8. Drawing Clusters and Hierarchies

Ralf Brockenauer and Sabine Cornelsen

Large graphs such as WWW connection graphs or VLSI schematics cannot be drawn in a readable way by traditional graph drawing techniques. An approach to solve this problem is, for example, fish-eye representation, which allows one to display a small part of the graph enlarged while the graph is shown completely (see e.g. Formella and Keller (1995)). Another way is drawing only a part of the graph. The method presented here is clustering, that is grouping the vertex set.

Apart from the use of clustering to draw large graphs, already clustered graphs occur in applications such as statistics (see e.g. Godehardt (1988)), linguistics (see e.g. Batagelj et al. (1992)) or divide and conquer approaches. To visualize these structures it is also important to find a method of drawing clustered graphs in an understandable way.

In this chapter, Section 8.1 gives an overview of several terms occurring in connection with clustering in the literature. Section 8.2 presents a few main methods of finding good clusters. The following three sections introduce some graph drawing algorithms for clustered graphs. Beginning with the special case of planar drawing methods in Section 8.3, Section 8.4 works on general graphs with a hierarchical structure, called compound graphs, and Section 8.5 deals with arbitrarily clustered graphs using force-directed methods. Finally, Section 8.6 shows a case study for drawing partially known huge graphs.

8.1 Definitions

The usage of the term clustering is not determined uniquely in the literature. In this chapter several terms concerning clustering are defined to give an overview. We only consider vertex clustering, but it is worth mentioning that clustering with respect to edges can be of interest as well. A method to do this can be found in Paulish (1993, Chapter 5).

Clustering of graphs means grouping of vertices into components called clusters. Thus, clustering is related to partitioning the vertex set.

**Definition 8.1 (Partition).** A (k-way) partition of a set \( C \) is a family of subsets \( (C_1, \ldots, C_k) \) with

\[
\bigcup_{i=1}^{k} C_i = C \quad \text{and} \quad C_i \cap C_j = \emptyset \quad \text{for} \quad i \neq j.
\]

The \( C_i \) are called parts. We refer to a 2-way partition as a bipartition.

Now, we can define one of the most basic definitions of clustered graphs.
Definition 8.2 (Clustered Graph). A clustered graph is a graph with a partition $(C_1, \ldots, C_k)$ on the vertex set. The $C_i$ are called clusters.

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{clustered_graph.png}
\caption{a) A clustered graph. Clusters are framed with rectangles. b) Quotient graph of the clustered graph in a).}
\end{figure}

Sometimes (e.g. Alpert and Kahn (1995)) the term clustering is only used for large $k \in \Theta(n)$ where $n$ is the number of vertices. The expression clustered graph is also used to denote the quotient graph defined by the partition.

Definition 8.3 (Quotient Graph). For a partition $(C_1, \ldots, C_k)$ on the vertex set of a graph $G = (\mathcal{V}, \mathcal{E})$, the quotient graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined by shrinking each part into a single node, i.e.

- $\mathcal{V} = \{C_1, \ldots, C_k\}$ and
- $(C_i, C_j) \in \mathcal{E} \iff i \neq j$ and $\exists v \in C_i, w \in C_j \ (v, w) \in \mathcal{E}.$

The elements of $\mathcal{V}$ are called nodes.

The quotient graph of the clustered graph in Figure 8.1 a) is shown in Figure 8.1 b). A drawing of the quotient graph is also called a black-box drawing (Paulish, 1993) of the corresponding clustered graph. Another way of shrinking subgraphs into a single node is proposed by Lengauer (1990). He calls the construction hierarchical graph. Lengauer used this structure to find faster algorithms, e.g., for planarity testing (Lengauer, 1989) or connectivity testing, for large graphs.

Definition 8.4 (Hierarchical Graph). A hierarchical graph is a finite sequence $\Gamma = (G_1, \ldots, G_k)$ of graphs $G_i$ called cells. The vertex set of the cells is divided into pins and inner vertices. The set of inner vertices, again, is divided into terminals and non-terminals. Each non-terminal has a type. The type of a non-terminal in $G_i$ is a cell $G_j$ with $j < i$. The degree of a non-terminal $v$ is the number of pins of its type $G$ and the neighbors of $v$ are bijectively associated with the pins of $G$. 
Fig. 8.2. Example for a hierarchical graph. Pins are drawn as rectangles, inner vertices as circles and non-terminals are shaded grey. The bijection between the neighbors of a non-terminal and the associated pins is given via the position in the figure.

An example for a hierarchical graph is shown in Figure 8.2. Note, that the cells need not to be connected. A hierarchical graph represents a graph which is obtained by expansion. It is a substitution mechanism that glues pins of a cell to neighbors of non-terminals the type of which is this cell. Note that if \( \Gamma = (G_1, \ldots, G_k) \) is a hierarchical graph, so is any prefix \( \Gamma_i = (G_1, \ldots, G_i), 1 \leq i \leq k \).

**Definition 8.5 (Expansion).** The expansion \( G(\Gamma) \) of a hierarchical graph \( \Gamma = (G_1, \ldots, G_k) \) is obtained recursively as follows:

- \( k = 1: \ G(\Gamma) = G_1 \)
- \( k > 1: \) For each non-terminal \( v \) of \( G_k \), let \( v \) be of type \( G_j \). Delete \( v \) and its incident edges and insert a copy of \( G(\Gamma_j) \) by identifying pins of \( G(\Gamma_j) \) with their associated vertex in \( G_k \).

Thus, a hierarchical graph is a clustering of the expansion graph and clusters can include other clusters. This can be illustrated as a tree – the hierarchy tree. The expansion and the hierarchy tree of the hierarchical graph in Figure 8.2 are shown in Figure 8.3.

Fig. 8.3. Example for the a) expansion and b) inclusion tree of the hierarchical graph in Figure 8.2.

A quite similar concept introduced by Feng et al. (1995) are hierarchical clustered graphs.

**Definition 8.6 (Hierarchical Clustered Graphs).** Hierarchical clustered graphs \( C = (G, T) \) consist of a graph \( G = (V, E) \) and a rooted tree
8. Drawing Clusters and Hierarchies

Let $T$ such that the leaves of $T$ are exactly $V$. Vertices of $T$ are called nodes. Each node $v$ of $T$ represents the cluster $V(v)$ of leaves in the subtree of $T$ rooted at $v$. $T$ is called the inclusion tree of $C$. An edge $e$ is said to be incident to a cluster $V(v)$, if $|e \cap V(v)| = 1$.

An example of a hierarchical clustered graph is shown in Figure 8.4 a). Algorithms for drawing planar hierarchical clustered graphs are presented in Section 8.3. A way of generalization is to allow clusters to intersect. Combined with a hierarchical structure this leads to compound graphs presented by Sugiyama and Misue (1991).

**Definition 8.7 (Compound Graphs).** A compound (directed) graph is a triple $D = (V, E, I)$ such that $D_a = (V, E)$ is a (directed) graph and $D_c = (V, I)$ is a directed graph. The elements of $E$ are called adjacency edges, those of $I$ inclusion edges.

Thus, $(v, w) \in I$ means that $v$ includes $w$. Of course, this interpretation only makes sense if the directed graph $D_c$ is acyclic. An example of a compound graph is shown in Figure 8.4 b). In Section 8.4 a drawing algorithm for compound graphs is given in the special case where $D_c$ is a rooted tree and $D_a$ is a directed graph, such that no vertex is adjacent to an ancestor. A hierarchical clustered graph can be seen as a compound graph where $D_c$ is a rooted tree and adjacency edges are only incident to leaves. Note the different meaning of $V$.

![Diagram](image)

**Fig. 8.4.** a) A hierarchical clustered graph. The inclusion tree is drawn by the inclusion representation. b) A compound graph. Inclusion edges are drawn as including rectangles.
8.2 Clustering Methods

There is a large number of clustering approaches. This section will only mention some of them to give an idea of the main methods. A good overview is given in Alpert and Kahng (1995) or Jain and Dubes (1988). The approaches introduced in this section that are not cited separately can be found there as well. Some partitioning methods concerning VLSI are also summarized in Lengauer (1990).

Clustering is a type of classification. This classification can be extrinsic or intrinsic. Extrinsic classification uses category labels on the objects and clusters are defined by these categories. For example, take trade relation between world wide spread companies and countries as clusters. On the other hand intrinsic classification is only based on the structure of the graph. In the following discussion, we consider intrinsic classification.

There are two main goals in clustering graphs. The first, which has applications, for example, in VLSI, parallel computing and divide-and-conquer algorithms, is to partition a graph into clusters of about the same size and with as few edges connecting the clusters as possible. The second one, which is a method used in statistical applications, is to explore the structure of the data. Thus, the number of clusters is not fixed.

8.2.1 k-Way Partition

The first goal can be formalized as the \textit{Min-Cut k-Way Partition} (cf. Lengauer (1990) p. 253). In the course of the following passage, let $G = (V, E)$ be a graph with vertex-weight $c : V \rightarrow \mathbb{N}$ and edge-weight $w : E \rightarrow \mathbb{N}$.

