Clustering with spectral methods

Clusterung mit spektralen Methoden

Diplomarbeit
von
Marco Gaertler[1]

Universität Konstanz
Fachbereich Mathematik und Statistik
Fachbereich Informatik und Informationswissenschaft

März 2002

[1]Email: gaertler@fmi.uni-konstanz.de

Abstract

Grouping and sorting are problems with a great tradition in the history of mankind. Clustering and cluster analysis is a small aspect in the wide spectrum. But these topics have applications in most scientific disciplines. Graph clustering is again a little fragment in the clustering area. Nevertheless it has the potential for new pioneering and innovative methods. One such method is the MCL process presented by VAN DONGEN in [vD00]. We investigated the question, if there is a similar approach which involves the graph structure more directly and has a linear space complexity.

Zusammenfassung

Das Einteilen, Gruppieren und Sortieren von Elementen hat eine sehr lange Tradition in der Menschheitsgeschichte. Dabei sind Clustermethoden und Clusteranalyse nur ein geringer Aspekt des Ganzen. Trotzdem finden diese Techniken Anwendungen in fast allen wissenschaftlichen Disziplinen. Die Spezialisierung auf das Clustern von Graphen ist wiederum nur ein winziges Mosaikstück des gesamten Bereiches. Nichts desto weniger enthält diese Einschränkung viele Möglichkeiten für bahnbrechende und innovative Vorgehensweisen. Eine solche Methode stellte VAN DONGEN in [vD00] vor. Wir haben uns in dieser Arbeit mit der Fragestellung auseinandergesetzt, ob es ein verwandtes Verfahren gibt, welches die zugrundeliegende Graphenstruktur noch gezielter einsetzt und zusätzlich mit linearer Speicherplatzkomplexität auskommt.

Acknowledgments

First of all I would like to thank my supervisors Prof. Dr. Wagner and Dr. Brandes for the possibility to create this thesis and for their continuous support. I also would like to thank my family for their love and guidance. Last but not least I would like to thank my friends. I was inspired and motivated by their relief and support. I really had a wonderful time during my studies at the University of Konstanz. I cannot list all the matters you have done for me nor a list of all your names, but especially I would like to thank: Martin and Kirsten Oberhofer, Roland Martin and Peggy Förster, Jens Lohner and Katja Schröder, Stephan Kolassa, Martin Maier, Steffen Mecke, Simon Huhn, Boris Köpf and Sonja Ziniel.

Contents

Chapter 1

Introduction

Imagine being a biologist and having to classify the flora. Short time after having temporally finished your classification someone shows up and presents a strange looking flower. She says that she tried to use your scheme to identify this flower, but it does nowhere completely fit. So she doubts the correctness of your work. Now you have two possibilities. You give up and start again or you can prove this flower to fit into your theory although some conditions are only partially fulfilled or even violated.

Clustering summarises the general process of grouping entities in a "natural" way. It is contained in many toolboxes of various scientific disciplines. We concentrate our research to the area of graph clustering. Graphs and methods involving graphs have become more and more popular in many topics concerning clustering. In the 1970s, when cluster analysis developed its own scientific discipline, graphs were used as data structure for implementation and occasionally for visual representation. As time passed graph theory became more powerful. Thus cluster procedures were first translated into the graph context and then graph theory was used as a tool to state cluster procedures. Still today there exist lots of opportunities for new pioneering algorithms. One such method is the MCL process presented by VAN DONGEN in [vD00].

We had some experiences with large sparse graphs and inspired by this approach we questioned ourselves:

> "Does there exists a method similar to the MCL process which has a lower space complexity?"

More precisely we were interested in a clustering method with linear space complexity and also having the method directly work with the input graph. Due to some practical knowledge concerning eigenvectors of graphs and layouts we restricted our investigations to spectral topics. Our research results will be presented in this thesis. We have developed a new cluster algorithm which invokes spectral methods, has linear space complexity and directly involves the input graph. Besides the spectral techniques we also use basic graph concepts like minimum spanning trees.

Overview

In the next three chapters we give brief introduction to all three topics considered in our research, namely: graph theory, linear algebra and cluster analysis. The MCL process is covered in chapter five. The next two chapters contains our results. In chapter six the generic version of our approach together with its analysis is presented. Algorithmic and implentational details are considered in chapter seven. Some examples are also show there. One of them concerns a quite theoretically situation and the usability of our approach. The others illustrate "real" graphs. Chapter eight has counter–examples as topic. Several problems are presented there, some of which occurred during our research or simply show that the viewer's intuition can easily be fooled. The final chapter contains closing remarks and some conclusions.

In the appendix brief summaries of numerical calculation and calculation concerning linear algebra in the context of graphs can be found. Also a symbol index is present.

Chapter 2

Introduction to graph theory

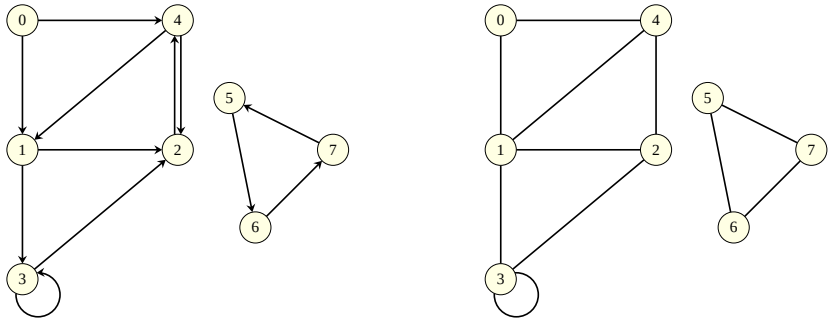
In this chapter we introduce some basic definitions and theorems about graphs and graph theory. The notation given here will be used in the following chapters.

2.1 Definition and notation

A *directed graph* G is a pair (V, E) such that V is a finite nonempty set and E is a binary relation on V. The set V is called the *vertex set* of G, and its elements are called *vertices*. The set E is called the *edge set* of G and its elements are called *edges*. We use *digraph* as a short term for directed graph. In an *undirected graph* the edge set consists of unordered pairs of vertices rather than ordered pairs. Given some digraph $G = (V, E)$ we define the *underlying undirected skeleton* as an undirected graph $[G]$ with the same vertex set V and the same edge set E, ignoring the order of the pairs.

For simplicity we often represent a graph in a pictorial way. Vertices are drawn as (labeled) circles and edges as lines or arrows. Arrows are used to indicate an edge's direction. Figure 2.1 shows a graphical representation of the graph $G_1 := (V_1, E_1)$ defined by $V_1 := \{0, 1, 2, 3, 4, 5, 6, 7\}$ and $E_1 := \{(0, 1), (0, 4), (1, 2), (1, 3), (2, 4), (3, 2), (3, 3), (4, 1), (4, 2), (5, 6), (6, 7), (7, 5)\}$.

Most of the following definitions and notations are similar for directed and undirected graphs. Thus we give only the general statements for directed graphs and mention some additional facts for undirected graphs.



(a) G_1 as digraph

(b) G_1 as undirected graph or $[G_1]$

Figure 2.1: Graphical representations of a graph G_1

Let $G = (V, E)$ be a directed graph. An edge $e \in E$ represents an ordered pair (v_s, v_t) with $v_s, v_t \in V$. So e *connects* v_s and v_t, and v_t is *adjacent to* v_s (by the edge e). The edge e is *incident from* v_s and *incident to* v_t. We call v_s the *source* of e and v_t its *target*. We also use source(e) for v_s and target(e) for v_t. A graph is *bidirected* if for every edge (u, v) the reversed combination (v, u) is also an edge. The edge (v, u) is called the *reversed edge* to (u, v). To every digraph we can associate a *reversed graph*. This graph has the same vertex set and the set of all reversed edges as edge set. The *neighbourhood* $\mathrm{N}(v)$ of a vertex v is the set of all vertices that are connected to v. The elements in $\mathrm{N}(v)$ are called the *neighbours* of v. The set $\mathrm{N}(v)$ can be split into two sets $\mathrm{N}^{\leftarrow}(v)$ and $\mathrm{N}^{\rightarrow}(v)$. The set $\mathrm{N}^{\leftarrow}(v)$ consists of all neighbours that v is adjacent to, and $\mathrm{N}^{\rightarrow}(v)$ consists of all neighbours that are adjacent to v. These sets are not necessarily disjoint. The *degree* of a vertex in a digraph is the number of edges which have v as source plus the number of edges which have v as target.[1] In an undirected graph the *degree* of a vertex is the number of edges which are incident to v. The *in–degree* of v is the cardinality of $\mathrm{N}^{\leftarrow}(v)$, and the *out–degree* is the cardinality

[1] We need this 'strange' formulation since we allow graphs to contain edges for which the source and the target are the same vertex. These edges are called *self–loops* and counted twice when determining the degree.

of $\mathrm{N}^{\rightarrow}(v)$. We use $\deg(v)$ as a short term for the degree of v, $\deg^{\leftarrow}(v)$ for the in–degree of v and $\deg^{\rightarrow}(v)$ for the out–degree of v. We obtain the following two equations:

$$\begin{aligned} \deg(v) &= \deg^{\leftarrow}(v) + \deg^{\rightarrow}(v) && \text{for directed graphs} \\ \deg(v) &= \deg^{\leftarrow}(v) = \deg^{\rightarrow}(v) && \text{for undirected graphs} \end{aligned}$$

Given a graph G we denote by $\delta(G)$ the minimal degree and by $\Delta(G)$ the maximal degree of G.

