parallel layout methods are not the subject of this work. More information about fluidic circuits can also be found in section 7.1.1.

6.4.1 Domain Clustering

In this section, some new answers to the following question are given: "Can a system learn a special, domain-dependent clustering method, if the user only gives some sample clusterings?". The method of definition by exemplary demands no explicit knowledge from the user and can help to bridge the gap between implicit expert knowledge and explicit algorithms.

Domain-dependent knowledge is expressed using node labels, i.e. unlike the clustering methods presented before, the node labels δ of a visualization graph $G = (V, E_c, E_v, \delta, w, \Delta)$ are used. The reader may note that arbitrary information can be coded into node labels δ, e.g. when modeling a computer network, labels may denote an IP-address or the function of the node (terminal, router, switch, WWW-server etc.).

The idea employed here is to map a graph G_d where domain-dependent knowledge is coded into the node labels onto a domain-independent graph G_i where all domain knowledge is coded into the edge distribution and the edge

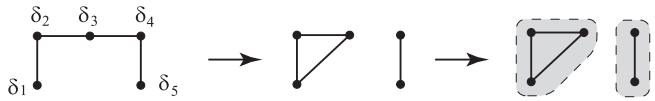

Figure 6.33: *Transfering a domain-dependent graph into a domain-independent one.*

weights. This graph can be clustered using the graph-theoretic techniques described in section 2. This is also shown in figure 6.33, in a first step the labeled graph G_d is transfered into a graph G_i which is only defined by nodes, edges, and edge weights. This graph is clustered in a second step. The remaining problem is to find a mapping between G_d and G_i, this step is called domain reduction.

Whether two nodes v_i, v_j of G_d belong to the same cluster, is defined by their labels and by others features of G_d, e.g. the distance between v_i and v_j in G_d. The next definition states this formally:

Definition 6.4.1 (Domain Reduction)
Let $G_d = (V^{(d)}, E_c^{(d)}, E_v^{(d)}, \delta^{(d)}, w^{(d)}, \Delta^{(d)})$ *be a visualization graph and let* $G_i = (V^{(i)}, E_c^{(i)}, w^{(i)})$ *be a graph without node labels.* G_i *results from* G_d *as follows:*

- $V^{(i)} = V^{(d)}$
- $E_c^{(i)} = \{\{v_1, v_2\} \mid v_1, v_2 \in V^{(i)}\}$
- $w^{(i)}(\{v_1, v_2\}) = like(\vec{f}(v_1, v_2))$

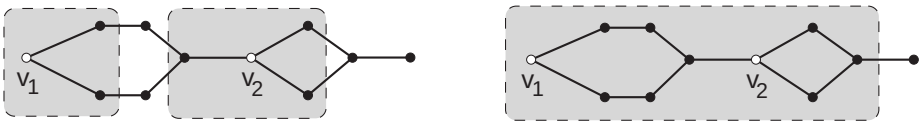

Figure 6.34: *A The local maximum flow for* $i = 1$ *(left-hand side) and for* $i = 2$ *(right-hand side).*

$\vec{f}(v_1, v_2)$ *denotes a function, which analyzes* v_1, v_2 *and their relation in* G_d *and compiles the result into a set of features* $(f_1, \ldots, f_p)$*. Typical features are:*

1. $\delta^{(d)}(v_1), \delta^{(d)}(v_2)$
2. *The distance*[13] $dist(v_1, v_2)$ *in* G_d*.*
3. *The maximum flow*[14] *between* v_1 *and* v_2 *in the subgraph of* G_d *induced by the nodes* $\Xi_i = \{v' \mid v' \in V^d, min(dist(v', v_1), dist(v', v_2)) < i\}$*. This so-called local maximum flow is normally calculated for* $1 \leq i \leq 5$ *and measures the connectivity between* v_1 *and* v_2 *in their neighbourhood. Figure 6.34 shows an example, on the left-side a graph and the subgraph as induced by* Ξ_1 *is shown (gray background), the local maximum flow between* v_1 *and* v_2 *is* 0*. On the right-side the subgraph induced by* Ξ_2 *is shown, now the local maximum flow between* v_1 *and* v_2 *is* 1*. The reader may note that the local maximum flow can be calculated quickly since the induced subgraph is relatively small.*

[13] For a definition of distance see definition B.0.4.
[14] For a definition of maximum flow see definition B.0.5.

4. Domain-specific features can be added.

$like$ is a function which takes the features $\vec{f}(v_1, v_2)$ and computes the likelihood of v_1 and v_2 being in the same cluster. Learning $like$ by analyzing given clusterings is the subject of the rest of this section.

In order to learn $like$, the user defines a set of clustered graphs $\{\{G_1, \{C_1^{(1)}, \ldots, C_{p_1}^{(1)}\}\}, \ldots, \{G_q, \{C_1^{(q)}, \ldots, C_{p_q}^{(q)}\}\}$ where $\{C_1^{(k)}, \ldots, C_p^{(k)}\}$,$p \in \mathbf{N}$ is a disjunctive clustering of G_k.

For every graph $G_k = (V^{(k)}, E^{(k)})$ and each pair of nodes $v_1, v_2 \in V^{(k)}$ we set $like(\vec{f}(v_1, v_2)) = 1 \Leftrightarrow \exists j \in \{1, \ldots, p_k\} : v_1, v_2 \in C_j^{(k)}$. For all other pair of nodes $v_1, v_2 \in V^{(k)}$ we set $like(\vec{f}(v_1, v_2)) = 0$.

Now a supervised learning strategy (e.g. neural networks or regression, see section A for details) can be used to learn $like$. This method is now applied to the clustering of hydraulic circuits:

Using such an adaptive method for the clustering of hydraulic circuits instead of a specially developed method as in section 6.4.1 has some advantages:

- ⊕ Developing a special clustering method takes a long time; almost a year has been necessary to develop the method presented in [149, 163]. Since not only the hydraulic axis, but also other types of cluster are needed for an analysis of hydraulic circuits, it makes sense to speed up the development process.

- ⊕ Experts often have problems expressing their implicit knowledge explicitly. A definition by example may help to bridge this knowledge-acquisition gap.

- ⊕ In some domains it is possible to get hold of examples, but an expert is hard to find. E.g. this is often true for medical domains.

Figure 6.35: *A manually clustered hydraulic circuit (left-hand side) and its structure (right-hand side).*

Figure 6.35 depicts a typical manual clustered hydraulic circuits: Two working units and a supply unit have been identified.

6.4.2 Discussion

In this example, 20 out of 62 circuits are clustered manually. As described above, for each pair of nodes v_1, v_2 a feature vector $\vec{f}(v_1, v_2)$ is computed. In addition to the general feature mentioned before, a domain-specific feature is

added: Each node is classified according to its function in the circuit (cylinder, tank, valve, pump, or miscellaneous). A function $like$, which maps from the features of two nodes onto their likelihood of being in the same cluster, is learned using a one-layer neural network and backpropagation (see section A for details).

Domain-independent graphs are constructed using both the learned function $like$ and the method of domain reduction (see definition 6.4.1). In a second step, these graphs are clustered using MAJORCLUST (see section 2.2.1).

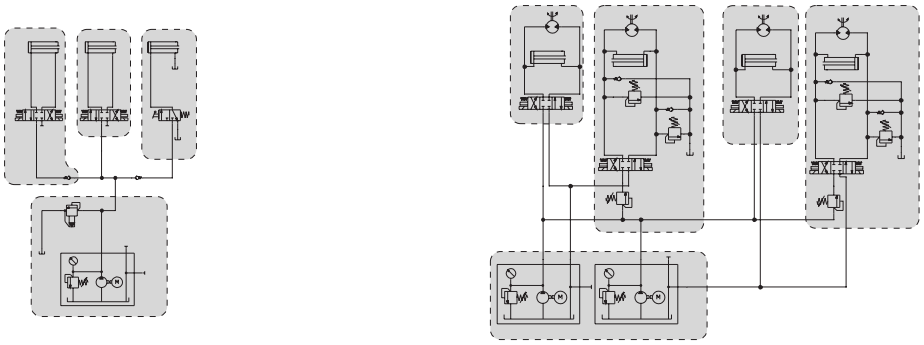

Figure 6.36: *Two automatically clustered Circuits*

Figure 6.36 shows two automatically clustered circuits; both clusterings are correct.

The method of domain reduction can correctly cluster 75% of the graphs in the learning set and 69% of the graphs in the test set. Obviously, it is possible to automatically learn a cluster method for hydraulic circuits by analyzing sample clusterings.

The method fails for circuits that comprise several small working units. Figure 6.37 shows a typical example; the five working units on the left are not recognized as single clusters, but they are identified incorrectly as one cluster.

The author thinks that it may be possible to apply this method to other technical domains as well, if the graphs from these domains fulfill certain requirements:

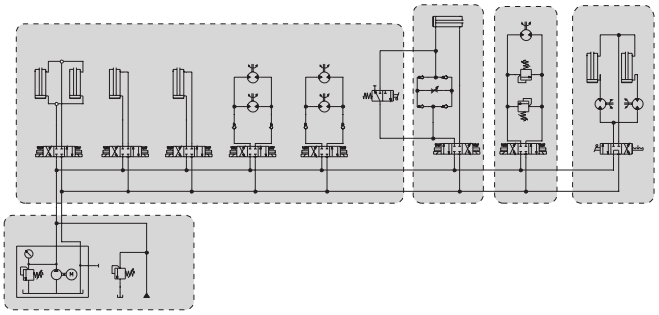

Figure 6.37: *An incorrectly clustered circuit*

- ☞ The node labels are given and define important domain knowledge.
- ☞ The wanted clusters can be defined using node labels and topological information only.

- ☞ All clusters are defined using the same set of instructions, i.e. the system can learn one set of instructions to identify all clusters.
- ☞ Clusters have at least a certain size.

Figure 6.38: *Two fluidic circuits visualized by a level-oriented layout. On the left side, components are depicted as nodes which emphasizes the structure of the circuit. On the right side, bitmaps are used to visualize single components.*

The domain of fluidic circuits is quite different from the other domains discussed in this work: Much knowledge about the domain exists and can be used to adjust the three steps of structure-based visualization:

- Step 1 has been discussed intensively.
- Classification, i.e. step 2, can be done quite easily: subgraphs which include cylinders are working units while supply units comprise tanks or pumps.
- As mentioned before, special layout methods exists for serial-parallel graphs. Level-oriented drawing (section 4.1.3) is also able to find reasonable layouts, figure 6.38 shows two examples.

This application is used mainly to show how domain knowledge can be used to improve clustering methods. The identification of substructures is a key aspect of the visualization of fluidic circuits. This example therefore underlines the importance of the concept of structure-based visualization.

Chapter 7

Visualizing Models given as Tables

Tabular data sets (see section 1.5.1) may be viewed as a table: rows define objects while columns correspond to object features. Such tables can be evaluated according to two different aspects: (*i*) Objects and their relations may be examined or (*ii*) the user may be interested in features and their relationship to one another. Section 7.1 deals with object analysis, while section 7.2 concentrates on the examination of feature relationships.

7.1 Object Visualization

In this section, an application for the visualization of objects, which are defined by tabular data sets, is presented. One focus is the usage of the knowledge-acquisition methods from section 5. A large variety of data is stored in the form of a table (databases, spreadsheets, etc.). Therefore, the techniques described in this section are also applicable to other applications.