\textbf{Definition 8.8 (Min-Cut k-Way Partition).} Given a fixed $k \in \mathbb{N}$ and $b(i), B(i) \in \mathbb{N}$ for $i = 1, \ldots, k$, find among all $k$-way-partitions $(C_1, \ldots, C_k)$ of $V$ which satisfy $b(i) \leq c(C_i) \leq B(i)$ for all $i = 1, \ldots, k$ one that minimizes the weight of the partition

$$w(C_1, \ldots, C_k) = \frac{1}{2} \sum_{i=1}^{k} \sum_{1 \leq j \leq i} w(e).$$

Exact cluster size balance is achieved by setting $b(i) = c(V)/k - \epsilon$ and $B(i) = c(V)/k + \epsilon$, where $\epsilon > 0$ may be necessary to obtain a solution at all.

\textbf{Move-based Approaches.} Unfortunately, the multiway partition problem is $\mathcal{NP}$-complete even in the special case of bipartition. A classical good graph bipartitioning heuristic was introduced by Kernighan and Lin (1970). A natural local search method for solving this problem is to start with an initial bipartition and to exchange pairs of vertices across the cut, if doing so improves the cut-size. To reduce the danger of being trapped in local minima, Kernighan and Lin modified the search, proceeding in a series of passes.
During each pass of the algorithm, every vertex moves exactly once. At the beginning of a pass each vertex is unlocked. Iteratively the pair of unlocked vertices with the highest gain is swapped, where the gain of vertices \( v_1 \in C_1, v_2 \in C_2 \) is defined by

\[
w(C_1, C_2) = w((C_1 \cup \{v_2\}) \setminus \{v_1\}, (C_2 \cup \{v_1\}) \setminus \{v_2\})
\]

that is the decrease in cut-weight that results from the pair swap. Then, both swapped vertices are locked. The swapping process is iterated until all vertices become locked. The bipartition with the lowest cut-weight observed during the pass is the initial bipartition for the next pass. The algorithm terminates when a pass fails to find a solution with lower weight than its initial bipartition.

Maintaining a sorted list of gains, the complexity of this algorithm is in \( O(n^2 \log n) \). Fiduccia and Mattheyses modified the algorithm of Kernighan and Lin to permit an \( O(|E|) \) implementation. The main difference is that a new bipartition is derived by moving a single vertex either from \( C_1 \) to \( C_2 \) or from \( C_2 \) to \( C_1 \) instead of exchanging two of them. Therefore, the algorithm must violate the exact cluster size balance constraint. The solution is permitted to deviate from an exact bipartition by the size of the largest vertex. The algorithm of Fiduccia and Mattheyses can also be extended to \( k > 2 \).

There are several functions combining cluster size balance and minimization of cut weights within a single objective. One of them is the Ratio Cut Partition proposed by Wei and Cheng (1991) for \( k = 2 \). Their approach has several generalizations for arbitrary \( k \). One of them is presented by Ribeiro-Neto and Sen (1997).

**Definition 8.9 (Ratio Cut Partition).** Find among all \( k \)-way-partitions \( (C_1, \ldots, C_k) \) of \( V \) one that minimizes

\[
R(C_1, \ldots, C_k) = \frac{w(C_1, \ldots, C_k)}{c(C_1) \cdot \ldots \cdot c(C_k)}.
\]

Finding a ratio cut partition is also \( \text{NP} \)-complete. In Wei and Cheng (1991) a heuristic based on the algorithm of Fiduccia and Mattheyses is proposed to find good bipartitions. In Figure 8.5 an example is shown where the ratio cut partition is a much more intuitive one than the exact cluster size balanced min-cut bipartition.

**Spectral Methods.** For a graph \( G = (\{v_1, \ldots, v_n\}, E) \) with enumerated vertex set, a partitioning solution can be represented in terms of vectors and matrices.

**Definition 8.10 (Characteristic Vector).** For a graph \( G \) with a given a \( k \)-way partition \( (C_1, \ldots, C_k) \) the characteristic vector for cluster \( C_h \) is the \( n \)-dimensional vector \( x_h = (x_{1h}, \ldots, x_{nh}) \in \{0, 1\}^n \) with \( x_{ih} = 1 \) if and only if \( v_i \in C_h \). The \( n \times k \) matrix \( X \) with column \( h \) equal to \( x_h \) is the assignment matrix of the partition.
Let \( A = (a_{ij}) \) be the adjacency matrix of an undirected graph \( G \) and \( D = (d_{ij}) \) the degree matrix, that is \( d_{ij} = \deg(v_i) \) for \( i = j \) and zero otherwise. The Laplacian matrix of \( G \) is defined as \( L = D - A \). Since \( L \) is a symmetric matrix

- all eigenvalues of \( L \) are real and
- there is a basis of the \( n \)-dimensional space of mutually orthogonal eigenvectors of \( L \).

Since \( x^T L x = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} (x_i - x_j)^2 \geq 0 \) for all \( x \in \mathbb{R}^n \), matrix \( L \) is positive semi-definite and thus, all eigenvalues are non-negative. Furthermore, the columns of \( L \) add to zero and we get

- the smallest eigenvalue of \( L \) is 0 with corresponding eigenvector \((1, \ldots, 1)\).

(The multiplicity of 0 as an eigenvalue of \( L \) is equal to the number of connected components of \( G \).)

Now, let \( x \) denote the characteristic vector for one part of a given partition \((C_1, C_2)\) in a connected undirected graph, then we can express the value of the cut defined by the partition by

\[
w(C_1, C_2) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} (x_i - x_j)^2 = x^T L x.
\]

Allowing non-discrete solutions, a normalized eigenvector \( \mu \) to the smallest positive eigenvalue minimizes \( x^T L x \) among all normalized \( x \). Although a non-discrete solution for \( x \) is meaningless, this result suggests heuristically finding the discrete solution closest to \( \mu \). Given cluster size constraints \( |C_1| = m_1 \) and \( |C_2| = m_2 \), the closest discrete solution is obtained by placing the \( m_1 \) vertices with the highest coordinates of \( \mu \) in \( C_1 \) and the rest in \( C_2 \) or vice versa.

This approach is known as spectral bipartitioning. Unfortunately, it can be arbitrarily worse than optimal, as illustrated by the following example: Let \( G \) be the graph shown in Figure 8.6 in which two \( n/4 \)-cliques are connected, each by a single edge, to an \( n/2 \)-clique. Spectral bipartition will cut \( K_{n/2} \) into equal halves, cutting \((n/4)^2\) edges. But the optimal cluster size balanced bipartition has weight 2.

**Fig. 8.5.** a) Exact cluster size balanced min-cut bipartition and b) ratio cut bipartition.
To improve the result, Frankle and Karp proposed to find an characteristic vector that is close to a linear combination of the eigenvectors of the $d$ smallest eigenvalues. A summary of spectral clustering including an extension to $k > 2$ is given in Alpert and Kahng (1995) Section 4.

8.2.2 Structural Clustering

We now turn to the second goal of graph clustering which tries to identify certain intuitive properties $p$ such as cliques or connectivity. In contrary to the last section, the number of clusters is not given. One important algorithm used to solve this problem is agglomerative clustering, which starts with the $n$-way partition and constructs iteratively a $k$-way partition from the $k + 1$-way partition.

For a complete graph $G = (V, E)$ with edge weight $w : E \to \mathbb{R}$ and no two edges having the same weight, the iterative step is defined in more detail, for example, by Hubert (see Jain and Dubes (1988) p. 63). For a given threshold $d$ let $G_d = (V, \{ e \in E : w(e) \leq d \})$. For a pair of clusters $(C_r, C_s)$ in the $k + 1$-way partition, define

$$Q_p(C_r, C_s) = \min \{ d; \text{the subgraph induced by } C_r \cup C_s \text{ in } G_d \text{ is either complete or has property } p \}.$$  

Merge cluster $C_p$ and $C_q$ if

$$Q_p(C_p, C_q) = \min_{r,s} Q_p(C_r, C_s).$$

Each specification of property $p$ defines a new clustering method. Every cluster must at least be connected. Some suitable graph properties $p(k)$ with integer parameter $k$ are listed below:

- **$k$-edge-connectivity**: All pairs of vertices are joined by at least $k$ edge disjoined paths. 1-edge-connectivity is also called **single-link** property and $k$-edge-connectivity is called **complete-link** property.
- **$k$-vertex-connectivity**: All pairs of vertices are joined by at least $k$ vertex disjoined paths.
- **vertex degree $k$**: A connected graph such that each vertex has at least degree $k$.
- **diameter $k$**: All pairs of vertices are joined by a path of length at most $k$. 

**Fig. 8.6.** Example of bad spectral bipartition.
An agglomerative algorithm constructs a hierarchical clustering. The above mentioned algorithm is serial. To minimize the height of the inclusion tree, an alternative strategy is to find many good clusters to merge, then perform all merges simultaneously.