Consider the digraph G_1 (defined on page 11). The neighbours of 3 are 1, 2 and 3. Its neighbourhood is $\mathrm{N}^{\leftarrow}(3) = \{1, 3\}$ and $\mathrm{N}^{\rightarrow}(3) = \{2, 3\}$. The degree of 3 is four, and both in–degree and out–degree are two.

A *path* p is a sequence $(v_0, v_1, \ldots, v_k)$ of vertices such that (v_{i-1}, v_i) is an edge for $i \in \{1, \ldots, k\}$. This path *connects* v_0 and v_k and has *length* k. We use $v_0 \overset{p}{\rightsquigarrow} v_k$ for the symbolic notation. A sequence $(v_0, v_1, \ldots, v_k)$ of vertices such that (v_{i-1}, v_i) or (v_i, v_{i-1}) is an edge for $i \in \{1, \ldots, k\}$ is called an *undirected path*. A *subpath* of p is a sequence $(v_p, \ldots, v_q)$ with $0 \leq p \leq q \leq k$. The path p is *simple* if all v_i are different. A *circle* is a path which connects a vertex with itself. A circle is called *elementary* if it contains no other circle as a subpath. A *self–loop* is a circle of length one. A graph is called *acyclic* if it contains no circle. If p and p' are two paths which connect v_s and v_t, then p and p' are *edge disjoint* if they have no edge in common. The paths p and p' are *vertex disjoint* if they share no vertex except for v_s and v_t. We define $\eta(v_s, v_t)$ as the maximal number of edge disjoint paths and $\kappa(v_s, v_t)$ as the maximal number of vertex disjoint paths that connect v_s and v_t.

Consider once more the digraph G_1 (defined on page 11). A (directed) path is for example $(0, 4, 1, 3, 2)$, and $(4, 1, 3)$ is a subpath of it. An example for an undirected path is $(0, 1, 4, 2)$, and $(3, 2, 4, 1, 2, 4, 1, 3)$ is a circle which contains $(2, 4, 1, 2)$ as elementary circle. Two vertex disjoint paths which connect 4 and 2 are $(4, 2)$ and $(4, 1, 2)$. Since the out–degree of 4 is two, there can be no more paths which connect 4 and 2 and are vertex disjoint to these two. So $\kappa(4, 2) = 2$.

The inequality $\kappa(v_s, v_t) \leq \eta(v_s, v_t)$ holds for any pair v_s and v_t of vertices, since every set of vertex disjoint paths is also edge disjoint.

A graph is *connected* if every pair of vertices is connected by a path. If the underlying undirected skeleton of a directed graph G is connected, G needs not to be connected. To emphasize that G as directed graph is connected we use sometimes the term *strongly* connected. A maximal subset of the vertex set that is connected is called a *connected component* of the graph. These subsets are well–defined. For a graph G we define the edge connectivity $\eta(G)$ and the vertex connectivity $\kappa(G)$ as:

$$\eta(G) := \min_{\substack{(v_s,v_t)\in V\times V \\ v_s\neq v_t}} \eta(v_s,v_t) \quad \text{and} \quad \kappa(G) := \min_{\substack{(v_s,v_t)\in V\times V \\ v_s\neq v_t}} \kappa(v_s,v_t)$$

The graph G_1 has $\{0\}$, $\{1,2,3,4\}$ and $\{5,6,7\}$ as connected components. The undirected skeleton $[G_1]$ has $\{0,1,2,3,4\}$ and $\{5,6,7\}$ as connected components. The graph G_2 defined in Figure 2.2 shows that κ and η are not the same since $\kappa(G_2) < \eta(G_2)$. Every path which connects 0 and 4 includes 2 as vertex, therefore $\kappa(G_2) < 2$, but $(0,1,2,4)$ and $(0,7,6,5,2,3,4)$ are two edge disjoint paths. Since every pair of vertices lies on a circle, we have $\eta(G_2) = 2$. We have $\kappa(G_2) = 1$, since G_2 is connected.

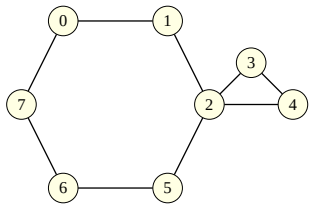
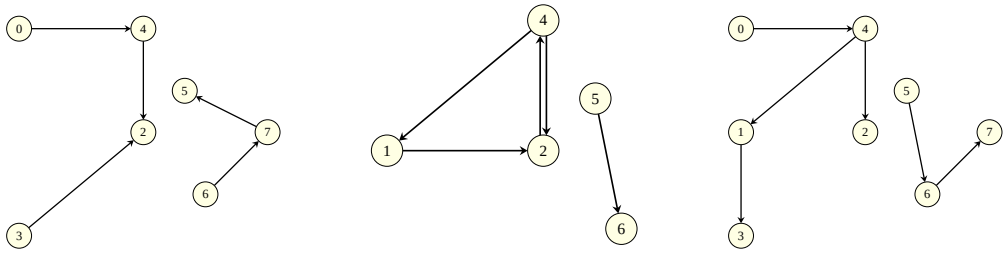


Figure 2.2: G_2

We say that a graph $G' = (V', E')$ is a *subgraph* of $G = (V, E)$ if V' is a nonempty subset of V and E' is a subset of E such that the sources and the targets of all edges in E' are contained in V'. By $E|V'$ we denote the set of all edges of E which have their source and target both in V'. The graph $G'' = (V', E|V')$ is called the *induced subgraph*. A connected component is a nonempty subset of the vertex set. We sometimes call the subgraph induced by a connected component also a connected component. From the context it should be clear if we talk about subsets of vertices or subgraphs. The subgraphs induced by the connected components of the underlying connected graph are called the *undirected connected components* of G. For undirected graphs the undirected connected components coincide with the connected components.

A *tree* is an acyclic graph with a connected underlying undirected skeleton. A graph of which the connected components are trees is called a *forest*. A

subgraph of a graph which has the same vertex set and is a tree or a forest is called *spanning*.



(a) a subgraph of G_1

(b) an induced subgraph of G_1

(c) a spanning forest of G_1

Figure 2.3: Subgraph, induced subgraph and spanning forest

The graph shown in figure 2.3(a) is a subgraph of G_1, but it is not induced by a subset of vertices, since $(5, 6)$ and $(2, 4)$ are not contained in it. However, figure 2.3(b) shows an induced subgraph of G_1 induced by $\{1, 2, 4, 5, 6\}$. Finally, figure 2.3(c) shows a spanning forest of G_1. There are no possible spanning trees for G_1, since the underlying undirected skeleton is not connected.

A graph G is *complete* if its edge set is maximal. A *clique* is a proper subset V' of the vertex set such that the subgraph induced by V' is complete. Two graphs $G = (V, E)$ and $H = (V', E')$ are *isomorphic* if there exists a bijective mapping f from V to V' such that

$$\forall\, u, v \in V : (u, v) \in E \iff (f(u), f(v)) \in E'.$$

2.2 Partitions and Cuts

In many applications it is necessary to split a graph into smaller pieces. Using simply connected components often introduces pieces which are too big. We now present some general definitions and describe certain purposes. Later

we will use the notations given here to specify the settings of the problems and approaches.

Let $G = (V, E)$ be a graph with connected underlying undirected skeleton and S a nonempty proper subset of V. Then S is a *separator* if the subgraph induced by $V \setminus S$ has more than one undirected connected component. We often use $\overline{S} := V \setminus S$ as a short term.

Definition 2.1
A *partition* $\mathbb{P}$ of a finite set X is a collection of sets $(X_1, \ldots, X_d)$, such that:

i.) X_i is a nonempty subset of X,

ii.) the union of all X_i is X and

iii.) any two different sets X_i and X_j are disjoint.

The subsets X_i are the *components* of $\mathbb{P}$. The *size* of $\mathbb{P}$ is d. A *partition* of a graph is a partition of its vertex set.

In the context of some applications the separation effects of partitions are too strong. In view of this effect we introduce the notion of *overlapping partitions*:

Definition 2.2
An *overlapping partition* $\mathbb{P}$ of a finite set X is a collection of sets $(X_1, \ldots, X_d)$, such that:

i.) X_i is a nonempty subset of X,

ii.) the union of all X_i is X and

iii.) for every $i \in \{1, \ldots, d\}$ the set X_i contains at least one element which is not contained in the union of the other components.

We use the same notations for *components* and *size* as for partitions. In addition, an *overlapping partition* of a graph is an overlapping partition of its vertex set.