7.1.1 Document Visualization

The visualization and retrieval of documents is subject to worldwide research because since the development of world spanning computer networks (Internet) exceeds the amount of accessible documents by far any administrable number. Typical documents are technical drawings, manuals, letters, or court decisions In order to allow for a rapid and correct access to all documents, visualization methods are used. For this, documents are placed on a plane in such a way that spatial closeness between documents corresponds to document similarities.

Such document management systems have been examined in several different works: In [183], a management system for arbitrary text documents is presented. This system extracts features from text documents and uses Multidimensional Scaling or Principle Component Analysis to visualize documents on a plane. In [22], a force-directed layout algorithm has been used to visualize bibliographic data in 3 dimensions. Self-organizing maps were employed in [106, 1]. Document clustering techniques were used in [187, 94, 1, 5]. The

proceedings of the International Conference on Information Knowledge Management (CIKM) or [15] may be used to obtain a more extensive overview.

The reader may note that the methodology of structure-based visualization allows for a separation between (*i*) general tasks (e.g. graph drawing) and (*ii*) specialized algorithms (e.g. Multidimensional Scaling), thus supporting the application of different standard algorithms to the problem of document visualization.

In this section, the visualization methodology presented in this work is applied to the retrieval of technical documents, i.e. to the visualization of a sets of fluidic circuits. Each fluidic circuit (e.g. hydraulic and pneumatic circuits) forms a node of a visualization graph, node pairs are connected by edges weighted with the appropriate document similarity. A short introduction to fluidic circuits is given in section 6.4.

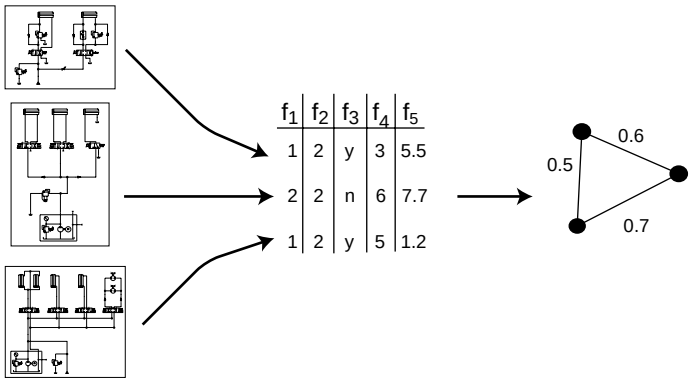

Figure 7.1: *Document Retrieval by Graph Visualization*

The main idea is to describe a given set of documents as a tabular data set (see section 1.5). Each row of this table consists of a set of features describing one document. This is also shown in figure 7.1: Each document on the left side forms a row of the table in the middle. The columns of this table correspond to features of fluidic circuits. Using the methods introduced in section 5, a graph is constructed. This graph is visualized using the method of structure-based visualization from section 1.3.

This section focuses on two main problems: (*i*) The identification of reasonable features for fluidic circuits and (*ii*) the application of the methods from section 5 to the automatic construction of similarity functions for fluidic circuits. A similarity function assesses the similarity between the features of two circuits. As mentioned before, this similarity function is used to define the edge weights of the visualization graph.
This process will now be described formally.

Formalization

Let $S = \{S_1, \ldots, S_n\}$ be a set of fluidic circuits. A tabular data set[1] $T_S = (V^T, E_c^T = \emptyset, E_v^T, \delta^T, w^T, \Delta^T)$ is constructed as follows:

- $V^T = \{v_1, \ldots, v_n\}$ where v_i represents the fluidic circuit S_i

[1] See definition 1.5.1

- $E_v^T = \emptyset$
- $\delta^T(v_i), v_i \in V^T$ is set of features describing the fluidic circuit S_i

In the following text, $v_i \in V^T$ will also be called a fluidic circuit.

As described in section 5, edges and edge weights have to be found in order to visualize a tabular data set. This is done by means of a so-called similarity function[2] $sim : (\Delta^T)^* \times (\Delta^T)^* \rightarrow [0,1]$ where $sim(\delta(v_i), \delta(v_j))$ denotes the similarity between the nodes v_i, v_j (i.e. fluidic circuits S_i and S_j).

Using this function sim, a visualization graph G_S can be constructed as follows: Let $T_S = (V^T, E_c^T = \emptyset, E_v^T, \delta^T, w^T, \Delta^T)$ be the tabular data set as defined above and let sim be a similarity function defined on T_S. Then a visualization graph $G_S = (V, E_c \neq \emptyset, E_v, \delta, w, \Delta)$ can be constructed as follows:

- $V = V^T$
- $E_c = \{ \{v_1, v_2\} \,| v_1, v_2 \in V \}$
- $w(v_i, v_j) = sim(\delta(v_i), \delta(v_j)), v_i, v_j \in V$
- $E_v = \emptyset$
- $\delta(v_i, v_j) = \delta^T(v_i, v_j), v_i, v_j \in V$

In the rest of this section, appropriate functions δ and sim will be introduced. First, a method is described which computes for each circuit a feature list δ.

Finding Features for a Fluidic Circuit

Engineers normally do not look at fluidic circuits from a behaviour point of view but instead employ a functional model of the circuit. Or, in other words, they are more interested in what a circuits does than in how it fulfills its function. When talking about circuit similarity, we normally mean circuits with similar functions. This is mainly caused by the problem-oriented approach of engineers: An existing plan is normally retrieved in order to help solve a new given problem; new problems are defined as a set of wanted functions.

Therefore, it make sense to reduce a circuit to a set a of features describing its function. This feature set can then be used to define a similarity function. In this section, only a short introduction to the feature definition of fluidic circuits is given, more extensive overviews can be found in [67, 161].

Fluidic circuits comprise functional units, so-called fluidic axes. The core of such an axis is a working element, i.e. a cylinder. From an engineer's perspective, every cylinder, i.e. every axes, implements a function. Typical examples are a cylinder raising a car or a cylinder locking a door. Thus, the functions of a circuit are defined by its fluidic axes.

$\delta(v_i)$ is defined as the combination of the features of the corresponding axes:

$$\delta(v_i) = \bigcup_{a_i \in A_i} \delta_{axis}(a_i),$$

where A_i denotes the fluidic axes of v_i. $\delta_{axis}(a_i)$ is a function which defines a set of feature describing a fluidic axes a_i.

[2] An introduction to similarity functions can be found in chapter 5.

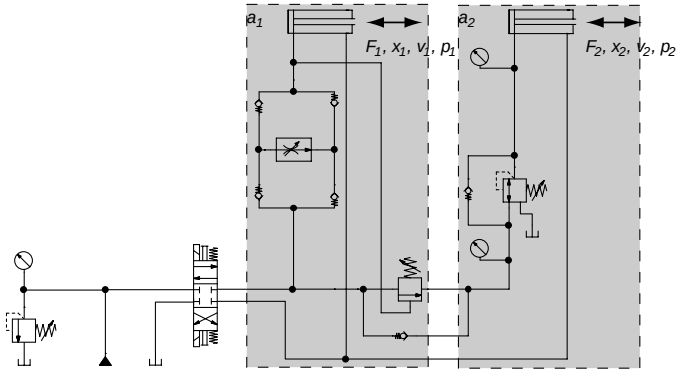

Figure 7.2: *A fluidic circuit with two axes a_1 and a_2. Each axes is described by the parameters force, position, velocity, and pressure of the corresponding working element.*

Axes are mainly described by their working element, i.e. in terms of force, positions, velocities, and pressure. This can also be seen in figure 7.2.

A working element can be specified by its position diagram: This diagram shows the cylinder position over a period of time. An example can be seen in figure 7.3: The working element of the axis a_1 is first extended. Afterwards, it is retracted. In a third time interval, it stays in a retracted position. Obviously, it is possible to identify three so-called phases: drive-out, drive-in, and null-phase. Since each phase implements a function (e.g. drive-out may correspond to the raising of a car), it makes sense to describe axes by their corresponding phases.

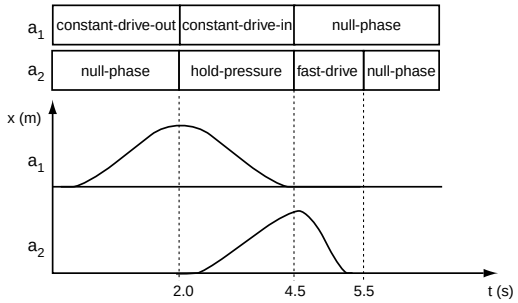

Figure 7.3: *Position diagram of two fluidic axes a_1 and a_2*

Each axis is described by two types of features, (*i*) phase descriptions and (*ii*) phase orders. These groups of features are defined as follows:

(*i*) *Phase Descriptions*
Phases are classified into the categories *constant-drive*, *position-drive*, *hold-position*, *accelerate*, *fast-drive*, *hold-pressure*, and *press*; each category in turn is characterized by 6 features:

1. How many phases of the specific category exist in the respective axis?
2. How long (in seconds) is the duration of the phase ?
3. Which distance (in mm) is covered by the working element?

4. Which force (in Newton) is applied to the working element?
5. How precise must the axis work? This is a value from $[0; 1]$ that defines the acceptable deviations from the duration, the distance, and the force.

(*ii*) *Phase Orders* These 49 features $f^{\text{ord}}_{i,j}, 1 \leq i, j \leq 7$ capture the order of the phases. For example, $f^{\text{ord}}_{1,2}$ is the number of times a phase of category 1 (*constant-drive*) is directly followed by a phase of category 2 (*position-drive*).

Together, the feature vector $\delta_{axes}(a)$ for an axis a is of the following form:

$$f(A) = \begin{pmatrix} (\text{phase description } \textit{constant-drive}) \\ (\text{phase description } \textit{position-drive}) \\ (\text{phase description } \textit{hold-position}) \\ (\text{phase description } \textit{accelerate}) \\ (\text{phase description } \textit{fast-drive}) \\ (\text{phase description } \textit{hold-pressure}) \\ (\text{phase description } \textit{press}) \\ (\text{phase orders}) \end{pmatrix}$$

Since the features of an fluidic axes and of fluidic circuits are now formally defined, the similarity functions will be learned in the next section.

Learning a Similarity Function for Fluidic Circuits

Finding similarities between fluidic circuits poses a knowledge acquisition problem, because experts are rare, expensive, and have their necessary knowledge only implicitly at disposal. When asked to express their knowledge about axes similarities explicitly as a mathematical function, most experts are overstrained.

Below, two methods from section 5 that tackle the knowledge acquisition problem are presented. The methods differ from each other in the explicity of the necessary expert knowledge. The following text assumes that the reader is familiar with the knowledge acquisition methods introduced in section 5.2.2. First, a similarity function for fluidic axes is presented.

Using the definitions from above, the similarity between two nodes v_i, v_j (representing circuits S_i, S_j) is defined as:

$$sim(\delta(v_i), \delta(v_j)) = \sum_{a_i \in A_i} \max\{sim_{axes}(\delta_{axis}(a_i), \delta_{axis}(a_j)) \mid a_j \in A_j\},$$

where A_i denotes the axes of S_i, A_j denotes the axes of S_j, $sim_{axes}(\delta_{axis}(a_i), \delta_{axis}(a_j))$ denotes a function measuring the similarity between the features $\delta_{axis}(a_i)$ and $\delta_{axis}(a_j)$, and $|A_i| \leq |A_j|$ holds.

Defining a similarity function $sim(\delta(v_i), \delta(v_j))$ between two fluidic circuits v_i and v_j has now been reduced to the problem of defining a similarity function $sim_{axes}(\delta_{axis}(a_i), \delta_{axis}(a_j))$ between two fluidic axes a_i and a_j.