A special drawing of the inclusion tree of a hierarchical clustered graph, that reveals the order in which clusters are merged, is called dendrogram. Cutting a dendrogram horizontally creates a partition of the vertex set. Figure 8.7 gives two examples on a weighted $K_5$, one for single- and the other for complete-link property.

![Dendrogram Diagram]

**Fig. 8.7.** Two dendrograms on a weighted $K_5$. a) single link, b) complete link.

### 8.2.3 Other Approaches

There are several other methods for clustering graphs. Duncan et al. (1998) presented a method to partition an already laid out graph along horizontal, vertical and diagonal lines. The approach of Sablowski and Frick (1996) is based on the successive identification of patterns in the graph. Other approaches consider constraints, e.g., some vertices should belong to different clusters. Nakano et al. (1997) presented a work on this topic.

Most of the clustering methods presented so far produce connected clusters. But it is important to know, that unconnected clusters might occur in practice. This can be seen immediately by considering extrinsic clustering, because it can be defined arbitrarily. But there are also some intrinsic classifications which produce unconnected components. They are, for example, used in social network analysis. The following two classifications presented by Wasserman and Faust (1994) are of this type.

**Definition 8.11 (Structural Equivalence).** Two vertices $v, w$ of a graph $(V, E)$ are structural equivalent if and only if they have the same neighborhood, that is if and only if for all $u \in V$ holds

$$ (v, u) \in E \iff (w, u) \in E $$

$$ (u, v) \in E \iff (u, w) \in E. $$

Thus, two vertices are structural equivalent if their rows and columns in the adjacency matrix are identical.
Definition 8.12 (Regular Equivalence). Two vertices \( v \) and \( w \) of a graph \((V,E)\) are regular equivalent \((v \equiv w)\) only if they have equivalent neighborhoods, that is only if for all \( u \in V \) holds
\[
(v,u) \in E \implies \exists u' \in V \quad u' \equiv u \wedge (w,u') \in E
\]
\[
(u,v) \in E \implies \exists u' \in V \quad u' \equiv u \wedge (u',w) \in E.
\]

Structural equivalence is a special case of regular equivalence. Regular equivalence partitioning is not uniquely determined. The partition with the fewest equivalence classes that is consistent with the definition of regular equivalence is called the maximal regular equivalence. For example, in a tree where all leaves have the same height, taking the levels as equivalence classes yields the maximal regular equivalence on that tree.

8.3 Planar Drawings of Hierarchical Clustered Graphs

In her PhD thesis, Feng (1997) presented a characterization for planar connected hierarchical clustered graphs and introduced some algorithms for drawing them.

Definition 8.13 (Connected Hierarchical Clustered Graphs). A hierarchical clustered graph \( C = (G,T) \) is connected, if each cluster induces a connected subgraph of \( G \).

Definition 8.14 (Drawing of Hierarchical Clustered Graph). A drawing \( D \) of a hierarchical clustered graph \( C = (G,T) \) includes the drawing of the underlying graph \( G \) and of the inclusion tree \( T \) in the plane. Each vertex of \( G \) is represented as a point and each edge \( \{v,w\} \) as a simple curve between \( D(v) \) and \( D(w) \). Each non-leaf node \( v \) of \( T \) is drawn as a simple closed region \( D(v) \) bounded by a simple closed curve such that
\[- D(\mu) \subset D(v) \text{ for all descendants } \mu \text{ of } v.
\][- D(\mu) \cap D(v) = \emptyset \text{ if } \mu \text{ is neither a descendant nor an ancestor of } v.
\][- D(e) \subset D(v) \text{ for all edges } e \text{ of } G \text{ with } e \subset V(v).
\][- D(e) \cap D(v) \text{ is one point if } |e \cap V(v)| = 1.
\]

The drawing of an edge \( e \) and a region \( D(v) \) have an edge-region-crossing, if \( e \cap V(v) \neq \emptyset \) but \( D(e) \cap D(v) \neq \emptyset \). Drawings where this occurs, are allowed, but they are not c-planar.

Definition 8.15 (c-Planar). A drawing of a hierarchical clustered graph is c-planar (compound planar), if there are no crossing edges and no edge-region-crossings.

For example, the drawing of the graph shown in Figure 8.4 on page 197 is c-planar.
**Theorem 8.16 (Characterization of c-Planar Graphs).** A connected hierarchical clustered graph \( C = (G, T) \) is c-planar if and only if there exists a planar drawing of \( G \), such that for each node \( \nu \) of \( T \) all vertices of \( V - V(\nu) \) are in the outer face of the drawing of \( G(\nu) \).

**Proof.** Consider a clustered graph \( C = (G, T) \) with a c-planar drawing \( D \). Suppose there is a node \( \nu \) of \( T \) and a vertex \( v \in V - V(\nu) \) which is not drawn in the outer face of \( D(G(\nu)) \). Hence, any simple region that contains \( D(G(\nu)) \) must also contain \( v \), contradicting the c-planarity properties.

Consider now a planar drawing of \( G \), such that for each node \( \nu \) in \( T \), \( G - G(\nu) \) is drawn in the outer face of the drawing of \( G(\nu) \). It remains to add cluster boundaries. Since \( G(\nu) \) is connected for each \( \nu \) in \( T \), the outer face is bounded by a - not necessarily simple - cycle. Thus, cluster boundaries are constructed recursively, following \( T \) from bottom to top, along their external facial cycle.

Based on Theorem 8.16, Feng et al. (1995) developed an algorithm which constructs a c-planar embedding of a hierarchical clustered graph, that is a circular ordering of the incident edges ordered around each cluster. The algorithm applies the PQ-tree technique presented by Booth and Lueker (1976) and modified by Chiba et al. (1985) and takes time \( O(n^2) \) under the additional condition, that each non-leaf node of \( T \) has at least two children.

It tries to embed the subgraph \( G(\nu) \) induced by each cluster \( V(\nu) \) recursively, following \( T \) from bottom to top. To guarantee the conditions in Theorem 8.16 for each virtual edge \( e \in E \), that is an edge such that \( e \cap V(\nu) = \{v_\nu\} \) has cardinality one, an additional vertex \( w_e \) and an edge \( \{v_\nu, w_e\} \) is added to \( G(\nu) \). Further one of the additional vertices is connected to all other additional vertices (see Figure 8.8).

![Graph](image)

**Fig. 8.8.** a) Graph \( G(\nu) \) with virtual edges is transformed into graph b). Additional vertices are shaded light grey. c) A wheel graph with 6 vertices on the rim.

To determine whether the embeddings of the children of a cluster \( \nu \) can be combined to an embedding of \( G(\nu) \), the graph \( G(\mu) \) for each child \( \mu \) of \( \nu \) is replaced by a representative graph which is more or less constructed by
replacing 2-connected components in \(G(\mu)\) by wheel graphs. A wheel graph consists of a vertex called hub and a simple cycle called rim, such that the hub is connected to every vertex on the rim (see Figure 8.8c). They showed that a representative graph with given ordering of the virtual edges can always be embedded in such a way that the rims are in the outer face without changing the ordering of the virtual edges.

### 8.3.1 Straight-Line Drawings with Convex Clusters

For a given c-planar embedding of a connected hierarchical clustered graph \(C = (G, T)\), Eades et al. (1996a) gave an algorithm to construct a drawing of \(C\) such that the edges of \(G\) are drawn as straight lines and the regions are convex. This drawing of \(C\) can be constructed in time \(O(n^2 \log n)\) which is dominated by the time needed for constructing the convex hull of the clusters.\(^1\)

The algorithm works as follows. First, graph \(G\) is triangulated. Then an \(st\)-numbering\(^2\) of the vertices of \(G\) is computed such that vertices in the same cluster are numbered consecutively. Such a numbering is called \(c-st\)-numbering. These numbers are now used as a layer assignment – thus, there is one vertex per layer – and an algorithm for constructing planar straight-line drawings of layered graphs, which is also presented by Eades et al. (1996a), is applied to draw the graph. Since each cluster has consecutive layers, the convex hull of its vertices satisfies all conditions of a region in a c-planar drawing.

Apart from the construction of a planar straight-line drawing of layered graphs, the critical part of this method is the construction of the \(c-st\)-numbering. To ensure that the vertices of the same cluster are numbered consecutively, Eades, Feng and Lin used a top-down approach, ordering the children of the root of \(T\) first and thus having a lexicographical numbering on the clusters. To compute an order of the child cluster of \(v\), an auxiliary graph \(F(v)\) is computed from \(G(v)\) by shrinking each child cluster to a vertex. If \(v\) is the root of \(T\), an edge \(\{s, t\}\) not belonging to any child cluster of \(T\) is chosen and an \(st\)-numbering is computed.

If \(v\) is not the root, vertices \(\sigma\) and \(\tau\) are added to the auxiliary graph \(F(v)\). For a virtual edge \(\{v, w\}\) with \(v \in V(v)\), let \(\mu\) be the lowest ancestor of \(v\) with \(w \in V(\mu)\) and \(k\) the number of the child cluster of \(\mu\) which \(w\) belongs to. If \(g(\nu) < k\) then edge \(\{v, \tau\}\) is added to \(F(v)\). Otherwise, edge \(\{\sigma, v\}\) is added. In case \(s \in V(v)\), the vertex representing the cluster containing \(s\) is set to be \(\sigma\); similarly for \(t\) and \(\tau\). Now a \(c-st\)-numbering is computed.