Let $\mathbb{P} = (V_1, \ldots, V_d)$ be a partition of a graph. By $E(V_i, V_j)$ we denote all edges with one vertex in V_i and the other in V_j. A *cut* is a partition of size two and the edges between the two components are called *cut edges*. Since a cut has two components we can simply write $(S, \overline{S})$ as partition. So the cut depends only on S and the graph. We denote the set of all cut edges by ∂S. A graph is called *bipartite* if there exists a partition $\mathbb{P} = (T, \overline{T})$ such that $E(T, T)$ and $E(\overline{T}, \overline{T})$ are empty or equivalently the graph has a cut such that every edge is a cut edge.

Before we state some splitting problems we define certain types of partitions:

Definition 2.3
Let $G = (V, E)$ be a graph and $k \in \mathbb{N}$. A partition $(B_1, \ldots, B_r)$ is called a *k–block partition* if the subgraph G_j induced by B_j is a maximal subgraph in G with $\kappa(G_j) = k$. An overlapping partition $(C_1, \ldots, C_r)$ is called a *k–component partition* if the subgraph G_j induced by any C_j is a maximal subgraph in G with $\eta(G_j) = k$.

Note that it is necessary to consider overlapping partitions for k–component partitions. For example, for G_2, displayed in Figure 2.2, a 2–component partition is $(\{2, 3, 4\}, \{0, 1, 2, 5, 6, 7\})$. But $(\{0, 1, 2, 5, 6, 7\}, \{3\}, \{4\})$ would be a 2–component partition if we used only partitions. The components of 2–component partitions of undirected graphs are also called *biconnected components*. Vertices which are contained in more than one component are called *cut points* or *articulation points* since removing them with all edges incident to them leads to a graph with more than one connected component. For more information see also [Jun99, p. 338ff.]

Now we state some splitting problems:

Problem 2.4 ((unweighted) Minimum–Cut)
Input: *a graph*

Output: *a cut with minimal number of cut edges with respect to all possible cuts.*

Problem 2.5 (balanced Minimum–Cut)
Input: *a graph*

Output: *a cut with components that differ in size at most by one and with minimal number of cut edges with respect to all such cuts.*

Problem 2.6 (bounded Minimum–Cut)
Input: *a graph with n vertices, two different vertices s and t and a real value α*

Output: *a cut which components have at most $\alpha \cdot n$ elements, s and t belongs to different components and with minimal number of cut edges with respect to all such cuts.*

Problem 2.7 (Separator)
Input: *a graph*

Output: *a separator with minimal cardinality with respect to all possible separators.*

2.3 Weight functions

Here we introduce the notions of *vertex* and *edge weight*, which add information to the pure graph structure. Let $G = (V, E)$ be a graph. A real–valued function is a *vertex weight* if its domain is V and an *edge weight* if its domain is E. We note that it is not necessary to use the real numbers as range, but it is quite common, since most operations, like comparison, are well–defined.

Problem 2.8 ((weighted) Minimum–Cut)
Input: *a graph with nonnegative edge weight*

Output: *a cut such that the sum of the weights of all cut edges is minimal with respect to all possible cuts.*

2.3.1 Example: computer network

Consider a computer network. This network consists of terminals which can communicate with each other. We are interested in two facts: first, how many terminals may break down until the whole network is disjoint, and second, how many connections must be cut to have at least two separated networks.

Model

First we make some assumptions. The number of terminals is fixed. Each terminal has some direct link to another terminal. Communication through a direct link is 'direct' and needs no other terminal. All other communication is somehow routed through the network; there has to be at least one sequence of direct links which connects the terminals. The connection capacity for a direct link is fixed and specified in the input. For modeling this we use an undirected graph. To each terminal in the computer network we associate a vertex and for each direct link we insert an edge. The capacity is represented by an edge weight function. Figure 2.4 shows an instance of such a model.

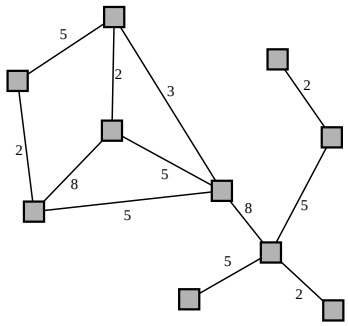


Figure 2.4: Instance of the computer network model

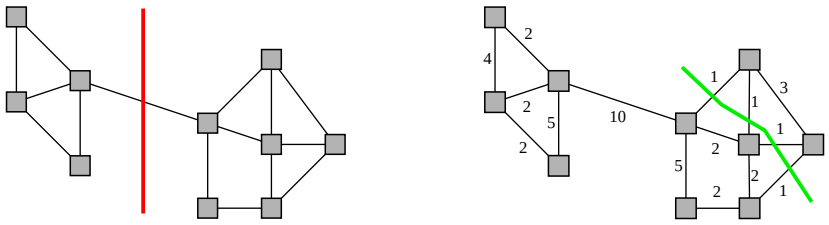
Solutions for the problems

"How many terminals must be out of order so that the whole network is disjoint?" To answer this question we need to look at separators. A separator in our models represents a set of terminals which are out of order. To be precise we have to search for a separator with minimal cardinality with respect to all possible separators (see problem 2.7). This is the solution, since such a separator is the minimal number of vertices (terminals) which must be removed (break down) to disconnect the network.

"How many connections must be cut to have at least two separated networks?" Here we consider two different aspects: The first one concerns the number of cables we have to cut in order to separate the network. The second one considers the amount of connection capacity which has to be removed until the network is separated. What is the difference between these two aspects? Cutting a single cable with high capacity may have greater impacts than cutting some cables with low capacities.

Both problems can be solved with the same technique. We are looking for a cut. A cut in the graph is equivalent to cutting cables or direct links in

the computer network. To answer the first question we have to calculate a minimal cut (see problem 2.4). The weighted analogue will answer the second question (for a description see problem 2.8), since the connection capacity and not the number of cables is counted.



(a) unweighted case (b) weighted case

Figure 2.5: Minimal cuts

Figure 2.5 displays the difference between an unweighted and a weighted cut in a sample instance. The figures show the undirected graph, and the edge weight is written near the line which represents the edge. The bold lines 'cut' the cut edges and stand for the cuts.

2.4 Data structures

Before we move to the next topic we give a short overview on how a graph can be represented by elementary data structure such as lists. Let $G = (V, E)$ be a given graph. For simplicity we assume V equals $\{1, \ldots, n\}$ for some natural number n.

Matrix representation

We can represent G as a real–valued square matrix $\mathcal{A}(G)$ of dimension n. For every edge (i, j) we set the entry (i, j) to one and all other entries to zero. Then $\mathcal{A}(G)$ is called an *adjacency matrix* of G. We require space for n^2 matrix entries. Access time depends only on the matrix structure and

is constant in general. If we need one time unit to access any entry of $\mathcal{A}(G)$ then we have the following access times:

- testing if two vertices are adjacent takes one time unit
- specifying $N^{\leftarrow}$ or $N^{\rightarrow}$ for a vertex takes n time units

  specifying N for a vertex takes $2n-1$ time units
- adding or deleting an edge takes one time unit
- deleting a vertex takes $2n-1$ time units, since we can simply set all $(i,\cdot)$ and $(\cdot,i)$ to zero (i represents the vertex to be deleted and $(i,\cdot)$ is a short term for all possible pairs which have i as first component)
- adding a vertex is more complex, since we cannot add a column and a row to a matrix in general. Therefore we need to copy the whole matrix. This would need n^2 time units and the space for $n^2+(n+1)^2$ elements

If we have exactly one edge weight in addition, we can also represent it using the adjacency matrix. In this case we assume that an edge weight of zero implies that the edge can be deleted from the graph. We set the entry (i,j) of $\mathcal{A}(G)$ to the weight of the edge (i,j) or to zero if there is no edge (i,j). Using this representation $\mathcal{A}(G)$ is called a *weighted adjacency matrix*.

List representation

A matrix representation uses a lot of space, especially if G has only a few edges. If we allow higher access times, we can reduce the used space. Two types are common:

First we present the *incidence list*. Here we associate to each vertex j a list I_j which contains all edges $(j,\cdot)$. We need space for $|E|$ edge elements.[2] We write $|I_j|$ for the size of I_j. The size of I_j equal to the degree of j. We assume that we need one time unit to access the first, last, previous or succeeding element a list. Deleting a given list element or adding a new list element also takes one time unit. Then we have the following access times:

[2] $|T|$ is the cardinality of the set T.

- testing if vertex i is adjacent to vertex j takes at most $|I_i|$ time units (just look at each edge element in I_i and test if its target is j)
- specifying $\mathrm{N}^{\rightarrow}(j)$ for a vertex j takes $|I_j|$ time units,
  specifying $\mathrm{N}^{\leftarrow}(j)$ for a vertex j takes $|E| - |I_j|$ time units and
  specifying $\mathrm{N}(j)$ for a vertex j takes $|E|$ time units
- adding an edge takes one time unit and deleting an edge (i, j) takes at most $|I_i|$ time units
- deleting a vertex j takes $|E|$ time units, since we need to delete the list I_j and check all other lists for the vertex j
- adding a vertex is very simple, since we just add a new list

Next, we describe the *adjacency list*. Here we combine the adjacency matrix with the incidence list. We associate to each vertex j a list A_j, which points to all nodes of $\mathrm{N}^{\rightarrow}(j)$. We need space for $|E|$ vertex elements. Assuming the same condition as for incidence list representations we obtain the same access times. The difference between these two list representations is the used space. An edge element is a pair of vertices and therefore uses twice as much space as a vertex element.