The similarity between axes is defined using the Weighted Linear Similarity Function of Interaction Level 1 (see example 5.1.1):
Let a_i and a_j be two fluidic axes and let $\delta_{axis}(a_i) = (f^i_1, \ldots, f^i_p)$ and $\delta_{axis}(a_j) =$

$(f_1^j, \ldots, f_p^j)$ be the corresponding feature vectors. $sim_{axes}(\delta_{axis}(a_i), \delta_{axis}(a_j))$ is then defined as follows:

$$sim_{axes}(\delta_{axis}(a_i), \delta_{axis}(a_j)) = \sum_{1 \leq k \leq p} w_k \cdot |f_k^i - f_k^j|,$$

with $w_i \in \mathbf{R}$.

Two knowledge-acquisition methods from section 5.2.2 are applied to find values for the weights w_i. As mentioned before, the so-learned similarity measure is then used to construct a visualization graph.

Knowledge Acquisition Method 1: Learning by Classification

As described in section 5.2.2, one way to obtain the necessary expert knowledge is to exploit a classification of the fluidic circuit. This knowledge can be used to learn values for the weights w_i.

67 fluidic axes are classified into 9 classes by a domain expert. Using regression[3] , a similarity measure is constructed. The error rate is defined as the percentage of axes pairs whose similarity is assessed incorrectly by the learned measure. The error rate on the learning set[4] is 12%, while the error rate on a test set[5] is 16%. Obviously, a good similarity measure is constructed.

Knowledge Acquisition Method 2: Learning by Examples

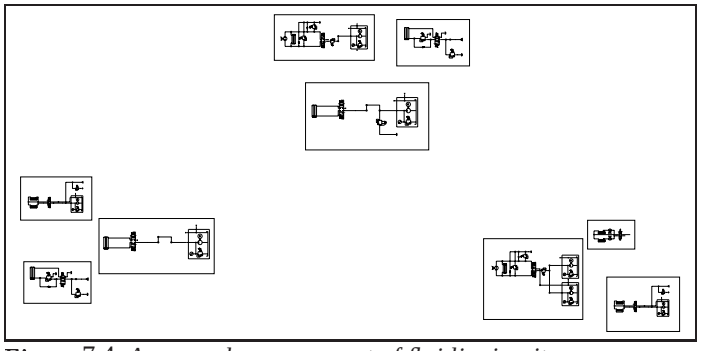

Figure 7.4: *A manual arrangement of fluidic circuits*

A graphical arrangement of circuit documents, which has been generated by the domain expert, is analyzed and similarity measures are constructed [6] . An example can be seen in figure 7.4. To evaluate the quality of the learned measure, the Mean Square Error (MSE) is used:

$$\frac{\sum_{A_i, A_j} (sim^t(A_i, A_j) - sim^e(A_i, A_j))^2}{m},$$

[3]For details concerning the following characteristics of the regression result please refer to section A.2: 33 features are used, 40 % of them represent phase orders, $R^2 = 0.482$, significance of the F-test= 0.0

[4]The learning set comprised axes pairs used for the learning process.

[5]The axes pairs in the test set have not been used for the learning process.

[6]For details concerning the following characteristics of the regression result please refer to section A.2: 16 features are used, 51 % of them represented phase orders, $R^2 = 0.2$, significance of the F-test= 0.001

A_i and A_j denote fluidic axes, m denotes the number of axes, sim^t denotes the similarity as predicted by the learned similarity measure, and sim^e denotes the empirical similarity measure defined by the manual layout.

On a test set, a MSE of 0.045 is achieved. All similarity values are from the interval $[0, 1]$, hence the MSE defines a (squared) average deviation from the correct similarity.

Only 16 features are used for the regression; all other features have constant values. 51 % of them represent phase orders. Therefore, both phase orders and phase descriptions are important. The most important features of the phase descriptions are precision, distance, and number of phases. Especially, the phase orders "constant-drive after constant-drive" and "position-drive after hold-position" are relevant.

The significance of the regression result[7] is 0.001; meaning that the result is not caused by random influences.

The R^2 value of the regression (measuring the amount of variation explained by the regression) is 0.2 which can be interpreted as the statement that the regression result can explain the observations only partially; i.e. there remain unexplained observations. In order to improve this result, a more complex similarity function is needed. Experts assume that dependencies between the features of an axes exist. Therefore, the Weighted Linear Similarity Function of Interaction Level 2 (see example 5.1.2) is used:

$$sim_{axes}(\delta_{axis}(a_i), \delta_{axis}(a_j)) = \sum_{1 \leq k \leq p} w_k \cdot |f_k^i - f_k^j| + \sum_{1 \leq k_1 \leq p} \sum_{1 \leq k_2 \leq p} w_{k_1,k_2} \cdot |f_{k_1}^i - f_{k_1}^j| \cdot |f_{k_2}^i - f_{k_2}^j|,$$

with $w_k, w_{k_1,k_2} \in \mathbf{R}$.

Applying regression to this function results in an MSE of 0.029. The R^2 value can be improved to 0.49. Hence this complex similarity measure is able to capture the dependencies between features.

I. e., also with this knowledge source a good similarity measure can be constructed.

Visualizing Fluidic Circuits

As described earlier, the constructed similarity measure can be used to transform a tabular data set into a visualization graph. While only a few circuits are normally used for the knowledge acquisition process, large numbers of circuits can be involved in the visualization process. Of course, as with all learning processes, the fluidic circuits used for the learning set must be representative. This problem can be solved by an expert who is normally able to choose an appropriate set of plans.

A modified version of STRUCTUREMINER (see section 6) is used to visualize circuits. Figure 7.5 shows a typical visualization and each plan is represented by its name. A different visualization mode can be seen in figure 7.6 where the circuits are now shown as graphs. The positioning and especially the grouping of circuits allows the user the assess similar documents quite easily. E.g. when the user knows the positions of two matching circuits, other circuits between them might also be interesting.

[7] Using an F-Test.

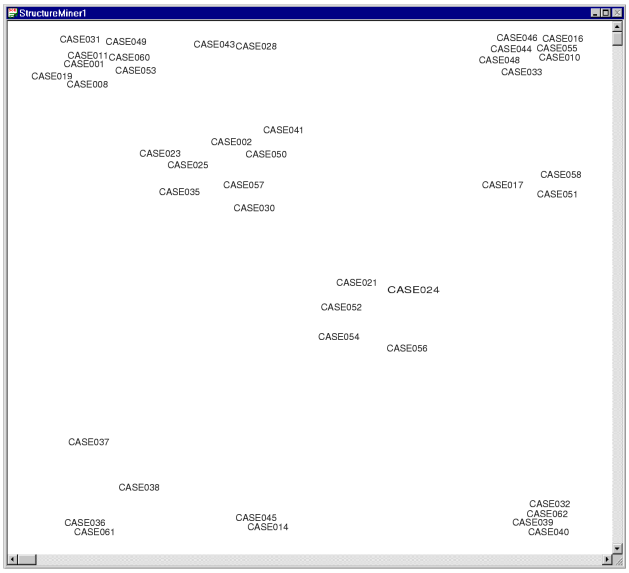

Figure 7.5: *Visualized Fluidic Circuits, textual display mode*

Parenthesis: Such a visualization can easily be used to support the design of fluidic circuits: Experts define the expectations of a to-be-designed circuit in form of a functional description, i.e. the expert gives a (partial) feature vector of the wanted circuits. This feature vector is visualized together with several known circuits. All existing circuits close to the new feature vector resemble the problem description and can therefore be used to solve the design problem. If no visualization is used and only the closest circuits are retrieved, this solution is nothing but a normal Case-Based Reasoning approach.

Discussion

This section applies the knowledge-acquisition methods from chapter 5 to the domain of document management. First, abstract features for fluidic documents are developed. In a second step, a similarity measure working on these features is learned by means of different knowledge acquisition methods. The results presented above show that reasonable similarity measures are found. In the final step, a given set of documents is transformed into a graph using this similarity measure. This graph is visualized using the method of structure-based visualization from section 1.3.

Since, as outlined above, high quality measures can be learned, the resulting layouts successfully reproduce the user's notion of document similarity. Experts (e.g. in [161]) find the introduced knowledge acquisition and visualization methods to be both helpful for the domain of fluidic circuits and for the general problem of document management.

This approach has several advantages when compared to existing methods:

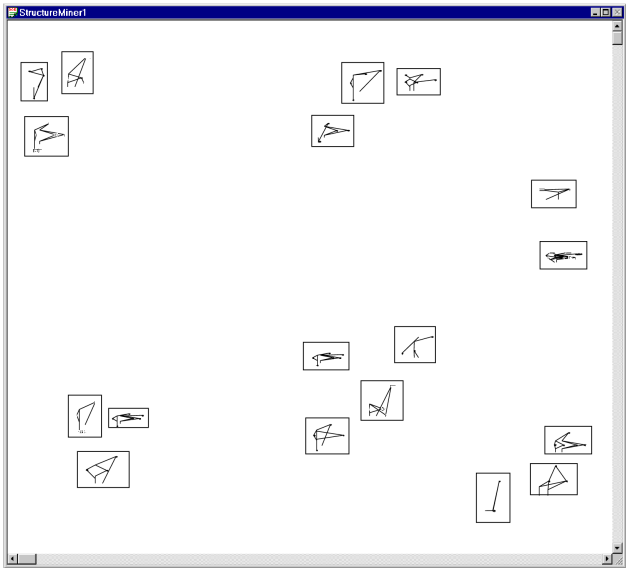

Figure 7.6: *Visualized Fluidic Circuits, graphical display mode*

⊕ The new knowledge-acquisition methods allow for a fast and reliable logging of expert knowledge.

⊕ Structure-based visualization emphasizes the underlying structure, thus supporting the fast access to large sets of documents.

⊕ Users can understand sets of unknown (fluidic) documents relatively quickly.

⊕ The visualization can support the design of fluidic circuits.

Many problems remain, e.g.:

- This approach to document management should be applied to different types of documents, e.g. text documents. Such documents demand the creation of different features.
- Further, already existing document management methods, e.g. taxonomies, search algorithms, can be combined with the methods presented here.

7.2 Visualization of Features

Tabular data sets (see definition 1.5.1) are normally depicted as a table. Rows define objects while columns correspond object features. Users are often interested in dependencies between features. E.g. table 1.1 describes persons by the features name, education, income, occupation, and gender. Typical question formulations are whether the gender influences the income or whether the education depends on the gender and on the citizenship.

In this section, features of tabular data sets (see section 1.5.1) are analysed and visualized. The main idea is to treat features (the columns of a tabular

data set) as nodes of a visualization graph (see definition 1.4.1). Each pair of nodes is connected by an edge which is weighted by the dependency between the corresponding features. Feature dependencies express an undirected and unsigned causality, i.e. two features f_1, f_2 depend on each other if values of f_1 (f_2) cause f_2 (f_1) to assume special values. Examples are age and the number of children (the older a person is, the more children she / he has) or income and type of car. The causalities may be probabilistic and may comprise more than just two features.

Parenthesis: Of course, feature dependencies can also be computed using the methods from chapter 5: Features become objects, while object descriptions become features[a], i.e. the table is rotated at 90 degrees. Using the terminology from chapter 5, in this section the similarities are defined implicitly by the given feature values, i.e. no explicit similarity function is used and the object similarity category is specified by the given values. Another difference exists: Features only comprise values of the same type (nominal, cardinal etc.). This allows for the application of special statistical methods.

[a] Nevertheless, the author will refrain from calling features objects and vice versa.