The only thing missing now is that the auxiliary graphs are 2-connected. This is a consequence of the following lemma.

---

\(^1\) In Eades et al. (1998), the time complexity is improved to \(O(n^2)\).

\(^2\) For the definition of \(st\)-numbering see Definition 2.9 on page 27.
Lemma 8.17. For every non-root node \( v \) of the inclusion tree of a connected c-planar hierarchical graph \( C = (G, T) \) with triangulated \( G \), the subgraph of \( G \) induced by \( V \setminus V(\nu) \) is connected.

Proof. Suppose that the subgraph of \( G \) induced by \( V \setminus V(\nu) \) has \( k \geq 2 \) components denoted by \( F_1, \ldots, F_k \). Since \( G \) is triangulated, it has a unique planar embedding. By Theorem 8.16, all vertices of \( G - G(\nu) \) are in the same face of \( G(\nu) \). Since \( G \) is connected, there is a face \( f \) of \( G \) such that its boundary contains an edge connecting \( G(\nu) \) and \( F_i \) and also an edge connecting \( G(\nu) \) and \( F_j \) for a \( j \neq i \). Because \( G \) is triangulated, \( f \) also contains an edge connecting \( F_i \) and \( F_j \), a contradiction.

Unfortunately, there are hierarchical clustered graphs such that any c-planar straight-line convex drawing strategy results in poor area requirement and angular resolution. Eades, Feng and Lin gave a family \( C_n = (G_n, T_n) \) of clustered graphs which require area \( O(2^n) \) and have angles between two edges incident to a vertex in \( O(1/n) \). A sketch of the recursive construction of the underlying graphs \( G_n \) can be seen in Figure 8.9. The root of \( T_n \) is adjacent to two nodes \( A \) and \( B \), which have children \( a_1, \ldots, a_n \) and \( b_1, \ldots, b_n \), respectively.

![Diagram](image)

Fig. 8.9. Example for poor vertex and angular resolution.

8.3.2 Orthogonal Drawings with Rectangular Clusters

Eades and Feng (1997) gave an algorithm to construct a drawing of a hierarchical clustered graph \( C = (G, T) \) with fixed embedding and degree at most 4 such that \( G \) is drawn orthogonal and the regions are rectangles. Using the constrained visibility representation, the algorithm takes time \( O(n^2) \), the drawing space \( O(n^2) \), and each edge has at most 3 bends.\(^3\)

The algorithm works as follows: First triangulate \( G \) and compute a c-st-numbering as constructed in the previous section. Orient the edges from lower

\(^3\) In Eades et al. (1999), the time complexity is improved to \( O(n) \).
to higher numbers. Construct a directed graph $G'$ from the oriented triangulation of $G$ by adding four additional dummy vertices and replacing virtual edges for each cluster to ensure rectangular regions. Construct a constraint visibility representation of $G'$ for a suitable set of non-intersecting paths. Construct an orthogonal drawing of $G$ from the visibility representation of $G'$ and finally reduce some bends.

How to construct a planar orthogonal drawing from a constraint visibility representation for non-clustered graphs is, for example, explained in Di Battista et al. (1999) Section 4.9 on page 130. A short introduction to this topic is also given in Section 6.4.2. So it remains to give the construction of $G'$ and the additional constraints. Proceeding from the leaves to the root of $T$, for

![Diagram](image)

**Fig. 8.10.** Virtual edges are bunched together.

each non-leaf node $v$ of $T$ add four dummy vertices denoted by $b(v)$ (bottom), $t(v)$ (top), $l(v)$ (left), and $r(v)$ (right) to $G(v)$ and split virtual edges $(v, w)$ of $G(v)$ by a dummy vertex in the following way (illustrated in Figure 8.10):

- If $v \in V(v)$ replace $(v, w)$ by $(v, t(v))$ and $(t(v), w)$.
- If $w \in V(v)$ replace $(v, w)$ by $(v, b(v))$ and $(b(v), w)$.

Add edges $(b(v), r(v))$, $(r(v), t(v))$, $(b(v), l(v))$ and $(l(v), t(v))$ to $G(v)$.

For a node $v \neq s$ on the way from $s$ to the root of $T$, let $u$ be the child of $v$ on this way. If $u \neq s$, add edge $(b(v), b(u))$, else add $(b(v), s)$. Similarly, for a node $v \neq t$ on the way from $t$ to the root of $T$, let $u$ be the child of $v$ on this way. Add edge $(t(u), t(v))$ respectively $(t, t(v))$. By this construction, $G'$ is a planar st-graph with $O(n)$ vertices and $O(n^2)$ edges.

Now, the alignment requirements in $G'$ for the visibility representation, which is a set of paths $\phi$, is specified. For each non-leaf node of $T$, the set $\phi$ contains the paths $(b(v), l(v), t(v))$ and $(b(v), r(v), t(v))$. Intercluster edges in $G$ are replaced by paths in $G'$. These paths are also added to $\phi$. Finally,
some paths containing edges incident to vertices of $G$ are added to $\phi$ to avoid unnecessary bends like in the non-clustered version. Thus, $\phi$ is a set of non-intersecting paths in the sense defined below and a constraint visibility representation can be computed.

**Definition 8.18 (Set of Non-Intersecting Paths).** Two paths $p_1$ and $p_2$ of a planar graph $G$ with given embedding are said to be non-intersecting if they are edge disjoint and there is no vertex $v$ of $G$ with edges $e_1, e_2, e_3,$ and $e_4$ incident to $v$ in this clockwise order around $v$, such that $e_1$ and $e_3$ are in $p_1$ and $e_2$ and $e_4$ are in $p_2$. A set of pairwise non-intersecting paths of $G$ is called a set of non-intersecting paths.

For each non-leaf node $v$, the rectangle bounded by the drawing of the corresponding vertices $b(v)$ and $t(v)$ and the drawing of the paths $(b(v), r(v), t(v))$ and $(b(v), l(v), t(v))$ is defined to be $D(v)$.

Having at most three bends per edge is as good as it gets: Eades and Feng gave a family $C_n = (G_n, T_n)$ of examples for hierarchical clustered graphs such that in every c-planar orthogonal drawing with rectangular clusters, there are at least $O(n)$ edges that bent more than twice. $G_n$ is a sequence

![Diagram](https://via.placeholder.com/150)

**Fig. 8.11.** Example for a lot of bends.

of $n$ copies of the graph $H$ shown in Figure 8.11 such that vertex $a_7$ of a previous copy of $H$ serves as vertex $a_1$ of the next copy. It is partitioned into two clusters. Cluster $A$ containing the $a$-vertices and cluster $B$ containing the $b$-vertices. The embedding is as sketched in the figure. Each copy of $H$ has at least one edge with more than two bends: At least one of the edges $\{a_4, b_1\}$ and $\{a_1, b_4\}$ has two or more bends in the cluster region of $A$. Suppose it’s $\{a_4, b_1\}$. Then at least one of the edges $\{a_4, b_1\}$ and $\{a_1, b_1\}$ has three or more bends. Thus $G_n$ has at least $n$ such edges, but $10n + 1$ vertices.

### 8.3.3 Multilevel Visualization of Clustered Graphs

Eades and Feng (1996) show a way to represent both the adjacency and inclusion relations of a clustered graph in the same drawing. Here, the inclusion
8. Drawing Clusters and Hierarchies

relation is not only just drawn as simple regions containing the drawing of their corresponding vertices, but as a tree structure that also geometrically visualizes this relation. As graphs get larger and larger it is a common strategy to visualize them at multiple abstraction levels. If the graph has a recursive clustering it is a natural approach to take the clustering of the graph as abstraction levels, which provides the possibility to zoom in and out within the clustered structure of the graph. The method presented in Eades and Feng (1996) is a three dimensional representation of the clustered graph with each cluster level drawn at a different z-coordinate, and with the inclusion relation drawn as a tree in three dimensions. This kind of representation also keeps track of the abstractions from one level to the next.

**Terminology.** The height of a cluster $v$, denoted by $\text{height}(v)$, is defined as the depth of the subtree of $T$ rooted at $v$.

For a clustered graph, its *view at level* $i$ is a graph $G_i = (V_i, E_i)$ where $V_i$ consists of the set of nodes of height $i$ in $T$. There is an edge $(\mu, \nu)$ in $E_i$ if there exists an edge $(u, v)$ in $E$ where $u$ belongs to cluster $\mu$ and $v$ belongs to cluster $\nu$; in other words, edge $(\mu, \nu)$ is an abstraction of all edges between cluster $\mu$ and $\nu$.