Chapter 3

Linear algebra and spectral methods

We assume that the reader is familiar with basic linear algebra. This includes concepts like matrices, vector spaces, linear mappings, scalar products and some well known propositions. For references see [Fis97], [Fie86] or [Lan72].

We first state some definitions and propositions which are often used in the succeeding chapters. Most of these will concern eigenvalue and eigenvector theory which is the basis for the second part of this chapter. We consider graphs from an algebraic point of view and introduce spectral methods for graphs. Here we skip most proofs since they can be either found in the given references or are easy to prove.

We use a standard notation: let A be a matrix and x a vector. Then we have

- $[A]_{i,j}$ for the ij–th entry of A and $[x]_k$ for the k–th entry of x
- A^T for the transpose of A and also x^T for the transpose of x
- I for the unit matrix and I_n for the unit matrix of order n
- $\mathbf{1}$ for the vector with all entries equal to one

3.1 Eigenvalues and eigenvectors

The theory concerning eigenvalues and eigenvectors is also called *spectral theory*. In this and the following section we consider the spectral theory from a general point of view.

Definition 3.1
Let A be a square $n \times n$ matrix over a field K. A pair $(\lambda, x) \in K \times K^n$ is an *eigenpair* if

$$x \neq 0 \qquad \text{and} \qquad Ax = \lambda x.$$

We call λ an *eigenvalue* and x and *eigenvector* of A. The *eigenspace* E_λ is the subset of all vectors x such that $Ax = \lambda x$. The *characteristic polynomial* p_A is defined by:

$$p_A(T) := \det(A - T \cdot I).$$

The set of all eigenvalues of A is called the *spectrum* of A and denoted by $\Lambda(A)$. For $K = \mathbb{R}$ or $K = \mathbb{C}$ we define the *spectral radius* of A as the maximal absolute value of all eigenvalues of A. The spectral radius is denoted by $\varrho(A)$.

We note some simple statements in eigenvalue theory:

- If A and B are two matrices of the same order and B is invertible, then A and $B^{-1}AB$ have the same eigenvalues and x is an eigenvector of A iff $B^{-1}x$ is an eigenvector of $B^{-1}AB$.
- The eigenspace E_λ is a subspace of K^n, and $E_\lambda = \ker(A - \lambda I)$. Every non–zero vector x in E_λ is an eigenvector of λ.
- The characteristic polynomial is well–defined, i.e. is invariant under basis transformation. Its roots coincide with the eigenvalues of the matrix.
- Eigenvectors of different eigenvalues are linear independent.
- A matrix of order n has at most n different eigenvalues.
- The spectral radius $\varrho(A)$ is the radius of the smallest disc D in the complex plane with centre zero such that all eigenvalues of A are contained in D or on its boundary.

Definition 3.2
Let A be a $n \times n$ matrix over a field K. We call A *diagonalisable* if there exists an invertible $n \times n$ matrix B such that $B^{-1}AB$ is a diagonal matrix. This is equivalent to K^n having a basis consisting of eigenvectors of A.

Not every matrix is diagonalisable. For example consider the field $\mathbb{R}$ and

$$A := \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}.$$

Then $(1,0)^T$ is an eigenvector of A to the eigenvalue 1. But there is no other eigenvector that is linear independent from $(1,0)^T$. So A can not be diagonalisable, since $\mathbb{R}^2$ has no basis containing only eigenvectors of A.

Definition 3.3
A matrix A is called *reducible* if there exists a permutation matrix P such that

$$P^{-1}AP = \begin{pmatrix} A_1 & B \\ 0 & A_2 \end{pmatrix}$$

and A_1 and A_2 are square matrices of order at least one. Otherwise A is called *irreducible*.

In the next two sections we describe two subclasses of $K^{n\times n}$ which have some general spectral properties. Throughout the rest of this chapter, we assume that all matrices are real valued. Since we can embed $\mathbb{R}$ in $\mathbb{C}$, we sometimes speak of complex eigenvalues of a real matrix. We note here that if λ is a complex eigenvalue of a matrix A, the complex conjugate $\overline{\lambda}$ is also an eigenvalue of A.

3.2 Non–negative matrices

Definition 3.4
A matrix A is called *nonnegative* if every entry of A is nonnegative. It is called *positive* if every entry is positive. We use the notation $A \geq 0$ and $A > 0$ to indicate that A is nonnegative respectively positive. We denote the set of all nonnegative $n \times n$ matrices $\mathcal{N}_0(n)$ and the subset of all positive matrices by $\mathcal{N}_+(n)$. Nonnegative and positive vectors are defined in an analogous way.

Proposition 3.5
For every natural number n the set $\mathcal{N}_0(n)$ is closed under transposition, addition and matrix multiplication.

One important theorem about the spectrum of nonnegative matrices is:

Theorem 3.6 (PERRON–FROBENIUS theorem)
Let A be a nonnegative irreducible square matrix of order n. Then the spectral radius $\varrho(A)$ is a simple positive eigenvalue of A, and there is a positive eigenvector belonging to $\varrho(A)$. No nonnegative eigenvector belongs to any other eigenvalue of A.

A proof can be found in [Fie86]. The eigenvalue $\varrho(A)$ is called the PERRON *root* of A and the associated eigenvector the PERRON *vector* of A. All conditions of theorem 3.6 are necessary:

- non–negativity: consider for example $A = -I_n$. A has spectral radius 1 but only -1 as eigenvalues.
- irreducibility: take $A = I_n$, it has 1 as spectral radius, but 1 is not a simple eigenvalue.

It is also possible that other eigenvalues have absolute value equal to $\varrho(A)$. Consider the matrix

$$A := \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

then $(1,1)^T$ is an eigenvector to the eigenvalue 1 which is also the spectral radius of A, and $(-1,1)^T$ is an eigenvector to the eigenvalue -1. Here we have

$$|-1| = |1| = \varrho(A)$$

The next theorem handles these eigenvalues with absolute value $\varrho(A)$:

Theorem 3.7
Let A be an irreducible nonnegative square matrix of order n. Let h be a positive integer. The following properties of A and h are equivalent:

1. *There exist exactly h distinct (complex) eigenvalues of A with absolute value equal to $\varrho(A)$.*

2. *There exists a permutation matrix P such that $P^T AP$ has the form*

$$\begin{pmatrix} A_1 & A_{12} & 0 & \dots & 0 \\ 0 & A_2 & A_{23} & \dots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \dots & A_{h-1,h} \\ A_{h1} & 0 & 0 & \dots & A_h \end{pmatrix},$$

where the diagonal blocks $A_1, \dots, A_h$ are square matrices of order at least one, $A_1 = \dots = A_h = 0$ and there is no permutation matrix which puts A into an analogous form having more than h block rows.

3. *If*

$$p_A(T) = (-1)^n T^n + \sum_{\ell=1}^{s} \alpha_{n_\ell} T^{n_\ell}$$

with $\alpha_{n_\ell} \neq 0$ and $n > n_1 > \dots n_s \geq 0$, then the greatest common divisor of the numbers

$$n - n_1, n_1 - n_2, \dots, n_{s-1} - n_s$$

is h.

4. *The maximal positive integer k such that*

$$\Lambda\left(\exp\left(\frac{2\pi i}{k}\right) \cdot A\right) = \Lambda(A)$$

is h.

We took the formulation from [Fie86]. A proof can also be found there. Before finishing this section on nonnegative matrices, we introduce an important subclass of $\mathcal{N}_0$:

Definition 3.8
Let $N \in \mathcal{N}_0(n)$. If every row sum equals 1 then N is called *stochastic*. The set of all stochastic matrices is denoted by $\mathcal{N}_S(n)$.