Since the dependencies are not known beforehand, but constitute the goal of the analysis, they must be induced from the tabular data sets by statistical means. Typical techniques can be classified according to the type of the features (see also section 5.1):

- **Nominal features:** Two main classes of algorithms can be identified: (*i*) classical statistical approaches and (*ii*) association rules. The reader may note that association rules are also based on statistical computations; the main different is rather that association rules (introductions can be found in [2, 3, 13]) have been developed in the field of computer science with a different focus than classical statistical approaches (see [6, 62]).

  Association rules use a two-step method: In a first step, promising candidate sets of dependent variables are identified. The second steps generates rules using these candidate sets. Here, only the first step is needed. Let n denotes the number of objects and p the maximum number of dependent features[8]. If only dependencies between pairs of features are wanted, the association rules method needs $O(n)$ time, otherwise its runtime behaviour is exponential in p. Association rules are a relatively fast method for identifying dependencies between larger sets of variables (larger p) but, since this method is based on heuristic decisions, it is difficult to assess the quality of the results.

  A common method for analysing the dependency between nominal features in the field of statistics is the Ξ^2-test ([6, 62]). This method allows for a quantitative assessment of feature dependencies. Dependencies between pairs of features can be computed in $O(n)$ time, for all other cases

[8] For reasons of simplicity, binary variables are assumed.

the run-time is exponential in p. For $p > 2$, the $log - linear$-analysis ([62]) is normally applied.

- **Cardinal features:** Regression (see section A) is normally used to compute (linear) dependencies between cardinal features. The coefficient of determination expresses the degree of dependency. An alternative is a transformation into nominal features by introducing intervals.

- **Mixed cardinal and nominal features:** Variance-analysis and discrimination-analysis (see [6, 62] for details) are well-known techniques to analyse such cases. Here, a different approach is used: Let f_1 be a nominal feature with q different values $\eta_1, \ldots, \eta_q$ and f_2 be a cardinal feature. f_1 is transformed into q new feature $f_1^1, \ldots, f_1^q$ with $f_1^j(v) = 1 \Leftrightarrow f_1(v) = \eta_j$ for all objects v and values η_j. f_1^j can be treated as a cardinal feature, thus the dependency between f_1 and f_2 can be computed by applying regression to the pairs f_1^j and f_2 ($1 \leq j \leq$q).

Parenthesis: It is also helpful to combine association rules and classical statistical approaches: The association rule algorithm is used to identify promising sets of dependent variables. Classical approaches are than used to analyze the degree of dependency between variables.

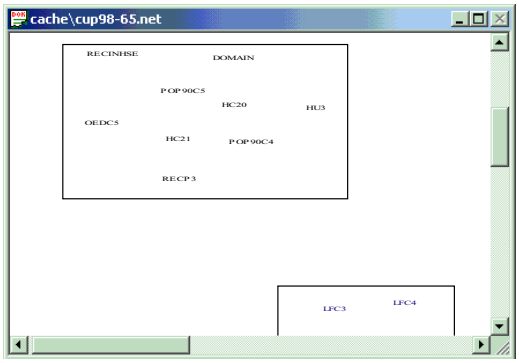

Figure 7.7: *A typical visualized cluster (using* MAJORCLUST *and factor-based drawing), all nodes are related to middle-class houses.*

This method is now applied to the analysis of large groups of people. Every person is described by 481 features, including personal information (age, number of children, etc.), information about the neighbourhood (average income, percentage of married people, etc.), and hobbies (pets, gardening, etc.). This extensive data set and further information about the features can be found in [89].

Here, only pair-wise dependencies between features of a group of 1000 people are analyzed and visualized. If both features are nominal, a Ξ^2-test is done and the Cramer's V value (see [6] for details) is used as the corresponding edge weight. Pure cardinal features are analyzed using regression and mixed feature pairs are treated as explained above.

Discussion

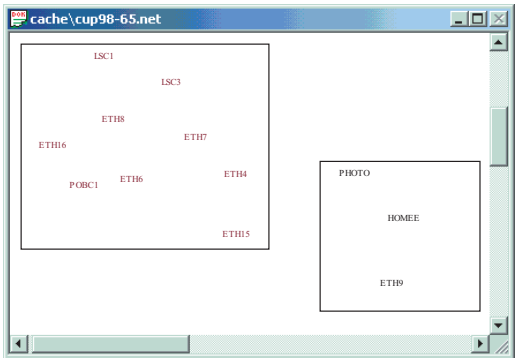

Figure 7.8: *Two visualized clusters (using* MAJORCLUST *and factor-based drawing). The left cluster shows features related to people who were born outside the U.S.A. or who speak a non-english language. The cluster on the right relates for unknown reasons the hobby photography, people who work at home, and people from the Philippines.*

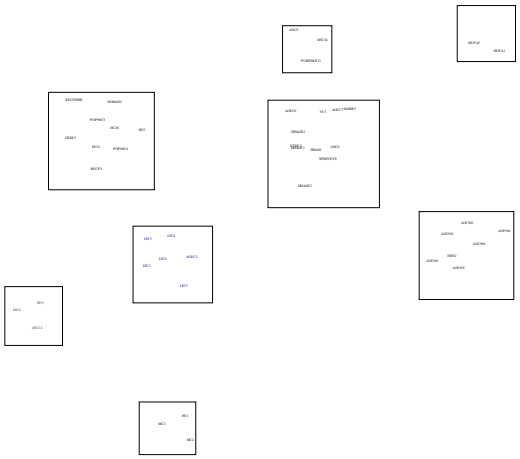

Figure 7.9: *A clustered and visualized set of features.*

MAJORCLUST (see section 2.2.1) identifies 20 clusters. 13 of them are intuitively meaningful; figures 7.7 and 7.8 show three typical clusters. Figure 7.9 depicts a part of the overall layout of the visualized feature graph. Hierarchical MAJORCLUST even identifies 50 clusters on 3 hierarchy levels; at least 30 of them are intuitively meaningful. Kohonen-Clustering (section 2.1.4) identifies 22 clusters, most of them similar to the clusters found by MAJORCLUST. MinCut-Clustering (section 2.1.2) finds 12 clusters, 11 containing $1-5$ nodes and one cluster comprising all other nodes. While MAJORCLUST finds the clusters within ≈ 40 seconds, Kohonen-Clustering needs ≈ 6 minutes and MinCut-Clustering needs about ≈ 50 minutes. Therefore, the author advocates the usage of MAJORCLUST for these type of graphs.

Since nothing is known about feature graphs beforehand, only general graph layout method should be applied. The author prefers all-pair factor-based layout combined with an incremental spring-embedder (see section 4.2),

since the overall structure of the graph is emphazised (see also figure 7.10). Force-directed methods (section 4.1.1) fail to produce a reasonable global layout. The run-time behaviour of all-pair factor-based layout and force-directed layout is comparable.

Figure 7.10: *Left side: A part of a feature graph visualized by means of all-pair factor-based layout. Right side: The same graph visualized with Tunkelang's version of force-directed layout.*

Several open research issues remain:

- ☞ More datasets should be used.
- ☞ An extensive comparison of the different methods for the identification of feature dependencies is needed.
- ☞ Causal characteristics of the feature dependencies should be emphasized.

Nevertheless, the evaluation of this application suggests that the methodology of structure-based visualization can be used to analyze complex feature dependencies.

Chapter 8

Summary

In this work, a novel methodology for the visualization of graphs is presented. Here, graphs are visualized with the purpose of data mining, i.e. the visualization should help the user to understand the concepts and information hidden in the graphs.

The methodology of structure-based visualization comprises three steps: (*i*) Identification of the graph's structure by means of clustering algorithms, (*ii*) cluster classification by means of classification functions or case-based classification, and (*iii*) layout of the clusters and nodes by means of graph drawing methods.

For tabular data sets, a fourth step exists: Node similarities are defined using knowledge acquisition techniques and machine learning.

These four steps are implemented as follows:

1. **Structure Identification:** In this work, graph structures are identified by clustering algorithms. First, existing clustering approaches are described and classified according to two criterions: (*i*) Optimization criterion and (*ii*) necessary additional knowledge.

   Then, three new clustering methods are presented:

   - **MajorClust:** MAJORCLUST allows for a fast and reliable clustering of graphs. Theoretical deliberations and practical tests using six applications in chapters 6 and 7 show the high quality of MAJORCLUST's clustering results.
   - **Λ-Optimization:** Λ is a graph-theoretical quality measure for clusterings. Using Λ, features of good clusterings are presented.
   - **Adaptive Clustering:** This method integrates domain-specific knowledge into the clustering process. For this, given exemplary clusterings are analyzed and clustering knowledge is learned. In section 6.4.1, this method is applied to the clustering of fluidic circuits.

2. **Structure Classification:** Identified clusters are illustrated by a label, e.g. a network traffic cluster may be identified as a special university institute. For this, two classification approaches are presented: (*i*) Direct classification and (*ii*) case-based classification. Machine learning methods are

used to abstract the necessary classification knowledge from exemplary classifications.

3. **Structure Envisioning:** In this step, the graphs are laid out, i.e. node positions are computed. First, existing methods are presented and classified according to two criterions: (*i*) Used quality criterion and (*ii*) corresponding graph class.

   A novel method, which combines principle component analysis and force-directed drawing, is presented in section 4.2. This method is applied successfully to several domains in chapters 6 and 7.

   All graph drawing methods used for this step are extended in order to integrate clustering knowledge.

4. **Creating Graphs from Tabular Data:** In this work, tabular data is treated as a special type of graph. In order to apply the methodology of structure-based visualization to tabular data, similarities between nodes have to be defined. This is done by means of a so-called similarity measure which assesses the similarity between two nodes.

   Since the manual definition of such measures overtaxes the capabilities of most experts, a new methodology for learning of similarity measures, which allows for the application of standard machine learning methods, is described in section 5.2.2 . Three knowledge acquisition methods are introduced which implement this methodology: (*i*) Learning by means of given similarities, (*ii*) learning by classification, and (*iii*) learning by examples. All these methods allow for a fast and abstract definition of object similarities.

The visual data mining techniques introduced in this work are applied to six domains. For this, the methodology of structure-based visualization and all algorithms have been implemented within the scope of the system STRUCTUREMINER. STRUCTUREMINER is applied to the following domains:

- **Network Traffic:** Network traffic may be seen as a graph: computers and active network components are modeled by nodes while communication links form the edges. Such graphs are visualized successfully in section 6.1. The following results are worth mentioning as well:
  - Clusters are classified by a means of a case-based approach.
  - The change of traffic over a period of time is shown using a specially developed animation method.
  - STRUCTUREMINER is used to support the analysis and planing of network topologies.
  - The connections between the field of visualization and simulation are explained using a novel network simulation approach.
- **Configuration Knowledge-bases:** Systems for the configuration of technical systems use a so-called knowledge base to store their domain knowledge. This knowledge-base may be seen a graph: technical components form the nodes and causal effects result in directed edges, i.e. an edge from node v_1 to node v_2 means that if component v_1 is chosen for the

configuration, v_2 must be chosen as well. STRUCTUREMINER helps the user to understand such knowledge bases. Interesting feature are:

  - Technical subsystem are recognized automatically, i.e. clusters are labeled.
  - STRUCTUREMINER allows for the choice of different layout methods for different clusters. By analyzing the user's choices, STRUCTUREMINER learns a function which automatically chooses for future clusters the appropriate layout method.
  - Different edge shapes and colors are used to represent different types of edges.

- **Protein Interaction Graphs:** Such graphs model the interactions between proteins and are important in the context of the Human Genome Project. STRUCTUREMINER makes instructive insights into these graphs and their structure possible.