In a *plane drawing* of a clustered graph, the vertices are drawn as points and edges as curves in the plane as usual. Each cluster $\nu \in T$ is drawn as a simple closed region $R$ that contains the drawing of $G(\nu)$, as defined in Definition 8.14. If a clustered graph has a c-planar representation (Definition 8.15), then it is c-planar (Figure 8.12).

**Fig. 8.12.** A plane drawing of a clustered graph; the graph shown here is c-planar.

**Multilevel Drawings.** A *multilevel drawing* (Fig. 8.13) of a clustered graph $C = (G, T)$ consists of:

- a sequence of plane drawings of representations from the leaf level (level 0) to the root level of $T$, where the view of level $i$ is drawn on the plane $z = i$.

- A three dimensional representation of $T$, with each node $\nu \in T$ of height $i$ drawn as a point on the plane $z = i$, and within the region of $\nu$ in the drawing of a view of that level.
For the plane drawing of a clustered graph several already presented automatic drawing algorithms can be used, such as the straight-line convex or orthogonal rectangular drawing methods (Section 8.3.1 and Section 8.3.2, see also Feng (1997)).

Here, the focus is only on the construction of the multilevel drawing which consists of two steps, the construction of view drawings for each level \( i, i = 0, \ldots, \text{height(root of } T) \), and the drawing of the inclusion tree \( T \).

1. **View drawing for each level:**

   1. For each level \( i, i = 0, \ldots, \text{height(root of } T) \), construct a plane drawing and translate it to the plane \( z = i \), starting at the leaf level.
   2. An edge \((\mu, \nu)\) in level \( i + 1 \) is an abstraction for all edges between cluster \( \mu \) and \( \nu \) in level \( i \). Choose one of these edges \((u, v)\) between cluster \( \mu \) and \( \nu \) as a representative edge, and derive the drawing of edge \((\mu, \nu)\) in the view of level \( i + 1 \) from the drawing of edge \((u, v)\) in the view of level \( i \). In the two-dimensional plane, cluster \( \mu \) and \( \nu \) are drawn as simple closed regions \( R(\mu) \) and \( R(\nu) \); the drawing of edge \((u, v)\) intersects the boundaries of these regions at points \( x \) and \( y \) (Figure 8.14). To construct the drawing of edge \((\mu, \nu)\) in the view of level \( i + 1 \), use the segment between \( x \) and \( y \) and translate it to the plane \( z = i + 1 \).

---

**Fig. 8.13**. Multilevel drawing.

**Fig. 8.14**. Deriving a drawing for abstraction edges.
2. 3D drawing of the inclusion tree $T$:

General results on 3D drawings can be found in Chapter 7. To find a three-dimensional drawing of the inclusion tree, every node $\mu \in T$ with height $i$ has to be placed on the plane $z = i$ and it also must be positioned in the corresponding region $R(\mu)$. This is achieved as follows:

1. Compute the position of the nodes of the inclusion tree recursively:
   - $Level\ i = 0$: (leaf level)
     Take the positions as computed in the plane drawing of level 0;
   - $Level\ i > 0$:
     Let the position for node $\mu \in T$ be the average of all $xy$-coordinates of its children from level $i - 1$;
2. Route the inclusion edges as straight line segments between the corresponding nodes.

8.4 Hierarchical Representation of Compound Graphs

Sugiyama and Misue (1991) introduce an extension to the class of clustered graphs, the class of compound digraphs, and they also present an algorithm to produce an automatic hierarchical representation of compound digraphs. The main difference between these two graph classes is the use of the inclusion relation. A clustered graph is a graph with a partition of its vertex set into clusters. In the representation of a clustered graph, the cluster regions are drawn as simple closed regions that contain the drawing of all the vertices belonging to that cluster; the inclusion relation is restricted to these cluster regions, and there are no edges connecting them. In a compound digraph, the inclusion relation as well as the adjacency relation is defined on the same set of vertices.

**Definition 8.19 (Compound Graph).** An inclusion digraph is a pair $D_c = (V,E)$ where $E$ is a finite set of inclusion edges whose element $(u,v) \in E$ means that $u$ includes $v$ (Figure 8.15 (a)).

An adjacency digraph is a pair $D_a = (V,F)$ where $F$ is a finite set of adjacency edges whose element $(u,v) \in F$ means that $u$ is adjacent to $v$ (Figure 8.15 (b)).

A compound digraph is defined as a triple $D = (V,E,I)$ obtained by compounding these two digraphs (Figure 8.15 (c)).

In Sugiyama and Misue (1991), the inclusion digraph $D_c$ is required to be a rooted tree and is also called the inclusion tree of $D$. The depth of a vertex $v \in V$ is the number of vertices on the path between $v$ and the root of $D_c$ and is denoted by $depth(v)$, with $depth(root) = 1$, where root denotes the root of $D_c$. The parent of a vertex $v \in D_c$ is denoted by $Parent(v)$. 
Fig. 8.15. (a) Inclusion tree $D_c$  (b) Adjacency graph $D_a$  (c) The compound digraph $D$ obtained from (a) and (b).

Fig. 8.16. The representation of a compound digraph from Figure 8.15 (c) and its compound levels.
Figure 8.16 shows a representation of the compound digraph and its compound levels of Figure 8.15 (c). The adjacency edges drawn with solid lines have downward orientation and edges drawn with broken lines upward. The vertices of the compound digraph are drawn as rectangles. The inclusion relation \((u, v) \in E\) is realized as the rectangle representing vertex \(u\) is inside the rectangle representing vertex \(v\). The conventions for the drawing of compound digraphs are specified more precisely below.

### 8.4.1 Conventions

#### Drawing conventions.

**C1:** *Vertex Shape:*
A vertex is drawn as a rectangle with horizontal and vertical sides.

**C2:** *Inclusion:*
An inclusion edge \((u, v)\) is drawn in such a way that the rectangle corresponding to \(u\) includes geometrically the rectangle corresponding to \(v\).

**C3:** *Hierarchy:
Vertices are laid out hierarchically in terms of both inclusive and adjacent relations on parallel-nested horizontal bands, called compound levels.

**C4:** *Down-Arrow:*
An adjacency edge \((u, v)\) is drawn as a downward arrow with possible bends, originating from the bottom side of the rectangle corresponding to \(u\) and terminating on the top side of the rectangle corresponding to \(v\).

#### Drawing Rules.
To enhance the readability and the aesthetic of the drawing, the following objectives should be satisfied as much as possible.

**R1:** *Closeness:*
Connected vertices are laid out as close as possible to each other.

**R2:** *Edge Crossings:*
The number of crossings between adjacency edges is reduced as much as possible.

**R3:** *Edge-Rectangle Crossings:*
The number of crossings between adjacency edges and the vertex rectangles is reduced as much as possible.

**R4:** *Line-Straightness:*
One-span adjacency edges, i.e., edges between adjacent levels, are drawn as straight lines, whereas long-span adjacency edges are drawn as polygonal lines with as few bends as possible.

**R5:** *Balancing:*
Edges originating from and ending at a vertex rectangle are laid out in a balanced form.

The above rules specify topological and metrical layout properties; their priority is top-down.
8.4.2 The Layout Algorithm

The algorithm consists of the following four steps that are similar to those of an algorithm by Sugiyama et al. (1981) and Sugiyama (1987) for the layered representation of a general digraph. Due to the complex structure of a compound digraph, these steps must be modified and extended, in particular to display the inclusion relations. A brief description of the four steps follows, but the focus will be mainly on the ideas and the modifications that have to be done to the original Sugiyama algorithm (see Chapter 5). A detailed description of this algorithm is given in Sugiyama and Misue (1991).

Step 1: Hierarchization

Due to the two kinds of relations existing in a compound graph, this step is different to the one in Sugiyama’s original algorithm for general graphs, so it will be explained in more detail.

A. Compound Level Assignment

In this step, the vertices of the compound digraph are assigned to compound levels to satisfy the drawing conventions. This level assignment places the vertices on parallel-nested horizontal bands. As shown in Figure 8.16, the compound levels can be expressed by the assignment of a sequence of positive integers to every vertex \( v \in V \).

Let \( \Sigma = 1, 2, 3, \ldots \) and \( \Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots \), and suppose that a lexicographical ordering is introduced for elements of \( \Sigma^+ \), e.g., \((1, 1, 2) < (1, 2, 1) < (1, 2, 2) \). Then the problem of assigning compound levels to the vertices of \( D \) is to find a mapping \( c:\text{-level} : V \rightarrow \Sigma^+ \) satisfying the Inclusion and Down-Arrow conventions (C2, C4).