The condition that every row sum equals 1 can be written as

$$N \cdot \mathbf{1} = \mathbf{1}, \tag{3.1}$$

and so we have:

Proposition 3.9
Every stochastic matrix has spectral radius 1 *and* $\mathbf{1}$ *is a corresponding eigenvector. The set* $\mathcal{N}_S(n)$ *is closed under matrix multiplication.*

Proof:
Let N be a stochastic matrix. Equation (3.1) implies that $\mathbf{1}$ is an eigenvector to eigenvalue 1 of N. So we have $\varrho(N) \geq 1$. To see that $\varrho(N) \leq 1$ we use the GERSHGORIN theorem (for details see [Fie86, th. 9.15, p. 215]). There we have:

$$\Lambda(A) \subset \bigcup_i \left\{ z \in \mathbb{C} : \left| [A]_{i,i} - z \right| \leq \sum_{j \neq i} \left| [A]_{i,j} \right| \right\}$$

Due to the facts that:

$$\forall\, i,j \colon [A]_{i,j} \in [0,1] \quad \text{and} \quad \forall\, i \colon \sum_{j \neq i} [A]_{i,j} = 1 - [A]_{i,i},$$

we obtain that the real parts of all eigenvalues of A have absolute value less or equal to 1. This proves $\varrho(N) \leq 1$ and so we have $\varrho(N) = 1$. Let N' be a stochastic matrix of the same order as N, then we have:

$$(N' \cdot N) \cdot \mathbf{1} = N'(N\mathbf{1}) = N'\mathbf{1} = \mathbf{1}$$

Since equation (3.1) is fulfilled for $N'N$ and the product of two nonnegative matrices of the same order is a nonnegative matrix (proposition 3.5) we have that $N'N$ is stochastic. □

3.3 Symmetric matrices

Definition 3.10
A matrix A is called *symmetric* if $A^T = A$. The set of all symmetric matrices of order n is denoted by $\mathcal{S}(n)$.

Proposition 3.11
For every natural number n *the set* $\mathcal{S}(n)$ *is closed under transposition and addition.*

Before we can state the main spectral theorem for symmetric matrices we need the notation of a scalar product and some of its properties:

Definition 3.12
A bilinear[1] mapping $\langle , \rangle : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is called a *(real) scalar product* if the following conditions hold:

1. symmetry: $\forall\, x, y \in \mathbb{R}^n\colon\ \langle x, y\rangle = \langle y, x\rangle$
2. positivity: $\forall\, x \in \mathbb{R}^n\colon\ \langle x, x\rangle \geq 0$ and $\langle x, x\rangle = 0$ iff $x = 0$

A mapping $\langle , \rangle_{\mathbb{C}} : \mathbb{C}^n \times \mathbb{C}^n \to \mathbb{C}$ is called a *(complex) scalar product* if the following conditions hold:

1. $\langle , \rangle_{\mathbb{C}}$ is linear in the first argument
2. symmetry: $\forall\, x, y \in \mathbb{C}^n\colon\ \langle x, y\rangle_{\mathbb{C}} = \overline{\langle y, x\rangle_{\mathbb{C}}}$
3. positivity: $\forall\, x \in \mathbb{C}^n\colon\ \langle x, x\rangle_{\mathbb{C}} \in \mathbb{R}_0^+$ and $\langle x, x\rangle_{\mathbb{C}} = 0$ iff $x = 0$

In general we use only real scalar products. We introduced the complex variant since it will simplify a proof later on.

Lemma 3.13
Let $\alpha \in \mathbb{R}^n$ be positive. Then

$$\langle x, y\rangle_\alpha := \sum_{i=1}^{n} [\alpha]_i \cdot ([x]_i [y]_i)$$

with $x, y \in \mathbb{R}^n$ is a scalar product. The vector α is also called the weight of the scalar product.

The scalar product with weight $\mathbf{1}$ is called the *euclidian* scalar product. It can be rewritten as

$$\langle x, y\rangle_{\mathbf{1}} = x^T \cdot y$$

and provides an alternative definition of symmetric matrices:

Lemma 3.14
A matrix $A \in \mathbb{R}^{n\times n}$ is symmetric iff equation (3.2) holds:

$$\forall\, x, y \in \mathbb{R}^n\colon\ \langle Ax, y\rangle_{\mathbf{1}} = \langle x, Ay\rangle_{\mathbf{1}} \tag{3.2}$$

[1] linear in both arguments

Proof:
Equation (3.2) can be written as:

$$\langle Ax, y\rangle_{\mathbf{1}} = (Ax)^T y = x^T A^T y \qquad \text{and} \qquad \langle x, Ay\rangle_{\mathbf{1}} = x^T Ay$$

If A is symmetric we have $A^T = A$ and so equation (3.2) holds. If equation (3.2) holds then we have $x^T A^T y = x^T Ay$ for all pairs of vectors. This implies $A^T = A$. □

We usually use the euclidian scalar product and so we omit the index and use only $\langle , \rangle$. Now we state some auxiliary lemmas which introduce some useful properties. This introduces a geometrical point of view. We define:

Definition 3.15
Let $x \in \mathbb{R}^n$. Then the *length* of x is defined by:

$$\|x\| := \sqrt{\langle x, x\rangle}.$$

Two vectors $x, y \in \mathbb{R}^n$ are *orthogonal* if $\langle x, y\rangle = 0$. A basis of $\mathbb{R}^n$ is called *orthogonal* if each pair of different vectors is orthogonal. It is called *orthonormal* if it is orthogonal and every vector has length 1.

Lemma 3.16
Vectors which are not zero and pairwise orthogonal are linear independent.

Proof:
Otherwise let $x_1, \ldots, x_k \in \mathbb{R}^n$ be a counter–example and

$$0 = \sum_{i=1}^{k} a_i x_i$$

with $a_i \in \mathbb{R}$ and not all a_i zero. Fix an arbitrary index j such that $a_j \neq 0$ then we have:

$$0 = \langle 0, x_j\rangle = \left\langle \sum_{i=1}^{k} a_i x_i, x_j \right\rangle = \sum_{i=1}^{k} a_i \langle x_i, x_j\rangle = a_j \langle x_j, x_j\rangle$$

Because x_j is not zero we have $\langle x_j, x_j\rangle \neq 0$, so $a_j = 0$. Since j was arbitrary we have a contradiction. □

Lemma 3.17 (Gram–Schmidt orthogonalisation process)
Let $x_1, \ldots, x_k \in \mathbb{R}^n$ be nonzero and pairwise orthogonal and $x \in \mathbb{R}^n$ a nonzero vector with

$$x \notin \operatorname{span}(x_1, \ldots, x_k).$$

Then there exists a vector $x_{k+1} \in \mathbb{R}^n$ such that:

1. *$x_1, \ldots, x_{k+1}$ are linear independent and pairwise orthogonal*
2. *$x_{k+1} \in \operatorname{span}(x_1, \ldots, x_k, x)$*

Proof:
We set x_{k+1} to:

$$x_{k+1} := x - \sum_{i=1}^{k} \frac{\langle x, x_i \rangle}{\langle x_i, x_i \rangle} x_i. \tag{3.3}$$

Since x_{k+1} is a linear combination of $x_1, \ldots, x_k$ and x condition 2 is fulfilled. The vector x_{k+1} is not zero since $x \notin \operatorname{span}(x_1, \ldots, x_k)$. So by lemma 3.16 it is sufficient to show that $x_1, \ldots, x_{k+1}$ are pairwise orthogonal and therefore we show that x_{k+1} is orthogonal to x_j for $j = 1, \ldots, k$:

$$\begin{aligned} \langle x_{k+1}, x_j \rangle &= \left\langle x - \sum_{i=1}^{k} \frac{\langle x, x_i \rangle}{\langle x_i, x_i \rangle} x_i, x_j \right\rangle = \langle x, x_j \rangle - \sum_{i=1}^{k} \frac{\langle x, x_i \rangle}{\langle x_i, x_i \rangle} \langle x_i, x_j \rangle \\ &= \langle x, x_j \rangle - \frac{\langle x, x_j \rangle}{\langle x_j, x_j \rangle} \langle x_j, x_j \rangle = 0 \end{aligned}$$

This completes the proof. □

Lemma 3.18
The space $\mathbb{R}^n$ has an orthonormal basis.

Proof:
Choose any basis of $\mathbb{R}^n$ and construct an orthogonal basis by successive application of lemma 3.17. By scaling each basis vector we get an orthonormal basis. □

In general it is not possible to have a basis of eigenvectors, because not all matrices are diagonalisable. Next we show that symmetric matrices are diagonalisable and that there exists an orthonormal basis of eigenvectors. The first step is:

Proposition 3.19
A real and symmetric matrix has at least one eigenvalue and all eigenvalues are real.

Proof:
Let A be a real and symmetric matrix. The matrix A can be viewed as complex matrix with real valued entries and therefore we have $A = \overline{A}$. Its characteristic polynomial p_A is not constant and thus has a complex root. Next we show that all eigenvalues of A are real. Therefor we consider the complex variant of the euclidian scalar product which is defined by:

$$\langle x, y \rangle_{\mathbf{1},\mathbb{C}} := x^T \overline{y}$$

So let (λ, x) be a (complex) eigenpair of A with $\lambda \in \mathbb{C}$. Then we have:

$$\begin{aligned} \lambda \langle x, x \rangle_{\mathbf{1},\mathbb{C}} &= \langle Ax, x \rangle_{\mathbf{1},\mathbb{C}} \\ &= (Ax)^T \overline{x} \;=\; x^T A^T \overline{x} \;=\; x^T A \overline{x} \\ &= x^T \overline{A} \overline{x} \;=\; v^T \overline{(Ax)} \\ &= \langle x, Ax \rangle_{\mathbf{1},\mathbb{C}} \;=\; \langle x, \lambda x \rangle_{\mathbf{1},\mathbb{C}} \\ &= \overline{\lambda} \langle x, x \rangle_{\mathbf{1},\mathbb{C}} \end{aligned}$$

Since $x \neq 0$ we have $\langle x, x \rangle_{\mathbf{1},\mathbb{C}} > 0$ and therefore $\lambda = \overline{\lambda}$. Thus λ is real. □

Lemma 3.20
Let $A \in \mathcal{S}(n)$ and (λ, x) an eigenpair of A. Then we have for $y \in \mathbb{R}^n$:

$$\langle y, x \rangle = 0 \qquad \Longrightarrow \qquad \langle Ay, x \rangle = 0$$

Proof:
We use lemma 3.14 and gain:

$$\begin{aligned} \langle Ay, x \rangle &= \langle y, Ax \rangle \;=\; \langle y, \lambda x \rangle \\ &= \lambda \langle y, x \rangle \end{aligned}$$

This proves the implication. □

Theorem 3.21
Let A be a real and symmetric matrix of order n. Then there exists an orthogonal basis of $\mathbb{R}^n$ consisting of eigenvectors of A.