  Furthermore, MAJORCLUST is used to examine an existing protein taxonomy; it can be proved that this taxonomy groups together such proteins that have only a few interactions with each other.

- **Fluidic Circuits:** This domain is used mainly to present a method which learns a domain-dependent clustering method by analyzing given sample clusterings. In section 6.4, results are given which prove the success of this approach.

- **Document Management:** The visual presentation of large numbers of documents can help to manage and access otherwise incomprehensible document sets. In this work, each document represents a fluidic circuit.

  In order to visualize document sets, three steps are necessary:

  1. **Generation of Document Features:** For each document, i.e. fluidic circuit, abstract features are generated automatically. These features give a functional description of fluidic circuits. The reader may note that this domain is therefore an example of the visualization of tabular data sets.
  2. **Definition of Document Similarities:** The knowledge acquisition and machine learning methods described above (see "Creating Graphs from Tabular Data"), are successfully applied to the definition of a similarity measure which assess the similarity between document features.
  3. **Visualization:** Using the learned similarity measure from the previous step, a document graph is constructed: documents form nodes; edges are weighted by document similarities. This graph is visualized using the methodology of structure-based visualization.

- **Visualization of Tabular Data Features:** Tabular data is often presented in the form of a table: objects define the rows while object features become the columns. E.g. if objects represent people, reasonable features are name, age, and gender. In this application, STRUCTUREMINER is used to visualize dependencies between features. E.g. as an example, a data

set consisting of $100,000$ people who are described by 481 features, is used.

For this, two steps are necessary:

1. First, the dependencies between features are computed. Depending on the type of feature (cardinal or nominal), different machine learning techniques are used, e.g. regression or Ξ^2-test. Next a dependency-graph is constructed: Nodes model features; the graph is totally connected. Edges are weighted by the degree of dependency.
2. This graph is visualized using the methodology of structure-based visualization.

The visualization helps the user to understand the dependencies between features. Most of the identified clusters correspond to reasonable clusters, e.g. people living in rural areas or rich neighbourhoods.

In conclusion, it can be said that the methodology of structure-based visualization has been validated by (*i*) theoretical deliberations and (*ii*) by applying it to six domains. For each step of the methodology, either existing approaches have been used, existing algorithms have been extended, or new methods have been developed.

Appendix A

Machine Learning

This chapter is not intended as an introduction to the field of machine learning. Its main purpose is to define two common machine learning techniques formally: neural networks[1] and regression. Furthermore, section A.1 explains how regression and neural networks can be used to learn a classification function.

The next two sections should (*i*) remind the reader of basics concepts and (*ii*) explain some important terms. For introductions to the field of machine learning, the reader may refered to [112, 113, 6]. Neural networks are introduced in [9, 130] and regression in [184, 116, 6].

Regression and neural networks try to learn a function g which maps from $q \in \mathbf{N}$ cardinal[2] independent features onto one so-called dependent variable. Both methods employ supervised-learning, i.e. g is learned by analyzing a set $L = \{(x, g(x) \mid x \in \mathbf{R}^p\}$, the so-called learning set. Normally, a second set $T = \{(x, g(x)) \mid x \in \mathbf{R}^p \wedge (x, g(x)) \notin L\}$, the so-called testset, is given which is used to examine the quality of the learned function.

Regression and neural networks respectively use a function $\tilde{g}$ to approximate g. $\tilde{g}$ should possess two features: (*i*) A resemblance to g and (*ii*) its structure should take the learning process into consideration. Therefore, the user has to know for both methods the general structure of g. This structure is used to choose an appropriate function template for $\tilde{g}$. E.g. regression often uses the linear function $g((x_1, \ldots, x_q)) = \sum_i w_i \cdot x_i$ where $w_i \in \mathbf{R}$. Regression and neural networks have also in common that they try to minimize the same error function

$$\Upsilon(L) = \sum_{(x,g(x))\in L} (\tilde{g}(x) - g(x))^2.$$

[1] In this work, the term "neural network" denotes feed-forward neural nets which use back-propagation as a learning method.

[2] See section 5.1.

Parenthesis: In section 5.1, a so-called weighted linear similarity function of interaction level ρ (see definition 5.1.2) has been introduced. There, each $\prod_{\alpha \in A} \alpha$ defines one of $\sum_{1 \leq i \leq \rho} \binom{p}{i}$ elements[a] in x, i.e. $q = \sum_{1 \leq i \leq \rho} \binom{p}{i}$. Both regression and neural networks can learn weighted linear similarity functions of arbitrary interaction levels ρ.

[a] p, A are defined by definition 5.1.2.

A.1 Learning a Classification Function

In order to apply regression or neural networks, $g(x), x \in \mathbf{R}^p$ must be a cardinal value, i.e. $g : \mathbf{R}^p \to \mathbf{R}$. Nevertheless, regression and neural networks are also used to learn a classification function. Such a function maps from a list of features $x \in \mathbf{R}^p$ onto a set of labels $\{l_1, \ldots, l_k\}$, i.e. $g : \mathbf{R}^p \to \{l_1, \ldots, l_k\}$. For this, a set of new function $\{g_1, \ldots, g_k\}$ is introduced where $g_i : \mathbf{R}^p \to \mathbf{R}$.

The functions g_i are now used to solve the classification problem for given features $x \in \mathbf{R}^p$:

$$c(x) = l_i \Leftrightarrow g_i(x) = \max\{g_j(x) | 1 \leq j \leq k\} \text{ with } 1 \leq i \leq k$$

Since $g_i(x), x \in \mathbf{R}^p$ denotes a cardinal value, regression and neural networks can now be used to learn the functions g_i. For this, k new learning set $L_1, \ldots, L_k$ are created as follows:

$$L_i = \{(x, 0) | (x, g(x)) \in L \wedge g(x) \neq l_i\} \cup \{(x, 1) | (x, g(x)) \in L \wedge g(x) = l_i\}$$

where $1 \leq i \leq k$

Regression and neural networks can now use the learning set L_i to learn the function g_i. Obviously, the problem of learning a classification function for k classes has been reduced to the problem of learning k functions which map onto cardinal values.

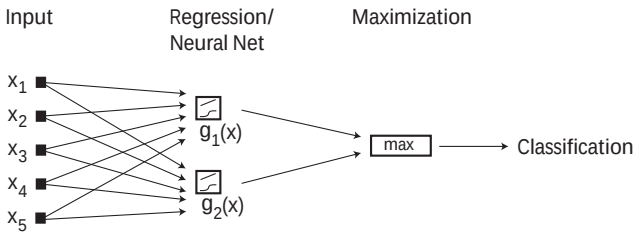

Figure A.1: *Using regression or neural networks to learn a classification function.*

This is also illustrated by figure A.1: For each class l_i a classification function g_i is learned using regression or neural networks. The classification of an input vector $x = (x_1, \ldots, x_p)$ is defined by the class l_i corresponding to the function g_i with the highest value $g_i(x)$, i.e. $\forall 1 \leq k \leq p : g_i(x) \geq g_k(x)$.

A.2 Regression

Linear Regression tries to approximate $g(x)$ by means of a linear function $\tilde{g}(x) = \sum_{1 \leq i \leq p} w_i x_i$ where $x = (x_1, \ldots, x_q)$. For this, weights $w_i \in \mathbf{R}$ are computed which minimize the term Υ. Figure A.2 gives an example: The linear function approximates the boxes whose positions are defined by the given patterns $(x, g(x)) \in L$. The dotted lines are the respective errors.

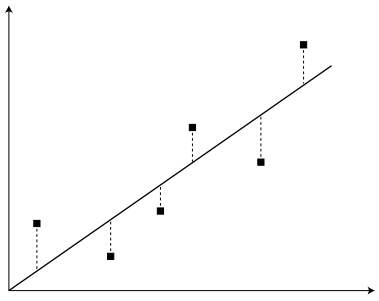

Figure A.2: *A learned linear function and the respective errors (dotted lines).*

By using matrices, linear regression can be written down more clearly. It is presumed that $L = ((x^{(1)}, g(x^{(1)})), \ldots, (x^{(|L|)}, g(x^{(|L|)})))$.

- $X \in M(|L|, p)$, i.e. X has $|L|$ rows and q columns. Row i is defined by the feature list $x^{(i)}$.
- $Y \in M(|L|, 1)$. Row i consists of $g(x^{(i)})$.
- $W \in M(|L|, 1)$. Row i consists of w_i.
- $F \in M(|L|, 1)$. Row i denotes the error defined by $g(x^{(i)}) - \tilde{g}(x^{(i)})$.

Obviously, $Y = X \cdot W + F$ holds. W can be computed by the term $W = (X^T \cdot X)^{-1} \cdot X^T \cdot Y$. For this, the matrix $X^T \cdot X$ must be invertible, i.e. the columns must be linear independent[3]. Thus, linear regression requires $O(|L|p^2 + p^3)$ time. Proofs and further explanations are given in [184, 116, 6].

Several methods exist which evaluate the quality of a regression result:

- **Coefficient of Determination:** This coefficient measures the "goodness of fit" of $\tilde{g}$, i.e. is answers the question of how closely $\tilde{g}$ approximates the patterns in L. It is defined as

$$r^2 = \frac{\sum_{1 \leq i \leq |L|} (\tilde{g}(x^{(i)}) - \bar{y})^2}{\sum_{1 \leq i \leq |L|} (g(x^{(i)}) - \bar{y})^2}$$

  where $\bar{y} = \frac{\sum_{1 \leq i \leq |L|} g(x^{(i)})}{|L|}$.

  r^2 has values between 0 (no fit) and 1 (perfect fit, i.e. all nodes are on the line).

[3] If the columns are not linear independent either neural networks are used or columns are deleted.

- **F-Test:** The F-test examines whether the regression result can also be applied to patterns which are not part of L. For this, the term $F_{emp} = \frac{r^2 \cdot (|L|-p-1)}{(1-r^2)p}$ is computed. Using the F-distribution, a standard statistical test is carried out.
- **t-Test:** A t-test can be carried out for each feature separately. This statistical hypothesis test examines whether a single feature is necessary for the regression result. Details can be found in [6].

A.3 Neural Networks

For this work, only one special class of neural nets is used: Multilayer, feed-forward perceptron networks which use backpropagation as a learning method.

Such a neural network consists of functional units, so-called perceptrons, which are arranged in layers. Let $c_{ij}, 1 \leq i \leq k_j, 1 \leq j \leq h$ denotes the ith perceptron in the jth layer. Obviously, h layers exist and layer j consists of k_j perceptrons. Perceptrons in layer $j, 1 \leq j \leq h-1$ can only be connected to other perceptrons in layer $j+1$; connections are directed towards layer $j+1$. A connection (c_1, c_2) between two perceptrons c_1 and c_2 is weighted by a factor $w(c_1, c_2) \in \mathbf{R}$. Layer 1 is called input-layer, layer h is called output-layer. V denotes the set of all perceptrons and E denotes the set of all connections. Figure A.3 (left side) shows an exemplary neural network.