The Inclusion convention (C2) can be expressed as follows:

\[
\Pi : \forall v \in V : c:\text{-level}(v) \in \Sigma^\text{depth}(v) \\
\Pi_2 : \text{For any inclusion edge } (v, w) \in E : c:\text{-level}(w) = \text{append}(c:\text{-level}(v), s); \\
\quad \quad \quad \quad s \in \Sigma \text{ and append is a function that appends a component to a sequence.}
\]

The Down-Arrow convention (C4) is more complicated: For any adjacency edge \((v, w) \in F\) there is a unique path \( P \) from \( v \) to \( w \) in the inclusion tree \( D_e \):

\[
P : \{v = p_m, p_{m-1}, \ldots, p_1, t, q_1, q_2, \ldots, q_{n-1}, q_n = w\},
\]

where \( t \) is the top vertex, i.e., \( t \) has minimal depth. \( P \) originates from the rectangle of \( v \), goes out across \( p_{m-1}, \ldots, p_1 \), passes \( t \), goes in across \( q_1, q_2, \ldots, q_{n-1} \), and terminates on \( w \). To formulate the Down-Arrow convention, the order among each pair \((p_i, q_i)\) of vertices that have the same depth for any adjacency edge \((v, w) \in F\) must be specified as follows:

D1: if \( \text{depth}(v) > \text{depth}(w) \) (or \( m > n \)),

\[
\begin{align*}
(a) \quad & c:\text{-level}(p_i) \leq c:\text{-level}(q_i), \quad i = 1, \ldots, n - 1 \\
(b) \quad & c:\text{-level}(p_m) < c:\text{-level}(w)
\end{align*}
\]
D2: if $\text{depth}(v) \leq \text{depth}(w)$ (or $m \leq n$),

(a) $\text{c-level}(p_i) \leq \text{c-level}(q_i)$, $i = 1, \ldots, m - 1$

(b) $\text{c-level}(v) < \text{c-level}(q_m)$.

For example, in Figure 8.15, the path corresponding to adjacency edge $(j,l)$ is $j,e,b,f,l$ where $b$ is the top vertex. Since $\text{depth}(j) = \text{depth}(l)$, we have $\text{c-level}(e) \leq \text{c-level}(f)$ and $\text{c-level}(j) < \text{c-level}(l)$ from D2.

A compound digraph has a compound level assignment if and only if there exists a mapping $\text{c-level}: V \rightarrow \Sigma^+$ satisfying II, I2, D1, and D2.

B. Hierarchization Algorithm

A hierarchical map of the graph can not always be determined because the digraph might be cyclic. If there are cycles in $D$, some of the adjacency edges need to be reversed in order to obtain a hierarchization. Because the problem of finding this minimum set of feedback adjacency edges is \text{NP}-complete (Garey and Johnson, 1991; Lempel and Cederbaum, 1966), heuristics are introduced for determining the edges that have to be reversed (see also Section 5.2).

In order to meet the requirements D1 and D2, every adjacency edge of $D$ is replaced with one of the two following types of adjacency edges, $\rightarrow$ and $\Rightarrow$, which represent the relations $<$ and $\leq$, respectively, in D1 and D2. If edges between the same pair of vertices are duplicated during the replacement, reducing rules such as $\rightarrow = \rightarrow + \rightarrow, \Rightarrow = \Rightarrow + \Rightarrow$, and $\rightarrow = \Rightarrow + \Rightarrow$ are applied to determine the resulting edge type. The graph derived by this edge replacement is called the derived graph of $D$. An adjacency edge in the derived graph is called an original edge, if $e \in F$, derived edge otherwise (Figure 8.17). Note that in the derived graph of $D$, every adjacency edge connects two vertices with identical depth. The derived graph of $D = (V, E, I)$ is denoted by $DD = (V, E, ID, \text{type})$, where $ID$ is the derived set of adjacency edges and $\text{type}: ID \rightarrow \{ \rightarrow, \Rightarrow \}$.

![Diagram](image_url)

Fig. 8.17. The derived graph (a) and the assigned compound digraph (b) obtained from the compound graph of Figure 8.15 (c); edges in (a) marked with asterisk (*) are derived edges.
Now, the Down-Arrow convention is established and the compound levels can be assigned to the vertices. If $DD$ is not cycle-free, the cycles have to be resolved. To do this, the strongly connected components of $DD$ are investigated, and the cycles are destroyed by either contracting strongly connected components into a so-called proxy vertex or by deleting some of the adjacency edges. This procedure leads to a cycle-free, hierarchical graph to which one finally can assign the compound levels. Of course, the deleted edges have to be put back in again, as well as the proxy vertices must be replaced again later on. All vertices of the component of $DD$, which have been contracted to a proxy vertex, are assigned the same compound level. Every adjacency edge $(v, w)$ of the compound digraph $D = (V, E, I)$ is now checked, whether $c$-level$(v) < c$-level$(w)$. If this does not hold, the direction of that edge is reversed. The result is an assigned compound digraph $DA = (V, E, IA, c$-level$)$ (Figure 8.17 (b)).

**Step 2: Normalization**

In an assigned compound digraph $DA$, an adjacency edge $(v, w) \in IA$ is said to be proper if and only if $c$-level$(\text{Parent}(v)) = c$-level$(\text{Parent}(w))$ and $\text{tail}(c$-level$(v)) = \text{tail}(c$-level$(w)) - 1$, where tail is a function, that returns the last number of the $c$-level-string as an integer. The assigned compound digraph $DA$ is now transformed into a proper compound digraph by replacing every non-proper adjacency edge with appropriate dummy vertices, dummy inclusion edges and dummy proper adjacency edges (cf. Sugiyama and Misue (1991) for more detail).

**Step 3: Vertex Ordering**

The idea of this step is similar as in Sugiyama's original algorithm for general graphs. The horizontal order of the vertices per level is determined by permuting their order on each level in such a way that the drawing rules Closeness, Edge Crossings, and Edge-Rectangle Crossings (R1 – R3) are satisfied as much as possible. The problem of minimizing edge crossings is $\mathcal{NP}$-complete even for only two levels (Garey and Johnson, 1991). Minimizing of edge-rectangle crossings, which is equivalent to the linear arrangement problem, is also $\mathcal{NP}$-complete (Garey and Johnson, 1991). Hence, heuristics, i.e., barycentric ordering as in Sugiyama's algorithm, are used to accomplish these tasks. In a compound digraph there exist also local hierarchies due to the inclusion relation, so vertex ordering must also be applied to these local hierarchies, i.e., subtrees of $D_L$. This step leads to an ordered compound digraph.

**Step 4: Metric Layout**

In this last step, the positions of vertices (i.e., horizontal and vertical positions, widths and heights of rectangles) are determined by attaining the Closeness, Line-Straightness, and Balancing rules (R1, R4, R5) as much as possible. This problem can be expressed as a quadratic programming problem; a heuristic called the priority layout method is also developed to solve the
problem. Once the vertex positions are determined, a routing for the edges can easily be achieved. The orientation of the reversed edges is changed back again, and all inserted dummy vertices and dummy edges are deleted and their corresponding originals are rearranged. This step finally leads to an automatic drawing of the original compound digraph.

8.5 Force-Directed Methods for Clustered Graphs

Force-directed graph drawing methods (cf. Chapter 4) can also be adopted to support and show the structure of a clustered graph. In the following sections we will see different ways of adaptation with different design goals.

One possibility to receive a more structured layout for clustered graphs is to decide between different spring forces. In Sections 8.5.2 and 8.5.3 we show two different approaches for an expanded force model for clustered graphs.

8.5.1 Inserting Dummy Vertices

The easiest way to achieve clustering is to insert dummy vertices as follows (Figure 8.18):

1. Let $G = (V, E)$ be a graph with a partition $(C_1, C_2, \ldots, C_k)$ on the vertex set $V$. For each $C_i$, $1 \leq i \leq k$, add a dummy attractor vertex $e_i$ to the graph.
2. Add attractive forces between an attractor $e_i$ and each vertex of the corresponding cluster $C_i$.
3. Add repulsive forces between pairs of attractors and between attractors and vertices not belonging to any cluster (i.e., if $\bigcup_{i=1}^{k} C_i \subseteq V$).

In this approach, no new forces have to be added. After inserting these attractors the vertices within a cluster will be closer to each other than before, and the distance between the clusters will grow.
8.5.2 Interactive Clustering

Huang and Eades (1998a) describe an animated interactive system for clustering and navigating huge graphs, called DA-TU, where they use the following expanded force model consisting of three different spring forces (Figure 8.19):

- **internal-spring**
  A spring force between a pair of vertices that belong to the same cluster.

- **external-spring**
  A spring force between a pair of vertices that belong to different clusters.

- **virtual-spring**
  In each cluster there is a virtual vertex (black vertices in Figure 8.19) that is connected to all vertices belonging to the same cluster by virtual edges; this is a similar approach to the concept of *attractors* described above. A virtual spring force exists between a vertex and a virtual vertex along a virtual edge.

![Figure 8.19. Expanded spring model.](image)

Additionally there is a gravitational repulsion force between each pair of vertices. All forces are applied additively to each vertex.

Some of the features of the DA-TU system are worth to be mentioned here because they show a possible application for clustering:

- The user can interactively change the graph.
- The user can interactively change the clustering of the graph.
- All transitions from one state of the graph to another are animated.
- Clusters can be interactively *contracted* and *expanded*, respectively. This is in particular useful for large graphs that do not fit on the screen or are too large to comprehend, so the clustered structure is used to navigate through the graph. If some of the clusters are contracted, DA-TU draws a so-called *abridgement* of the given graph.