Proof:
We prove this by induction on n.

$n = 1$: Choose any vector of length 1. This is sufficient as basis.

$n > 1$: By proposition 3.19 we know A has real eigenvalues, so let x be an eigenvector of A with $\|x\| = 1$. Since $x \neq 0$ there exist vectors $y_1, \ldots, y_{n-1}$ such that $x, y_1, \ldots, y_{n-1}$ is a orthonormal basis of $\mathbb{R}^n$. Because of lemma 3.20 we have

$$\forall\, 1 \leq i \leq n-1 \colon Ay_i \in \operatorname{span}(y_1, \ldots, y_{n-1})\,.$$

We now use this new basis and gain:

$$\begin{pmatrix} \lambda & 0 \ldots 0 \\ 0 & \\ \vdots & A' \\ 0 & \end{pmatrix} \simeq A$$

Because A is symmetric A' is also symmetric. By the induction hypothesis we get an orthonormal basis $x_1, \ldots, x_{n-1}$ consisting of eigenvectors for the subspace $\operatorname{span}(y_1, \ldots, y_{n-1})$. Then $x, x_1, \ldots, x_{n-1}$ is an orthonormal basis of $\mathbb{R}^n$ consisting of eigenvectors of A. □

This completes the first part of this chapter. Next we consider graphs from an algebraic viewpoint.

3.4 Graphs and matrices

As we have seen in section 2.4 there is a certain connection between graphs and matrices. In this section we pick up this topic again, this time from the algebraic point of view, and show that some algebraic properties have analogies in graph theory. For better readability we consider only unweighted graphs. All definitions and theorems can usually be carried out for weighted graphs. In appendix A we state some of the essential definitions.

Definition 3.22
Consider a real matrix A of order n. We define a directed graph $\mathsf{G}(A)$ associated to A by:

- the set $\{1, \ldots, n\}$ as vertex set and
- there is an edge connecting i with j if $[A]_{i,j} \neq 0$.

Definition 3.22 is somehow an inverse construction to the adjacency matrix, in fact we have the following lemma:

Lemma 3.23
Let H be a graph and $\mathcal{A}(H)$ an adjacency matrix. Then H is isomorphic to $\mathsf{G}(\mathcal{A}(H))$. Let B be a real matrix of order n with entries in $\{0, 1\}$ and B' an adjacency matrix of $\mathsf{G}(B)$. Then there exists a permutation matrix P such that

$$B' = PBP^{-1}.$$

The proof of lemma 3.23 is quite simple, since we just have to fix a vertex numbering. Also the concept of incidence can be expressed by matrices:

Definition 3.24
Let $G = (V, E)$ be a graph. An *incidence matrix* of G is a $|V| \times |E|$ matrix $\mathcal{I}(G)$ with

$$[\mathcal{I}(G)]_{v,e} = \begin{cases} -1 & \text{if source}(e) = v \text{ and } e \text{ is not a self–loop} \\ 1 & \text{if target}(e) = v \text{ and } e \text{ is not a self–loop} \\ 0 & \text{otherwise} \end{cases}$$

for $v \in V$ and $e \in E$.

We now state some simple relations between the adjacency matrix and the corresponding graph:

Lemma 3.25
Let $G = (V, E)$ be a graph and $\mathcal{A}$ an adjacency matrix. Then $\mathcal{A}^T$ is an adjacency matrix of the reversed graph of G. Let $v, w \in V$ and $k \in \mathbb{N}$ then $[\mathcal{A}^k]_{v,w} \neq 0$ iff there exists a path between v and w in G with length k.

Proof:
Let $H = (V, F)$ be the reversed graph to G. For two vertices v, w we have:

$$(v, w) \in E \qquad \Longleftrightarrow \qquad (w, v) \in F \tag{3.4}$$

Using the definition of the adjacency matrix we get

$$(v, w) \in E \qquad \Longleftrightarrow \qquad [\mathcal{A}]_{v,w} \neq 0 \tag{3.5}$$

Combining equations (3.4) and (3.5), we obtain:

$$\begin{aligned} [\mathcal{A}]_{v,w} \neq 0 &\iff (v, w) \in E \\ &\iff (w, v) \in F \\ &\iff [\mathcal{A}]_{w,v} \neq 0 \\ &\iff [\mathcal{A}^T]_{v,w} \neq 0 \end{aligned}$$

This completes the first part. Since the second part can be proved by induction we omit it. For example it can be found in [Jun99]. □

Lemma 3.26
Let $G = (V, E)$ be a graph and $\mathcal{A}$ an adjacency matrix. Then $\mathcal{A}$ is symmetric iff G is bidirected respectively undirected. The matrix $\mathcal{A}$ is irreducible iff G is connected.

Proof:
Assume G is directed. Let $v, w \in V$. If $\mathcal{A}$ is symmetric we have

$$(v, w) \in E \Longleftrightarrow [\mathcal{A}]_{v,w} \neq 0 \Longleftrightarrow [\mathcal{A}]_{w,v} \neq 0 \Longleftrightarrow (w, v) \in E.$$

If G is bidirected we have

$$[\mathcal{A}]_{v,w} \neq 0 \Longleftrightarrow (v, w) \in E \Longleftrightarrow (w, v) \in E \Longleftrightarrow [\mathcal{A}]_{w,v} \neq 0.$$

If G is undirected we can interpret G as bidirected digraph and use the upper proof. For the second part assume first that $\mathcal{A}$ is reducible but G is connected. Then we can split V in two non–empty subsets V' and V'' such that

$$\mathcal{A} = \begin{pmatrix} \mathcal{A}(G') & * \\ 0 & \mathcal{A}(G'') \end{pmatrix}, \tag{3.6}$$

where G' and G'' are the induced subgraphs of V' and V''. Since both of them are non–empty we pick two vertices $v \in V'$ and $w \in V''$. Since G is

connected there exists a path from w to v. By induction we gain the existence of an edge e with $\text{source}(e) \in V''$ and $\text{target}(e) \in V'$. So we have

$$[\mathcal{A}]_{\text{source}(e),\text{target}(e)} \neq 0$$

which is a contradiction to equation (3.6). Second assume $\mathcal{A}$ is irreducible and G is not connected. Then there are two vertices v and w such no path connects w with v. By V'' we denote the set of all vertices in V which are reachable from w

$$V'' := \left\{ u \in V \,:\, \exists\, p \text{ path}\colon w \overset{p}{\rightsquigarrow} u \right\}.$$

The set V'' is not empty since $w \in V''$. Let G'' be the subgraph induced by V''. Since $v \notin V''$ we have $\overline{V''} \neq \emptyset$. So let G' be the subgraph induced by $\overline{V''}$. There exists no path connecting a vertex in V'' with a vertex in $\overline{V''}$. Otherwise assume (u, u') is a counter–example with $u \in V''$, $u' \in \overline{V''}$ and $u \overset{p}{\rightsquigarrow} u'$. Since $u \in V''$ we have a path p_u connecting w with u. Then the concatenation of p_u and p is a path. This connects w with u', so $u' \in V''$. But this is a contradiction to $u' \in \overline{V''}$. Now we consider a vertex ordering such that first all vertices of $\overline{V''}$ are listed and than the rest. The adjacency matrix G with respect to this ordering has the following form:

$$\mathcal{A}(G) = \begin{pmatrix} \mathcal{A}(G') & * \\ 0 & \mathcal{A}(G'') \end{pmatrix}$$

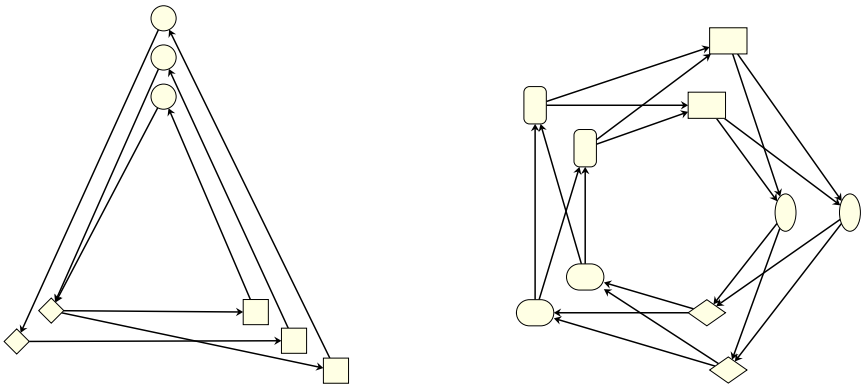
By lemma 3.23 we have $\mathcal{A}(G) = P\mathcal{A}P^{-1}$ for some permutation matrix P. The matrix $\mathcal{A}(G)$ is reducible and so we obtain a contradiction to $\mathcal{A}$'s irreducibility. □

Together with lemma 3.26 and theorem 3.7 we can characterise a certain graph family, therefore we define:

Definition 3.27
A connected digraph $G = (V, E)$ is called *k–cyclic* if there exists a partition $V_1, \ldots, V_k$ of G such that:

$$\forall\, e \in E\colon \text{source}(e) \in V_i \quad \text{and} \quad \text{target}(e) \in V_{i+1 \bmod k} \tag{3.7}$$

Two cyclic graphs are shown in figure 3.1. The vertices of different partition components have different shapes.