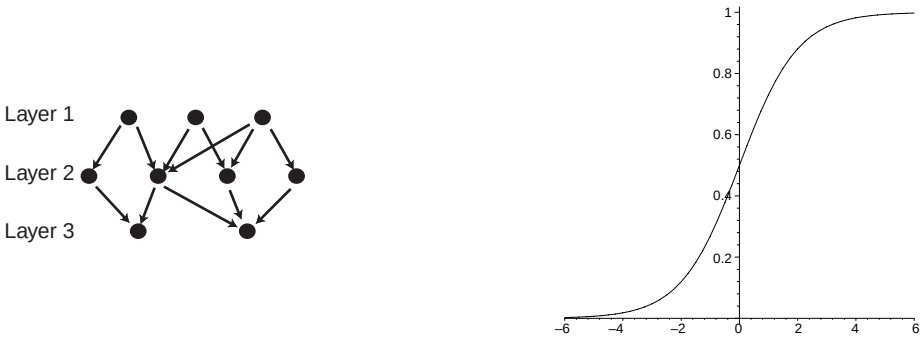

Figure A.3: *Left side: A multilayer neural net, right side: the sigmoidal threshold function*

Each perceptron c_{ij} is defined by two states: its activation $net(c_{ij})$ and its output $o(c_{ij})$. While the activation designates an internal perceptron state, the output of a perceptron can be seen by all of its descendants. The activation of a perceptron $c \in V$ is defined as follows:

$$net(c) = \sum_{(c',c) \in E} w(c', c) o(c').$$

The output of a perceptron $c \in V$ is defined as $o(c) = f(net(c))$ where f denotes a function $f : \mathbf{R} \to \mathbf{R}$. In this work, the sigmoidal function $f : x \mapsto \frac{1}{1+e^{-x}}$ is used. This function can be seen on the right side of figure A.3. The reader may note that the structure of $\tilde{g}$ is coded into the topology of the neural network.

The features given in the learning set L are used as inputs for the perceptrons of the input-layer. Using a feature value x_k as the input for a perceptron

c_{ij} in the input layer means that the output $o(c_{ij})$ of c_{ij} is set to the feature value x_k.

In this work, only one perceptron c_h is needed in the output layer. For a given feature set $x \in \mathbf{R}^p$, the output function $o(c_h)$ should approximate $g(x)$. As mentioned above, the learning set comprises both features and respective values of g. Therefore, a supervised learning strategy can be applied.

Backpropagation uses a gradient optimization method to minimize Υ. For this, the individual error terms $\gamma(x, g(x)) = (o(c_h) - g(x))^2$ are minimized iteratively ($\Upsilon(L) = \sum_{(x,g(x))\in L} \gamma(x, g(x))$). $o(c_h)$ denotes the output of c when the features x are used as inputs for the input-layer perceptrons.

Since backpropagation, unlike regression, does not minimize Υ in one step but minimizes the terms γ one after the other, the order of the elements of L may influence the learning process.

Now, all patterns are presented to the neural network, i.e. they are used as inputs for the perceptrons in the input-layer. For each pattern $(x, g(x)) \in L$, the weights of a connection (c_1, c_2) are adjusted by adding the term $\eta\delta(c_2)o(c_2)$ to the hitherto weight. η is a given learning factor which sometimes decreases over time. $\delta(c_2)$ is an error term for c_2 and is defined as follows:

$$\delta(c_2) = \begin{cases} o(c_2)(1 - o(c_2))(o(c_2) - g(x)) & c_2 = c_h \\ & \text{i.e. } c_2 \text{ belongs to} \\ & \text{the output layer} \\ o(c_2)(1 - o(c_2)) \sum_{(c_2,c_3)\in E} \delta(c_3) w(c_2, c_3) & \text{else} \end{cases}$$

Proofs and further explanations for this learning algorithm can be found in [9].

The quality of the result is often evaluated by means of the so-called mean squared error:

$$MSE(L) = \frac{\sum_{(x,g(x))\in L} \gamma(x, g(x))}{|L|} = \frac{\Upsilon(L)}{|L|}$$

The reader may note that, by having more than one perceptron in the output layer, several functions sharing the same input can be learned at the same time. E.g. each of the functions g_i, which are used for classification tasks (see the descriptions above), can be modeled by one perceptron of the same neural network. That way, the functions g_i can share parts of their computations (since they share parts of the network).

Appendix B

Some Graph-Theoretic Definitions

Definition B.0.1 (Walk)
Let $G = (V, E_c)$ be a graph and $(e_1, \ldots, e_p)$ a set of edges. If there exists a set of nodes $(v_0, \ldots, v_p)$ with $e_i = \{v_{i-1}, v_i\}$, the set of edges is called walk. If $v_0 = v_p$ holds, it is called a closed walk.

Definition B.0.2 (Path, Cycle)
Let $G = (V, E_c)$ be a graph and $(e_1, \ldots, e_p)$ a set of edges. If there exists a set of nodes $(v_0, \ldots, v_p)$ with $e_i = \{v_{i-1}, v_i\}$ and $\forall 0 \leq i, j \leq p : v_i \neq v_j$, the set of edges is called path. If $v_0 = v_p$ and $p \geq 3$ hold, the set of edges is called a cycle.

Definition B.0.3 (Length of a Path)
Let $G = (V, E_c, w)$ be a graph and $p = (v_0, \ldots, v_p)$ a path in G. The length of the path p is defined as $\sum_{0 \leq i \leq p-1} w(\{v_i, v_{i+1}\})$.

Definition B.0.4 (Distance $dist$)
Let $G = (V, E_c, w)$ be a graph. The distance $dist(v_1, v_2)$ between $v_1, v_2 \in V$ is defined as the length of the shortest path between v_1 and v_2

Definition B.0.5 (Maximum Flow)
Let $G = (V, E_c)$ be a graph. The maximum flow between $v_1, v_2 \in V$ is defined as the maximum number of edge-disjunctive paths in G from v_1 to v_2, i.e. as:
$\max(\sum_{p \in \{(v_0^1, \ldots, v_{p_1}^1), \ldots, (v_0^k, \ldots, v_{p_k}^k)} \; | v_0^i = v_1, v_{p_i}^i = v_2, v_r^i \neq v_t^i, v_r^i = v_t^j \rightarrow v_{r+1}^i \neq v_{t+1}^j \}} 1)$

For a weighted graph $G = (V, E_c, w)$ the maximum flow between $v_1, v_2 \in V$ is defined very similar, only is every path now assessed by the minimum weight of an used edge:
$\max(\sum_{p \in \{(v_0^1, \ldots, v_{p_1}^1), \ldots, (v_0^k, \ldots, v_{p_k}^k)} \; | v_0^i = v_1, v_{p_i}^i = v_2, v_r^i \neq v_t^i, v_r^i = v_t^j \rightarrow v_{r+1}^i \neq v_{t+1}^j \}} \vartheta(p))$.

$\vartheta(p)$ denotes the smallest weight of an edge in p, i.e.
$\vartheta((v_0, \ldots, v_p)) = \min_{1 \leq i \leq p-1}(w(\{v_i, v_{i+1}\}))$.

Details can be found in [80].

Definition B.0.6 (Induced Subgraph)
Let $G = (V, E_c)$ be a graph and $C \subseteq V$. The graph $G(C) = (C, E')$ with $E' = \{\{v_0, v_1\} \in E_c \mid v_0 \in C \wedge v_1 \in C\}$ is called the subgraph of G induced by C.

Definition B.0.7 (Cut of a Subgraph)
Let $G = (V, E_c)$ be a graph and $C \subseteq V$. The cut of C ($cut(C)$) is defined as:

$$cut(C) = |\{\{v_0, v_1\} \in E_c \mid v_0 \in C \wedge v_1 \notin C\}|$$

Definition B.0.8 (Tree)
An undirected graph $G = (V, E_c)$ is called a tree iff (i) G is connected and (ii) G contains no cycles.

Bibliography

[1] A.Becks, S.Sklorz, and C.Tresp. Semantic structuring and visual querying of document abstracts in digital libraries. In *Second European Conference on Research and Advanced Technology for Digital Libraries (ECDL'98)*, 1998.

[2] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases, 1993.

[3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules, 1994.

[4] D. Aha. Tolerating noisy, irrelevant, and novel attributes in instance-based learning algorithms. *International Journal of Man-Machine Studies*, 1992.

[5] J. Aslam, K. Pelekhov, and D. Rus. Using star ckusters for filtering. In *9th International Conference on Information Knowledge Management (CIKM 2000)*, 2000.

[6] K. Backhaus, B. Erichson, W. Plinke, and R. Weber. *Multivariate Anaylsemethoden*. Springer, 1996.

[7] T. Bailey and J. Cowles. Cluster definition by the optimization of simple measures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, September 1983.

[8] G. D. Battista, P. Eades, R. Tamassi, and I. G. Tollis. *Graph Drawing - Algorithms for the Visualization of Graphs*. Prentice Hall, 1999.

[9] R. Beale and T. Jackson. Neural computing. Institute of Physics Publishing, Bristol, Philadephia, 1990.

[10] R. A. Becker, S. G. Eick, and A. R. Wilks. Visualizing network data. *IEEE Transactions on Visualization and Computer Graphics*, 1(1), march 1995.

[11] K. Überla. *Faktorenanalyse*. Springer, 1977.

[12] M. Bernard. On the automated drawing of graphs. In *3rd Caribbean Conf. on Combinatorics and Computing*, 1981.

[13] M. Berry and G. Linoff. *Data Mining Techniques*. Wiley Computer Publishing, 1997.

[14] M. Bertold and D. Hand, editors. *Intelligent Data Analysis: An Introduction*. Springer Verlag, 1999.

[15] L. Bielawski and J. Boyle. *Electronic document Management Systems*. Prentice-Hall, 1997.

[16] A. Bonzano, P. Cunningham, and B. Smyth. Using introspective learning to improve retrieval in cbr: A case study in air traffic control. In *Proceedings of the Second ICCBR Conference*, 1997.

[17] I. Borg and P. Groenen. *Modern Multidimensional Scaling*. Springer, 1997.

[18] M. Bramer, editor. *Knowledge discovery and data mining: theory and practice*. IEEE Books, 1999.

[19] J. Branke, F. Bucher, and H. Schmeck. Using genetic algorithms for drawing undirected graphs. In *Proc. Third Nordic Workshop on Genetic Algorithms and thier Applications*, 1997.

[20] K. Brodlie, L. Carpenter, R. Earnshaw, J. Gallop, R. Hubbold, A. Mumford, C. Osland, and P. Quarendon, editors. *Scientific Visualization*. Springer, 1992.

[21] R. Cannon, J. Dave, and J. Bezdek. Efficient implementation of the fuzzy c-means clustering algorithms. *IEEE Trans. Pattern Analysis Machine Intelligence*, 8, 1986.

[22] M. Chalmers and P. Chitson. Bead: Explorations in information visualization. In *15th Int'l SIGIR*, 1992.

[23] W. Cleveland. *The Elements of Graphing Data*. Wadsworth Advanced Books and Software, 1985.

[24] W. Cleveland. *Visualizing Data*. AT&T Bell Laboratories, 1993.

[25] H. Clifford and W. Stephenson. *An Introduction to Numerical Classification*. Academic Press, 1975.

[26] A. Constantine and J. Gower. Graphical representation of asymmetric matrices. *Applied Statistics*, 1978.

[27] T. O. Council. http://www.olapcouncil.org/.

[28] R. H. Creecy, B. M. Masand, S. Smith, and D. Waltz. Trading mips and memory for knowledge engineering. *Communications of the ACM*, 35, 1992.

[29] R. Davidson and D. Harel. Drawing graphs nicely using simulated annealing. *ACM Trans. Graph.*, 15(4), 1996.

[30] J. de Leeuw and G. Michailides. Graph layout techniques and multidimensional data analysis. Statistics Series 248, UCLA, 1999.

[31] R. Diekmann, B. Monien, and R. Preis. Using helpful sets to improve graph bisections. Technical Report tr-rf-008-94, University of Paderborn, 1994.