**Definition 8.20 (Abridgement, Ancestor Tree).** A clustered graph \( C' = (G', T') \) is an abridgement of the clustered graph \( C = (G, T) \) if \( T' \) is an ancestor tree of \( T \) with respect to a set \( U \) of nodes of \( T \) and there is an edge between two distinct nodes \( u \) and \( v \) of \( G' \) if and only if there is an edge in \( G \) between a descendant of \( u \) and a descendant of \( v \).

In other words, the ancestor tree \( T' \) is a subtree of \( T \) consisting of all nodes and edges on paths between elements of \( U \) and the root. Figure 8.20 shows such an ancestor tree for the set \( U \) of black nodes as the shaded area of the original inclusion tree.

**Fig. 8.20.** The shaded area is the ancestor tree of the set of black nodes.

### 8.5.3 Meta Layouts

A similar approach is taken by Wang and Miyamoto (1995). They also use three forces but in a slightly different way. Instead of inserting virtual vertices as attractors in each cluster they use the concept of a *meta-graph*.

First, the edge set of the given graph is divided into *intra-edges*, i.e., edges between vertices belonging to the same cluster, and *inter-edges*, i.e., edges between vertices belonging to different clusters.

The forces in the force-directed placement are also divided into two categories:

- *intra-force*
  A spring force between a pair of vertices that belong to the same cluster.
- *inter-force*
  A spring force between a pair of vertices that belong to different clusters.
A force-directed placement is constructed by applying the intra- and inter-
forces. An undirected graph $G_{\text{meta}}$ is then constructed by collapsing the
clusters of $G$ into meta-vertices and transforming inter-edges of $G$ existing
between a pair of clusters into one meta-edge each. This sounds similar to
the concept of a quotient graph (Definition 8.3), but goes further than that.
A layout for $G_{\text{meta}}$ is called meta-layout of $G$ and can be obtained from
the force-directed placement where the dimensions and center of each meta-vertex
are set to the dimensions and center of the underlying subgraph, respectively
(Figure 8.21). To calculate the forces between the meta-vertices, an improved
force-directed placement is used that takes the different vertex-sizes into
account (Wang and Miyamoto, 1995). The net force on a meta-vertex is defined
as the meta-force on all vertices contained in the cluster represented by that
meta-vertex.

Fig. 8.21. Meta-graph and meta-layout.

In Figure 8.21 a force-directed layout and a partition are given on the left
side, the corresponding meta-layout is shown on the right. The forces that
are applied to vertex $c$ are:

- The intra-force on $c$ is the sum of forces between $c$, $a$ and $c$, $b$
- The inter-force on $c$ is the sum of forces between $c$ and the vertices of
  subgraphs $S_2$ and $S_3$
- The meta-force on $c$ is the net force on meta-vertex $S_1$ in the meta layout.

To finally compute a drawing of the clustered graph, Wang and Miyamoto

**Divide-and-Conquer Drawing Approach.** A divide-and-conquer draw-
ing algorithm would draw a clustered graph in the following three steps:

1. divide the graph into subgraphs (cluster);
2. draw the subgraphs;
3. compose the subgraph layouts together to form the resulting layout.

The problem with this approach is that inter-edges are not taken into account which may result in a drawing with many crossings between inter-edges or long inter-edges.

In Wang and Miyamoto (1995), the last two steps of this divide-and-conquer approach are combined within one force-directed placement algorithm by using the following composite force $F_{\text{comp}}$ to position a vertex:

$$F_{\text{comp}} = F_{\text{intra}} + S(t)F_{\text{inter}} + (1 - S(t))F_{\text{meta}}$$

where $F_{\text{intra}}$, $F_{\text{inter}}$, $F_{\text{meta}}$ are the intra-, inter- and meta-force on a vertex, respectively, and $S(t) \in [0, 1]$ is a function of layout time $t$ such that $S(t)$ decreases as $t$ increases after a threshold $t'$ and reaches 0 at another threshold $t'' > t'$.

By applying this composite force, the force-directed placement can be divided into three phases:

1. **Between time 0 and time $t'$**: $S(t) = 1 \Rightarrow F_{\text{comp}} = F_{\text{intra}} + F_{\text{inter}}$.
   The force-directed placement leads to a layout with uniform edge lengths and a small number of edge crossings, as shown in Figure 8.22 (a).

2. **Between time $t'$ and time $t''$**: $S(t)$ decreases:
   The strength of the inter-forces is reduced while the strength of the meta-forces is increased at the same time.

3. **at time $t''$**: $S(t) = 0 \Rightarrow F_{\text{comp}} = F_{\text{intra}} + F_{\text{meta}}$.
   Inter-forces do not count anymore; the intra-forces keep the vertices contained in the same cluster close together while the meta-forces fix the final positions of the clusters and eliminate possible overlaps between clusters. The resulting structured layout is shown in Figure 8.22 (b).

Wang and Miyamoto (1995) also present a way to add layout constraints to their force-directed placement algorithm by integrating a constraint solver.

**Integration of a Constraint Solver.** Layout constraints of the three following types can occur:

- **absolute constraints**, to fix an absolute vertex position
- **relative constraints**, to constrain the position of a vertex in relation to other vertices
- **cluster constraints**, to cluster several vertices into a subgraph that can be processed as a whole.

While trying to solve given constraints, some vertices may block others from reaching their optimal positions calculated by the force-directed placement algorithm. This may lead to a poor layout.
If a constraint for two vertices $A$ and $B$ is given that prevents vertex $A$ from reaching its optimal position, then vertex $B$ is called a barrier for vertex $A$. Figure 8.23 shows several examples of barriers; the arrows indicate the direction and the strength of the forces calculated by the layout algorithm, the corresponding constraint shown below would be violated if these movements would be performed. In Figure 8.23 (a) no movement of either vertex is possible without violating the corresponding constraint. In the other two examples, the vertices could at least be partially moved until one of the vertices becomes a barrier for the other.

To avoid barriers while at the same time improving the layout, the vertices could be moved together without changing their relative positions. This is done by introducing rigid sticks to represent constraints in the force-directed placement. If vertex $v_1$ becomes a barrier for vertex $v_2$, a rigid stick is introduced between them so that they have to move like one rigid object. The movements of $v_1$ and $v_2$ are determined by the weighted average of the forces that are working on them:

$$f = \frac{w_1 f_1 + w_2 f_2}{w_1 + w_2}$$

where $f$ is the new resulting force on $v_1$ and $v_2$, $f_1$ and $f_2$ are the old forces on $v_1$ and $v_2$, respectively, and $w_1$ and $w_2$ are weights of $v_1$ and $v_2$, respectively.
The layout algorithm and the solver cooperate to solve the given constraints as much as possible while at the same time keeping a good layout resolution. This cooperation works in an iteration of the following four steps:

Step 1: Calculate forces;
Step 2: Introduce sticks and distribute forces;
Step 3: Calculate new positions;
Step 4: Satisfy constraints.

Steps 1 and 3 are performed by the layout algorithm, whereas the other two steps are performed by the solver. Figure 8.24 shows an example of applying constraints to the graph of Figure 8.22.

\[ d3.y = d5.y \]

\[ \text{abs}(a_4.x - a_2.x) \leq 120 \]
\[ \text{abs}(a_1.x - a_5.x) \leq 70 \]
\[ a_5.y = a_1.y = a_2.y - 64 \]
\[ a_4.y = a_3.y = a_2.y \]

\[ c_2.y = c_5.y = c_4.y = c_1.y \]

Fig. 8.24. Resulting layout of the graph of Figure 8.22 after adding the constraints on the left side.

8.6 Online Graph Drawing of Huge Graphs

- A Case Study

Traditional graph drawing algorithms assume that the given graph can be laid out in a readable and understandable way on the screen or on paper. But there are important situations where this assumption does not hold. Suppose for example the graph displaying parts of the WWW or graphs arising in information retrieval. These graphs can be very large and there is no way to fit them in a readable way on a display medium.
Most graph drawing systems approach the layout of huge graphs in the following way:

1. Layout the graph on a virtual and very large page.
2. Provide a smaller window with scroll bars to show the part of interest, and to allow the user to navigate through the graph.

However, some problems are involved in this approach:

- The whole graph may not be known, e.g., in distributed systems, where a local vertex only knows part of the graph.
- To explore the graph, the user can only move geometrically through the graph by the means of the scroll bars. But the user might want to explore the graph in a logical way, in particular, if the graph contains relational data or hyperlinks. Moreover, long edges that do not fit on the screen are hard to follow. A more user-oriented approach would be better; the user should be able to control the logical content of the display.
- There is no mental map (Eades et al., 1991) that helps the user to keep track of his exploration so far. Even worse, the user can not see the whole graph and might get lost in empty areas.
- Besides that, it costs a lot of memory to store and display the large virtual screen.