(a) a 3–cyclic graph (b) a 5–cyclic graph

Figure 3.1: Cyclic graphs

Theorem 3.28 (Characterisation of k–cyclic graphs)
Let G be a digraph with more than one vertex. The following statements are equivalent:

1. *G is k–cyclic*
2. *the length of all cycles with positive length in G have k as greatest common divisor and G is connected*
3. *every adjacency matrix $\mathcal{A}$ of G has exactly k eigenvalues with absolute value $\varrho(\mathcal{A})$*

Proof:

(1) $\Longleftrightarrow$ (3): Consider the adjacency matrix with respect to the k–cyclic partition. Then we have:

$$\mathcal{A} = \begin{pmatrix} 0 & A_{12} & 0 & \dots & 0 \\ 0 & 0 & A_{23} & \dots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \dots & A_{k-1,k} \\ A_{k1} & 0 & 0 & \dots & 0 \end{pmatrix}, \tag{3.8}$$

where the diagonal blocks are square matrices. There exists also no permutation matrix P such that $P^T \mathcal{A} P$ has a similar form with more block rows since the subgraphs induced by the cyclic partition have empty edge sets. Since $\mathcal{A}$ is nonnegative and irreducible (see lemma 3.26) theorem 3.7 implies the assertion.

(1) $\Longrightarrow$ (2): By contradiction we see that every cycle contains at least one vertex in V_1. Without loss of generality we assume every cycle starts from a vertex in V_1. Because of equation (3.8) we see that we need exactly k edges to reach again a vertex in V_1. So the length of every cycle is a multiple of k. This proves the implication.

(2) $\Longrightarrow$ (1): Let v_0 be an arbitrary vertex of G, then we define a partition $V_1, \ldots, V_k$ by:

> v belongs to V_{j+1} if there exists a path of length j mod k connecting v_0 and v.

This partition is well–defined. Otherwise let $v \in V$ be a counter–example. Thus two paths p_1 and p_2 exist with $v_0 \rightsquigarrow v$ and length j_1 and j_2 such that

$$j_1 \not\equiv j_2 \mod k. \tag{3.9}$$

Because G is connected there exists a path p' of length j' connecting v and v_0. Then (p_1, p') respectively (p_2, p') forms a cycle in G and so we have:

$$j_1 + j' \equiv j_2 + j' \equiv 0 \mod k$$

This is a contradiction to equation (3.9).

None of the V_j is empty. Since G is connected there exists at least one non–trivial cycle. This cycle contains at least k elements. This proves that $V_1, \ldots, V_k$ is a well–defined partition of G.

The partition $V_1, \ldots, V_k$ is a k–cyclic partition. Otherwise let the edge $e := (v, w)$ be a counter–example. Without loss of generality we assume $v \in V_1$ and $w \in V_j$ with $j \neq 2$. Choose

any vertex u of V_2. This is possible since $V_2 \neq \emptyset$. Since G is connected there exists a path p_1 connecting v with u, a path p_2 connecting u with w and a path p_3 which connects w with v. Let j_i denote the length of p_i for $i = 1, 2, 3$. We obtain two cycles: one cycle (p_3, e) of length j_3+1 and another cycle (p_1, p_2, p_3) of length $j_1 + j_2 + j_3$. This implies:

$$\begin{aligned} j_3 &\equiv -1 \mod k \\ j_1 + j_2 + (-1) &\equiv 0 \mod k \end{aligned}$$

Since $v \in V_1$, $w \in V_j$ and $u \in V_2$ we have

$$j_1 \equiv 1 \mod k \qquad \text{and} \qquad j_2 \equiv j - 2 \mod k.$$

Combining these equations we gain:

$$j - 2 \equiv 0 \mod k,$$

and since $0 < j < k$ and $j \neq 2$ we have a contradiction. □

This completes our treatment of the algebraic point of view of the adjacency matrices and we continue with some other interesting matrices associated to graphs.

3.5 The Laplacian matrix and its properties

Throughout the rest of this chapter, we assume that graphs have a fixed vertex and edge ordering and every matrix respects these orderings.

Definition 3.29
Let $G = (V, E)$ be a digraph. The *Laplacian matrix* of G is defined by:

$$\mathcal{L}(G) := \mathcal{I}(G) \cdot \mathcal{I}(G)^T.$$

The *degree matrix* $\mathcal{D}(G)$ of G is a diagonal matrix of order $|V|$ with

$$[\mathcal{D}(G)]_{v,v} = \deg(v)$$

for every $v \in V$.

In definition 3.29 we explicitly used a digraph. To handle undirected graphs we transform them into a digraph by assigning a random direction to every edge. This is well–defined, consider the following lemma:

Lemma 3.30
Let G be a digraph. Let $\mathcal{L}(G)$ the Laplacian matrix of G, $\mathcal{A}(G)$ the adjacency matrix of G and $\mathcal{D}(G)$ the degree matrix of G. Then following equation holds:

$$\mathcal{L}(G) = \mathcal{D}(G) - \mathcal{A}(G) - \mathcal{A}(G)^T \tag{3.10}$$

Let H be an undirected graph, $\mathcal{L}(H)$ a Laplacian matrix, $\mathcal{A}(H)$ the adjacency matrix and $\mathcal{D}(H)$ the degree matrix of H. Then the following equation holds:

$$\mathcal{L}(H) = \mathcal{D}(H) - \mathcal{A}(H) \tag{3.11}$$

Proof:
We prove equation (3.10) entry by entry. Let E' be the set of all edges without self–loops and

$$\deg'(v) := \begin{cases} \deg(v) & \text{if } v \text{ has no self–loop} \\ \deg(v) - 2 & \text{otherwise} \end{cases}$$

Note that $\deg'$ is the degree with respect to E'.

- For the diagonal entry corresponding to $v \in V$ we obtain:

$$\begin{aligned} [\mathcal{L}(G)]_{v,v} &= [\mathcal{I}(G) \cdot \mathcal{I}(G)^T]_{v,v} \\ &= \sum_{\substack{e \in E' \\ \text{source}(e)=v}} 1 \cdot 1 + \sum_{\substack{e \in E' \\ \text{target}(e)=v}} (-1) \cdot (-1) + \sum_{\substack{e \in E \\ e=(v,v)}} 0 \cdot 0 \\ &= \deg'(v) \end{aligned}$$

  Since $[\mathcal{A}(G)]_{v,v} \neq 0$ iff v has a self–loop we gain

$$[\mathcal{D}(G) - \mathcal{A}(G) - \mathcal{A}(G)^T]_{v,v} = \begin{cases} \deg(v) & \text{if } v \text{ has no self–loops} \\ \deg(v) - 2 & \text{otherwise} \end{cases}$$

- For $v, w \in V$ with $v \neq w$ we obtain:

$$\begin{aligned}
[\mathcal{L}(G)]_{v,w} &= [\mathcal{I}(G) \cdot \mathcal{I}(G)^T]_{v,w} \\
&= \sum_{e \in E} [\mathcal{I}(G)]_{v,e} [\mathcal{I}(G)]_{w,e} \\
&= \sum_{\substack{e \in E \\ \{\text{source}(e), \text{target}(e)\} = \{v,w\}}} [\mathcal{I}(G)]_{v,e} [\mathcal{I}(G)]_{w,e} \\
&= \begin{cases} -1 & \text{if only } (v,w) \in E \\ -1 & \text{if only } (w,v) \in E \\ -2 & \text{if } (v,w), (w,v) \in E \\ 0 & \text{otherwise} \end{cases}
\end{aligned}$$

  This matches with $[-\mathcal{A}(G) - \mathcal{A}(G)^T]_{v,w}$, since we have -2 if both edges (v,w) and (w,v) exist, -1 if only one of them is present in E and 0 otherwise. Since $\mathcal{D}(G)$ is a diagonal matrix we gain the equality.