[32] C. Duncan, M. Goodrich, and S. Kobourov. Balanced aspect ratio trees and their use for drawing very large graphs. In S. Whitesides, editor, *Graph Drawing*, Lecture Notes in Computer Science. Springer, 1998.

[33] P. Eades. A heuristic for graph-drawing. *Congressus Numerantium*, 42:149–160, 1984.

[34] P. Eades. Drawing free trees. Bulletin of the Institute for Combinatorics and its Application, 1992.

[35] P. Eades, R. Cohen, and M. Huang. Online animated graph drawing for web navigation. In *Proc. Graph Drawing '97*. Springer, 1997.

[36] P. Eades and Q.-W. Feng. Multilevel visualization of clustered graphs. In S. North, editor, *Graph Drawing*, Lecture Notes in Computer Science. Springer, 1996.

[37] P. Eades and Q.-W. Feng. Drawing clustered graphs on an orthogonal grid. In G. DiBattista, editor, *Graph Drawing*, Lecture Notes in Computer Science. Springer, 1997.

[38] R. Earnshaw and N. Wiseman. *AN Introductory Guide to Scientific Visualization*. Springer, 1992.

[39] L. A. R. Eli B. Messinger and R. R. Henry. A divide-and-conquer algorithm for the automatic layout of large directed graphs. *IEEE Transactions on Systems, Man, and Cybernetics*, January / February 1991.

[40] R. Elmasri and S. Navathe. *Fundamentals of Database Systems*. Benjamin / Cummings Publ. Company, 1994.

[41] B. S. Everitt. Cluster analysis. Eward Arnolds, New York, Toronto, 1993.

[42] C. Faloutsos and K.-I. Lin. Fastmap: A fast algorithm for indexing, datamining and visualization of traditional and multimedia datasets. In *SIGMOD Conference*, 1995.

[43] U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining*. AAAI / MIT Press, 1996.

[44] A. Flexer. On the use of self-organizing maps for clustering and visualization. *Principles of Data-Mining and Knowledge Discovery (PKDD99)*, 1999.

[45] K. Florek, J. Lukaszewiez, J. Perkal, H. Steinhaus, and S. Zubrzchi. Sur la liason et la division des points d'un ensemble fini. *Colloquium Methematicum*, 2, 1951.

[46] K. D. Forbus and J. de Kleer. *Building Problem Solvers*. MIT Press, Cambridge, MA, 1993.

[47] T. Fruchterman and E. Reingold. Graph-drawing by force-directed placement. *Software-Practice and Experience*, 21(11):1129–1164, 1991.

[48] G. Furnas and A. Buja. Prosection views: Dimensional interference through sections and projections. *Journal of Computational and Graphical Statistics*, 3(4), 1994.

[49] R. T. G. DiBattista, P. Eades and I. Tollis. Algorithms for drawing graphs: An annotated bibliography. *Computational Geometry*, 4, 1994.

[50] E. Gansner, E. Koutsofios, S. North, and K. Vo. A technique for drawing directed graphs, 1993.

[51] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, CA, 1979.

[52] I. Gath and B. Geva. Unsupervised optimal fuzzy clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7), 1989.

[53] Markus Rümekasten. *Hybride, tolerante Synchronisation für die verteilte und parallele Simulation gekoppelter Rechnernetze*. PhD thesis, University of Paderborn, 1997.

[54] E. Godehardt. *Graphs as Structural Models*, volume 4. Vieweg, 1988.

[55] J. Gower. Measures of similarity, dissimilarity and distance. In S. Kotz, N. Johnson, and C. Reads, editors, *Encyclopedia of Statistical Science*, volume 5. John Wiley and Sons, 1985.

[56] S. Guha, R. Rastogi, and K. Shim. Cure: An efficient clustering algorithm for large databases. *Proceedings of ACM SIGMOD International Conference on Management of Data*, 1998.

[57] R. Hadany and D. Harel. A multi-scale algorithm for drawing graphs nicely. In P. Widmayer, G. Neyer, and S. Eidenbenz, editors, *Graph Theoretic Concepts in Computer Science (WG '99)*. Springer, 1999.

[58] F. Halsall. *Data Communications, Computer Networks and Open Systems*. Addison-Wesley, 1996.

[59] J. Han and M. Kamber. *Data Mining : Concepts and Techniques*. Morgan Kaufmann, August 2000.

[60] J. Hanke and J. Reich. Kohonen map as a visualization tool for the analysis of protein sequences: multiple alignments, domains and segments of secondary structures. *Computer Applications in the Biosciences*, 12(6), 1996.

[61] D. Harel and Y. Koren. A fast multi-scale method for drawing large graphs. In *Proc. Working Conf. on Anvanced Visual Interfaces*. ACM Press, 2000.

[62] J. Hartung. *Statistik*. Oldenbourg, 1999.

[63] S. R. Hedberg. Smart data miners are cashing in on valuable information buried in private and public data sources. *Byte Magazine*, October 1995.

[64] M. Heinrich and E. W. Jüngst. A Resource-based Paradigm for the Configuring of Technical Systems for Modular Components. In *Proc. CAIA '91*, pages 257–264, 1991.

[65] B. Hendrickson. Multidimensional spectral load balancing. Technical Report SAND93-0074, Sandia National Laboratories, 1993.

[66] B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. Technical Report SAND 93-1301, Sandia National Laboratories, 1993.

[67] M. Hoffmann. *Zur Automatisierung des Designprozesses fluidischer Systeme*. eingereichte dissertation, University of Paderborn, Department of Mathematics and Computer Science, 1999.

[68] K. Holm. *Die Befragung 3*. UTB, 1976.

[69] N. Howe and C. Cardie. Examining locally varying weights for nearest neighbor algorithms. In *Proceedings of the Eleventh ICML*. Morgan Kaufmann, 1997.

[70] M. Huang and P. Eades. A fully animated interactive system for clustering and navigating huge graphs. In S. Whitesides, editor, *Graph Drawing*, Lecture Notes in Computer Science. Springer, 1998.

[71] B. Huffaker, J. Jung, D. Wessels, and K. Claffy. Visualization of the growth and topology of the nlanr caching hierarchy. http:/ / www.caida.org / Tools / Plankton / Paper.

[72] B. Huffaker, E. Nemeth, and K. Claffy. Otter: A general-purpose network visualization tool. http:/ / www.caida.org / Tools / Otter / Paper.

[73] *IJCAI Workshop on "Learning about User"*, 1999.

[74] A. Inselberg and B. Dimsdale. Parallel coordinates: A tool for visualizing multi-dimensional geometry. In *Visualization*, 1990.

[75] M. Jambu. *Explorative Datenanalyse*. Gustav Fischer Verlag, 1992.

[76] R. Jayakumar. $o(n^2)$ algorithms for graph planarization. *IEEE Trans. Comp.-Aided Design*, 8, 1989.

[77] D. Johnson, C. Aragon, L. McGeoch, and C. Schevon. Optimization by simulated annelaing; an experimental evaluation; part 1, graph partitioning. *Operations Research*, 37(6), 1989.

[78] S. Johnson. Hierarchical clustering schemes. *Psychometrika*, 32, 1967.

[79] M. J"unger, G. Rinaldi, and S. Thienel. Practical performance of efficient minimum cut algorithms. Technical Report 97.271, Angewandte Mathematik und Informatik - Univeristät zu Köln, 1997.

[80] D. Jungnickel. *Graphen, Netzwerke und Algorithmen*. BI Wissenschaftsverlag, 1990.

[81] S. T. K. Sugiyama and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernectics*, 11(2), 1981.

[82] H. Kaiser. Formulas for component scores. *Psychometrika*, 27, 1962.

[83] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31:7–15, 1989.

[84] M. Kanehisa. *Post-genom Informatics*. Oxford University Press, 2000.

[85] D. Karger and C. Stein. A new approach to the minimum cut problem. *Journal of the ACM*, 43:601–640, 1996.

[86] G. Karypis, E.-H. Han, and V. Kumar. Chameleon: A hierarchical clustering algorithm using dynamic modeling. Technical Report Paper No. 432, University of Minnesota, Minneapolis, 1999.

[87] G. Karypsis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. Technical Report 95-035, University of Minnesota, Minneapolis, 1995.

[88] J. Kay, editor. *proc. of the seventh international conference on user modeling (UM99)*. Springer, 1999.

[89] Data for the kdd cup 1998, provided by the paralyzed veterans of america. http://kdd.ics.uci.edu/databases/kddcup98/kddcup98.html, 1998.

[90] D. Keim, H.-P. Kriegel, and M. Ankerst. Recursive pattern: A technique for visualizing very large amounts of data. In *Proc. Visualization*, 1995.

[91] D. A. Keim. Databases and visualization. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, Montreal, Canada, 1996. Tutorial.

[92] D. A. Keim and H.-P. Kriegel. Visualization techniques for mining large databases: A comparison. *IEEE Transactions on Knowledge and Data Engineering*, 1996.

[93] B. Kernighan and S. Lin. Partitioning graphs. *Bell Laboratories Record*, January 1970.

[94] H. Kim. A semi-supervised document clustering technique for information organization. In *9th International Conference on Information Knowledge Management (CIKM 2000)*, 2000.

[95] K. Kira and L. Rendell. A practical approach to feature selection. In *Proceedings of the Ninth International Conference on Machine Learning*, 1992.

[96] T. Kohonen. *Self Organization and Assoziative Memory*. Springer, 1990.

[97] J. Kolodner. *Case-based reasoning*. Morgan Kaufmann, San Mateo, CA, 1993.

[98] W. L. G. Koontz and K. Fukunaga. Asymptotic analysis of a nonparametric clustering technique. *IEEE Transactions on Computers*, c-21(9), 1972.

[99] W. L. G. Koontz and K. Fukunaga. A nonparametric valley-seeking technique for cluster analysis. *IEEE Transactions on Computers*, c-21(2), 1972.

[100] A. D. Kraus. *Matrices for Engineers*. Hemisphere, 1987.

[101] J. B. Kruskal and J. B. Seery. Designing network diagrams. In *Proc. First general Conference on Social Graphics*. U.S. Department of the Census, 1980.

[102] B. Kuipers. Qualitative Simulation. *Artificial Intelligence*, 29, 1986.

[103] J. LeBlanc, M. Ward, and N. Wittels. Exploring n-dimensional databases. In *Proc. Visualization*, 1990.

[104] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout*. B. G. Teubner, Stuttgart, 1990.

[105] M. Lewis-Beck, editor. *Factor Analysis and Related Techniques*. SAGE Publications Toppan Publications, 1994.

[106] X. Lin, D. Soergel, and G. Marchionini. A self-organizing semantic map for information retrieval. In *annual international ACM SIGIR conference on research and development in information retrieval*, 1991.

[107] I. Lisitsyn and V. Kasyanov. Higres - visualization system for clustered graphs and graph algorithms. In J. Kratochvil, editor, *Graph Drawing*, Lecture Notes in Computer Science. Springer, 1999.

[108] G. Liu. *Introduction to Combinatorical Mathematics*. McGraw Hill, 1968.

[109] J. McCarthy. private communications. Technical report, Stanford University, 1956.

[110] B. Meyer. Self-organizing graphs - a neural network perspective of graph layout. In S. Whitesides, editor, *Graph Drawing 1998*. Springer, 1998.

[111] S. Meyer zu Eißen. Natürliche Graphpartitionierung am Beispiel von Aufgabenmodellen in Unternehmensnetzwerken. Master's thesis, University of Paderborn, 2000.