To deal with these problems, several techniques have been proposed (see Sarkar and Brown (1994); Eades et al. (1991); Nielsen (1990); Mukherjea et al. (1994); Robertson et al. (1993)). For example, in Sarkar and Brown (1994) the fish-eye view technique is described where a detailed picture of a subgraph is shown along with the so called context of that subgraph. This kind of view provides the user with more information about the position of the subgraph within the whole graph. Another approach are three-dimensional methods, such as cone trees (Robertson et al., 1993) which lead to an increase in density of information on the screen.

These techniques work effectively for graphs of moderately large size, but they can not be applied when the graph is not completely known. Moreover, they still predefine the geometry of the graph.

The aim of Online Graph Drawing is the visualization of huge graphs which may be partially unknown. At any time, a tiny but non-empty subgraph called the logical frame is known and displayed on the screen. The user can explore the huge graph by changing the logical frame.

The layout of such a logical frame has to satisfy the usual aesthetic criteria for drawing graphs, e.g., minimization of edge crossings and uniform vertex distribution. Additionally, the transition from one picture of a logical frame to the next should preserve the user's mental map (Eades et al., 1991), i.e., successive drawings should not differ much, so that the user can easily follow the change in the drawing and does not loose orientation in a completely different layout when changing the focus.
Eades et al. (1997b) describe a model of Online Graph Drawing as well as an instantiation of that model in a system for Online Force-Directed Animated Visualization (OFDAV) for assisting web navigation (see also Eades et al. (1997a)). An interesting part of that model is a new force-directed drawing algorithm, that can be used to produce a continuous sequence of layouts according to the above mentioned criteria.

In OFDAV, the view of the user is focused on a small subgraph, the logical frame, that is defined by a focus vertex \( v \). A force-directed graph drawing algorithm is used to draw this subgraph as well as its logical neighborhood. The user can change focus by selecting another vertex within the displayed frame which then becomes the new focus node, and the view changes according to this selection. Multiple animation steps are used to guide the user through the change of view and to preserve the mental map. A linear history is also kept by lining up a certain number of previously visited focus vertices.

The focus vertices together with their neighborhoods form a clustering of the graph.

The Online Graph Model. To explore a huge, partially unknown graph \( G = (V, E) \), a sequence of logical frames \( F_1 = (G_1, Q_1), F_2 = (G_2, Q_2), \ldots \) is used (Figure 8.25):

![Diagram of a graph G with logical frames F1, F2, ..., Fn](diagram.png)

Fig. 8.25. The path of exploration of a huge graph \( G \) by a sequence of logical frames.

Each logical frame \( F_i = (G_i, Q_i) \) consists of a connected subgraph \( G_i = (V_i, E_i) \) of \( G \) and a queue \( Q_i \) of focus vertices. Successive frames differ only by a few vertices. The sequence of logical frames represents the sequence of subgraphs that are viewed by the user of the system and is determined by the interaction of the user who can change the focus and thereby decides which new logical frame has to be displayed. To define the logical frame more precisely, we need to explain the concept of neighborhood.

Suppose that \( G = (V, E) \) is a graph, \( v \in V \), and \( d \) is a non-negative integer. Then, the distance-\( d \) neighborhood \( N_d(v) \) of \( v \) is the subgraph of \( G \)
induced by the set of vertices whose graph-theoretic distance from \( v \) is at most \( d \); note that \( v \in N_d(v) \). In GFDAV only distance-1 neighborhoods are used, so we write \( N(v) \) instead of \( N_1(v) \) in the following, and call it the neighborhood of \( v \).

**Definition 8.21 (Logical Frame, Focus Vertex).** Given a queue \( Q = (v_1, v_2, \ldots, v_s) \) of vertices of graph \( G = (V, E) \), the subgraph of \( G \) induced by the union of \( N(v_1), N(v_2), \ldots, N(v_s) \) is called a logical frame \( F = (G', Q) \) (Figure 8.26 (a)), with \( G' = (V', E') \) and

\[
V' = \bigcup_{i=1}^{s} N(v_i) \quad E' = \{(u, v) \in E \mid u, v \in V'\}.
\]

The vertices of the queue \( Q \) are the focus vertices of the logical frame \( F \).

**Clustering.** Suppose that \( Q = (v_1, v_2, \ldots, v_s) \) is the queue of focus vertices in \( G \). Each neighborhood \( N(v) \), for \( v \in Q \), can be divided into two parts, the common part \( C(v) \), and the local part \( P(v) \), defined as follows:

- \( C(v) \) is the part of the neighborhood \( N(v) \) that also occurs in the neighborhood of other focus vertices \( v' \neq v \), that is:

\[
C(v_j) = \bigcup_{i=1, i \neq j}^{s} N(v_j) \cap N(v_i).
\]

- \( P(v) \) is the part of the neighborhood \( N(v) \) that does not occur in the neighborhood of any other focus vertex \( v' \neq v \) (Figure 8.26 (b)), that is:

\[
P(v_j) = N(v_j) - \bigcup_{i=1, i \neq j}^{s} N(v_i).
\]

The user can explore the graph by changing the focus vertex, and this exploration is visualized by a sequence of logical frames. In practice, only a small number of vertices can be displayed on the screen at a time, in particular if the vertices are labeled, e.g., by the names of the HTML-pages as in GFDAV. Here, a global constant \( B \) is introduced as an upper bound for the length of the focus queue \( Q \). For www-graphs, small values of \( B \) between 7 and 10 ensure that there are about 20 to 60 vertices on the screen at a time.

The transition from one logical frame to the next is obtained by adding the new focus vertex with its local neighborhood. If the length of \( Q \) was already \( B \) before, then the last recently used focus vertex and its local neighborhood are deleted (FIFO policy).

The local neighborhoods of the focus vertices in each frame can be viewed as the clusters of this frame. This will become clearer in the next section where the force model is explained.
The Force Model. The force model is based on Eades (1984) and consists of a combination of Hooke’s law springs and Newtonian gravitational forces. In order to address the specific criteria of this online drawing approach, extra Newtonian gravitational forces among the neighborhoods, $N(v_i), N(v_{i+1}), \ldots, N(v_{i+R})$, of the focus vertices are added. These forces are used to separate the neighborhoods so that the user can visually identify the changes induced by changing the focus. This leads to a clustering of the displayed subgraph.

The total force applied to a vertex $v$ of a logical frame $F_i = (G_i, Q_i)$, with $G_i = (V_i, E_i)$ is

$$f = \sum_{u \in N(v)} f_{uv} + \sum_{u \in V_i} g_{uv} + \sum_{u \in Q_i} h_{uv}$$

where $f_{uv}$ is the force exerted on $v$ by the spring between $u$ and $v$, and $g_{uv}$ and $h_{uv}$ are the gravitational repulsions exerted on $v$ by one of the other vertices $u$ in $F_i$ (Figure 8.27).

The details of the modified force-directed drawing algorithm are given in Eades et al. (1997b). Here, we will only explain the idea of this approach.

The modified force model aims to satisfy the following four aesthetic criteria:

1. The spring force $f_{uv}$ between adjacent vertices is aimed to ensure that the distance between vertices $u$ and $v$ is approximately equal to the zero energy spring length.
2. The gravitational force $g_{uv}$ ensures that the vertices are not too close together and distributed evenly.
3. The extra gravitational force $h_{uv}$ aims to minimize the overlaps among the (local) neighborhoods within a logical frame. This also ensures that the next vertices that have to disappear are placed close together which makes the identifying of the deleted objects easier for the user.
4. $h_{uv}$ also aims to keep the layout of the queue of focus vertices close to a straight line; new vertices appear in one end of that line while old vertices disappear at the other end. This helps the user in understanding the direction of exploration of the huge graph.

8.7 Summary

For a graph that does not have a natural cluster-structure it is not at first glance clear what a good clustering strategy is. Often, the vertices are gathered together with respect to graph connectivity. Two main heuristics to do this are introduced in Section 8.2. One is to integrate cut size and cluster size balance within a single objective function, like the ratio cut partition, and to optimize them with a suitable procedure. Another one makes use of the eigenvalues of the Laplacian matrix of the graph. Besides connectivity, other graph properties like similarity of neighborhoods can also be of interest.

Once a graph has a cluster-structure, the question arises, how to make this structure visible. If we want to draw a really large graph, it seems to be a good method to draw only the quotient graph. But sometimes one is interested to also see what happens within a cluster. In Section 8.3 and Section 8.4 two methods are presented that draw clusters as shapes which include the corresponding vertices. Additionally, crossings between edges and borders of shapes are avoided. Especially in the planar case in Section 8.3, such a crossing is only allowed if one endpoint of the edge is within the corresponding cluster and the other one is outside of it.

Another way to show the cluster-structure is to draw vertices that belong to the same cluster closer together than such that are in different clusters. Using force-directed methods, one can achieve this by adding a dummy vertex to each cluster or by regarding clusters as big vertices. In this case the force-function must respect the different size of the vertices.
Bibliography


Indermark, K., Thomas, W., Huch, F., Leucker, M., and Noll, T. (1999). Various texts about the TRUTH system for modelling concurrent systems,


Bibliography