[112] R. Michalski, J. G. Carbonell, and T. Mitchell, editors. *Machine Learning: An Artificial Intelligence Approach*, volume 1. Tioga, Palo Alto, California, 1983.

[113] R. S. Michalski, I. Bratko, and M. Kubat, editors. *Machine Learning & Data Mining: Methods and Applications*. John Wiley, 1998.

[114] B. Mirkin. *Mathematical Classification and Clustering*. Kluwer Academics Publishers, 1996.

[115] S. Mitchell Hedetniemi, E. Cockayne, and S. Hedetniemi. Linear Algorithms for Finding the Jordan Center and Path Center of a Tree. *Transportation Science*, 15:98–114, 1981.

[116] R. H. Myers. *Classical and Modern Regression with Applications*. Duxbury Press, Boston, 1986.

[117] H. Nagamochi, T. Ono, and T. Ibaraki. Implementing an efficient minimum capacity cut algorithm. *Mathematical Programming*, 67, 1994.

[118] NetFactual. http://www.netfactual.com/, January 2001.

[119] F. J. Newbery. Edge concentration: A method for clustering directed graphs. *Software Engineering Notes, ACM Press*, 14(5), 1997.

[120] A. Newell. *IPL-V Programmer's Manual*. Rand Corporation, rm-3739-rc edition, 1963.

[121] R. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. *Proc. of the VLDB Conference, Santiago, Chile*, 1994.

[122] O. Niggemann and B. Stein. Visualization by example. In *Proceedings Adaptivität und Benutzermodellierung in interaktiven Softwaresystemen (ABIS-2000)*, 2000.

[123] O. Niggemann, B. Stein, and M. Suermann. On Resource-based Configuration—Rendering Component-Property Graphs. In J. Sauer and B. Stein, editors, *12. Workshop "Planen und Konfigurieren"*, tr-ri-98-193, Paderborn, Apr. 1998. University of Paderborn, Department of Mathematics and Computer Science.

[124] O. Niggemann, B. Stein, and J. Tölle. Visualization of network traffic. In *IEEE International Conference on Communications (ICC 2001)*, 2001.

[125] O. Nigggemann and B. Stein. A meta heuristic for graph drawing. *Proceedings of the Working Conference on Advanced Visual Interfaces (AVI 2000)*, 2000.

[126] B. Nour-Omid, A. Raefsky, and G. Lyzengaomid:1986. Solving finite element equations on concurrent computers. In *Parallel computers and thier impact on mechanics*. AM. Soc. Mech. Eng., New York, 1986.

[127] The network simulator - ns-2. http://www.isi.edu/nsnam/ns/.

[128] D. Ostry. Drawing graphs on convex surfaces. Master's thesis, Dept. of Computer Science, University of Newcastle, 1996.

[129] T. Ozawa and H. Takahashi. *Graph Theory and Algorithms*, volume 108 of *Lecture Notes in Computer Science*, chapter A Graph-planarization Algorithm and its Application to Random Graphs. Springer-Verlag, 1981.

[130] Y.-H. Pao. *Adaptive Pattern Recognition and Neural Networks*. Addison-Wesley, Reading, MA, 1989.

[131] A. Papakostas and I. Tollis. Algorithms for area-efficient orthogonal drawings. *Comp. Geom. Theory Appl.*, 9(1-2), 1998.

[132] J. Park, M. Lappe, and S. A. Teichmann. Mapping protein family interactions. submitted, 2000.

[133] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[134] K. Pearson. Contribution to the mathematical theory of evolution. *Phil. Trans.*, 1894.

[135] X. L. Peter Eades and W. F. Smyth. A fast and effictive heuristic for the feedback arc set problem. In *Information Processing Letters 47*, Elsevier Science Publishers, 1993.

[136] A. Plana. http://altaplana.com/olap/.

[137] *Proceedings Adaptivität und Benutzermodellierung in interaktiven Softwaresystemen (ABIS-2000)*, 2000.

[138] A. Reckmann. Ähnlichkeitsmaße und deren Parametrisierung für die fallbasierte Diagnose am Beispiel einer medizinischen Anwendung. Master's thesis, University of Paderborn, 1999.

[139] E. Reingold and J. Tilford. Tidier drawing of trees. *IEEE Transactions on Software Engineering*, 7(2):223–228, 1981.

[140] J. Rekimoto and M. Green. The information cube: Using transparency in 3d informattion visualization. In *Proc. 3rd Annual Workshop on Information Technologies and Systems*, 1993.

[141] J. Ritsert, E. Stracke, and F. Heider. *Grundzüge der Varianz- und Faktorenanalyse*. Campus Verlag, 1976.

[142] S. J. Roberts. Parametic and non-parametric unsupervised cluster analyis. *Pattern Recognition*, April 1996.

[143] T. Roxborough and Arunabha. Graph Clustering using Multiway Ratio Cut. In S. North, editor, *Graph Drawing*, Lecture Notes in Computer Science, Springer Verlag, 1996.

[144] M. R"umekasten. *ATLAS - Eine Kurzeinf"uhrung*. University of Paderborn, March 1992.

[145] R. Sablowski and A. Frick. Automatic Graph Clustering. In S. North, editor, *Graph Drawing*, Lecture Notes in Computer Science, Springer Verlag, 1996.

[146] S. L. Salzberg. A nearest hyperrectangle learning method. *Machine Learning*, 1991.

[147] G. Sander. Graph Layout through the VCG Tool. Technical Report A/03/94, 1994.

[148] G. Sander. Layout of Compound Directed Graphs. Technical Report A/03/96, 1996.

[149] A. Schulz. Graphenanalyse hydraulischer Schaltkreise zur Erkennung von hydraulischen Achsen und deren Kopplung. Master's thesis, University of Paderborn, 1997.

[150] Structural classification of proteins. http://scop.mrc-lmb.cam.ac.uk/scop/index.html.

[151] B. Seck. Ein Simulationsmodel zur Auswertung aktiver Komponenten in einem Konfigurationssystem für Rechnernetze. Master's thesis, University of Paderborn, 1997.

[152] B. Shneiderman. Tree visualizations with treemaps: A 2d space-filling approach. *ACM Transactions on Graphics*, 11(1), 1992.

[153] A. Shoshani. Olap and statistical databases: Similarities and differences. In *Proc. ACM PODS*, 1997.

[154] P. Sneath. The application of computers to taxonomy. *J. Gen. Microbiol.*, 17, 1957.

[155] P. Sneath and R. Sokal. *Principles of Numerical Taxonomy*. W.H. Freeman Comp., 1973.

[156] H. Sp"ath. *Cluster-Analyse-Algorithmen*. R. Oldenbourg Verlag, 1975.

[157] C. Stanfill and D. Waltz. Toward memory-based learning. *Communications of the ACM*, 29:1213–1228, 1986.

[158] B. Stein. Supporting Hydraulic Circuit Design by Efficiently Solving the Model Synthesis Problem. In *International ICSC Symposium on Engineering of Intelligent Systems (EIS 98)*, pages 1274–1280. ICSI Academic Press, Feb. 1998.

[159] B. Stein and M. Hoffmann. On adaptation in case-based design. In *Proceedings of the third International ICSC Symposia on Soft Computing (SOCO '99)*. ICSC Academic Press, 1998.

[160] B. Stein and O. Niggemann. *25. Workshop on Graph Theory*, chapter On the Nature of Structure and its Identification. Lecture Notes on Computer Science, LNCS. Springer, Ascona, Italy, July 1999.

[161] B. Stein and O. Niggemann. Generation of similarity measures from different sources. The Fourteenth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA / AIE-2001), 2000.

[162] B. Stein, O. Niggemann, and U. Husemeier. Learning Complex Similarity Measures. In *Jahrestagung der Gesellschaft für Klassifikation*, Bielefeld, Germany, 1999. Springer.

[163] B. Stein and A. Schulz. Topological analysis of hydraulic systems. Technical Report tr-ri-98-197, University of Paderborn, 1998.

[164] B. Stein and E. Vier. *Recent Advances in Information Science and Technology*, chapter An Approach to Formulate and to Process Design Knowledge in Fluidics. Second Part of the Proceedings of the 2nd IMACS International Conference on Circuits, Systems and Computers, CSC '98. World Scientific, London, 1998.

[165] P. Struss. Fundamentals of Model-Based Diagnosis of Dynamic Systems. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence, Nagoya, Japan*, 1997.

[166] K. Sugiyama and K. Misue. Visualization of structural information: Automatic drawing of compound graphs. *IEEE Transactions on Systems, Man, and Cybernetics*, 1991.

[167] K. Sugiyama and K. Misue. Graph drawing by magnetic-spring model. *J. Viusal Lang. Comp.*, 6(3), 1995.

[168] K. Sugiyama and K. Misue. A simple and unified method for drawing graphs. In R. Tamassia and I. Tollis, editors, *Proc. Graph Drawing*. Springer, 1995.

[169] K. Supowit and K. Misue. The complexity of drawing trees nicely. *Acta Informatica*, 18:359–368, 1983.

[170] A. S. Tanenbaum. *Computer Networks*. Prentice Hall, 1996.

[171] L. Technologies, R. B. Systems, and SGI. Improved customer control through churn management. http://www.sgi.com/software/mineset/tech_info/churn.html.

[172] W. Togerson. Multidimensional scaling: I. theory and method. *Psychometrika*, 1952.

[173] E. Tufte. *The Visual Display of Quantitative Information*. Graphcs Press, 1983.

[174] J. Tukey. *Exploratory Data Analysis*. Addison-Wesly, 1977.

[175] D. Tunkelang. *A Numerical Optimization Approach to General Graph Drawing*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, 1999.

[176] J. Ullman. *Priciples of Database and Knowledge-Based Systems, Vol. I/II*. Computer Science Press, 1988.

[177] K. Urahama. Unsupervised learning algorithm for fuzzy clustering. *IEICE Trans. Inf. and Syst.*, E76-D(3), 1993.

[178] A. Wagner. Optimierungsverfahren für das Graph-Layout Problem. Master's thesis, Universität Paderborn, 2001.

[179] X. Wang and I. Miyamoto. Generating customized layouts. In *Graph Drawing*. Springer, 1995.

[180] S. Wess. Fallbasiertes Problemlösen in wissensbasierten Systemen zur Entscheidungsunterstützung und Diagnostik: Grundlagen, Systeme und Anwendungen. Technical report, Sankt Augustin: Infix, 1996.

[181] J. H. Wilkinson. *The Algebraic Eigenvalue Problem*. Clarendon Press, 1965.

[182] R. Wilson and M. Span. A new approach to clustering. *Pattern Recognition*, 23(12), 1990.

[183] J. Wise, J. Thomas, K. Pennock, and D. Lantrip. Visualizing the nonvisual: Spatial analysis and interaction with information from text documents. In *Proc. IEEE Inf. Visualization*, 1995.

[184] T. Wonnacott and R. Wonnacott. Regression: a second course in statistics. John Wiley & Sons, New York, Chichester / Brisbane / Toronto, 1981.

[185] Z. Wu and R. Leahy. An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, November 1993.

[186] J.-T. Yan and P.-Y. Hsiao. A fuzzy clustering algorithm for graph bisection. *Information Processing Letters*, 52, 1994.

[187] J. York and S. Bohn. Clustering and dimensionality reduction in spire. In *Automated Intelligence Processing and Analysis Symposium*, 1995.

[188] C. T. Zahn. Graph-theoretical methods for detecting and describing gestalt clusters. *IEEE Transactions on computers*, C-20(1), 1971.