By the same technique we see that equation (3.11) holds for undirected graphs. In particular, we see that the Laplacian matrix of an undirected graph is independent of the chosen orientation. □

Since some graph properties can be recognised in the structure of the associated Laplacian matrix, we list some important properties:

Proposition 3.31
Let $G = (V, E)$ be a digraph, $n := |V|$ and $\mathcal{L} = \mathcal{L}(G)$ the Laplacian matrix to G. Then we have

1. *$\mathcal{L}$ is symmetric and positive semidefinite:*

$$\forall\, v \in \mathbb{R}^n \colon v^T \mathcal{L} v \geq 0$$

   being more precise we have:

$$x^T \mathcal{L} x = \sum_{(u,v) \in E} ([x]_u - [x]_v)^2 \tag{3.12}$$

2. $\operatorname{rank} \mathcal{L} = n - c$, *where c is the number of connected components of $[G]$*

3. *Let Q be a $(n-1)$–minor of $\mathcal{L}$. Then the absolute value of Q equals the number of spanning trees in $[G]$.*

Proof:
We prove only the first two properties. For a proof of the third see [Jun99, p. 104ff.].

- Property 1:

  By using lemma 3.30 we see that $\mathcal{L}$ is symmetric. To see equation (3.12) we use the definition of $\mathcal{L}$. Therefore let $\mathcal{I}$ denote the incidency matrix of G and x an arbitrary vector in $\mathbb{R}^n$:

$$\begin{aligned} x^T \mathcal{L} x &= x^T (\mathcal{I}\mathcal{I}^T) x \\ &= x^T \mathcal{I}\mathcal{I}^T x \\ &= (\mathcal{I}^T x)^T (\mathcal{I}^T x) \\ &= \sum_{(u,v)\in E} ([x]_u - [x]_v)^2 \end{aligned} \tag{3.13}$$

  Since in equation (3.13) we sum up only nonnegative values we see that $\mathcal{L}$ is positive semidefinite.

- Property 2:

  We prove this by induction on c:

  $c = 1$: Because of equation (3.12) we know that $\mathbf{1}^T \mathcal{L} \mathbf{1} = 0$. Therefore we obtain that $\mathbf{1}$ is contained in the kernel of $\mathcal{L}$. This implies $\operatorname{rank} \mathcal{L} \leq n-1$. We know by property 3 that every $(n-1)$ minor of $\mathcal{L}$ is nonzero. This implies $\operatorname{rank} \mathcal{L} \geq n-1$. A proof for this conclusion can be found in [Fis97, p. 197ff.].

  $c > 1$: Let $G_1, \ldots, G_c$ be the undirected connected components of $[G]$. If we use a vertex ordering according to these connected components we gain:

$$\mathcal{L} = \begin{pmatrix} \mathcal{L}(G_1) & & 0 \\ & \ddots & \\ 0 & & \mathcal{L}(G_c) \end{pmatrix}$$

This implies

$$\operatorname{rank} \mathcal{L} = \sum_{i=1}^{c} \operatorname{rank} \mathcal{L}(G_i) = \sum_{i=1}^{c} (|V_i| - 1) = |V| - c.$$

This completes the proof. □

Alternative proofs can also be found in [GR01].

Definition 3.32
Let G be a graph with n vertices. The *Laplacian spectrum* of G is defined as:

$$\Lambda_{\mathcal{L}}(G) := \Lambda(\mathcal{L}(G))$$

and is the set of all eigenvalues of the Laplacian matrix of G. Since this spectrum is real and $\mathcal{L}(G)$ is positive semi–definite we use also

$$0 = \lambda_1(G) \leq \lambda_2(G) \leq \cdots \leq \lambda_n(G)$$

for the eigenvalues of the Laplacian matrix.

We note that the Laplacian spectrum of G is the union of the Laplacian spectra of its undirected connected components.

Corollary 3.33
Let G be a graph with connected undirected underlying skeleton. Then we have

$$\lambda_1(G) = 0 \qquad \textit{and} \qquad \lambda_2(G) > 0.$$

Proof:
By property 2 of proposition 3.31 we have $\dim(\ker \mathcal{L}(G)) = 1$, so $\lambda_1(G) = 0$. Since $\ker \mathcal{L}(G) = E_0$ and $\lambda_2(G) \geq \lambda_1(G)$ we obtain $\lambda_2(G) > 0$. □

Next we cite some interesting and also important relations between graphs and their Laplacian spectrum. We will omit most of the proofs and refer instead only to some literature.

Proposition 3.34
Let $G = (V, E)$ be a graph on n vertices. The function $\pi \colon \mathbb{R}^n \to \mathbb{R}$ defined as:

$$\pi(x) := \frac{\sum_{(u,w)\in E} ([x]_u - [x]_w)^2}{\sum_{w\in V} ([x]_w)^2}$$

has in $\mathbf{1}$ *a global minimum with* $\pi(\mathbf{1}) = 0$*. Restricting* π*'s domain to*

$$D := \{x \in \mathbb{R}^n \setminus \{0\} \ : \ \langle x, \mathbf{1}\rangle = 0\}$$

we obtain that π *has in* x_2 *a global minimum and in* x_n *a global maximum where* x_2 *and* x_n *are eigenvectors of the Laplacian matrix of* G *to the eigenvalues* $\lambda_2(G)$ *respectively* $\lambda_n(G)$*.*

A proof can be found in [GR01, p. 288]

Theorem 3.35
Let $G = (V, E)$ *be an undirected graph with* n *vertices. The following inequations hold:*

- *from [Fie73, section 4]:*

$$2\eta(G)\left(1 - \cos\left(\frac{\pi}{n}\right)\right) \leq \ \lambda_2(G) \ \leq \kappa(G) \leq \eta(G) \leq \delta(G) \tag{3.14}$$

- *from [Moh91, theorem 2.2] or [Moh97, propos. 2.8]:*

$$\lambda_2(G) \leq \frac{n}{n-1}\delta(G) \leq \frac{n}{n-1}\Delta(G) \leq \lambda_n(G) \leq 2\Delta(G)$$

- *from [Moh97, lemma 3.1]: let* $(S, \overline{S})$ *be a cut in* G *then we have:*

$$\lambda_2(G)\frac{|S|(n-|S|)}{n} \leq |\partial S| \leq \lambda_n(G)\frac{|S|(n-|S|)}{n}$$

  If G *is not complete on* n *vertices*

$$\min\left\{\frac{|\partial S|}{|S|} \ : \ S \subset V, 0 < |S| \leq \frac{n}{2}\right\} \leq \sqrt{\lambda_2(G)\Big(2\Delta(G) - \lambda_2(G)\Big)}$$

- *from [GR01, lemma 13.7.1]: let* S *be a nonempty proper subset of* V *then we have:*

$$\lambda_2(G) \leq \frac{|V| \cdot |\partial S|}{|S| \cdot |\overline{S}|}$$

- *from [GR01, corollary 13.7.2]:*

$$\frac{\lambda_2(G)}{2} \leq \min\left\{\frac{|\partial S|}{|S|} \ : \ S \subset V, 0 < |S| \leq \frac{n}{2}\right\} \tag{3.15}$$

- *from [Moh91, eq. 6.10, p. 888]: let D be the diameter of the graph (the diameter is the maximal number of edges which are needed to connect two vertices)*

$$D \geq \frac{4}{|V| \cdot \lambda_2} \tag{3.16}$$

We finish this section by giving a reason why the Laplacian spectrum is so well studied. Consider the following problem:

Problem 3.36 ((general) *d*–dimensional vertex–placement)
Input: *a graph $G = (V, E)$ on n vertices — without loss of generality we assume $V = \{1, \ldots, n\}$ — and a cost function cost: $\mathbb{R}^{n \times d} \to \mathbb{R}$*

Output: *a position function $\varrho\colon V \to \mathbb{R}^d$ such that if cost has a global minimum then the matrix A is such a minimum with*

$$A := \begin{pmatrix} \varrho(1) \\ \ldots \\ \varrho(n) \end{pmatrix}.$$

Reviewing equation (3.12) we can interpret the Laplacian matrix respectively its quadratic form as a cost function. Consider the 1–dimensional vertex–placement problem and as cost function the total edge length, i.e. the total distance of two adjacent vertices. Of course, placing all vertices at the same position is a solution, but in general not desired. Since $\varrho\colon V \to \mathbb{R}$ we can interpret ϱ as vector indexed with the vertices. So we usually postulate that our placement assigning vector has unit length and is orthogonal to $\mathbf{1}$. With a little linear algebra or proposition 3.34 we obtain that all eigenvectors to eigenvalue $\lambda_2(G)$ are a global minimum.

For the moment we denote one of the eigenvectors of $\lambda_2(G)$ with unit length as $x_2(G)$. It is sufficient to know that $x_2(G)$ is an eigenvector to $\lambda_2(G)$ to complete the proof of proposition 3.34. Fiedler was one of the first researchers to investigate the structure of $x_2(G)$. In [Fie75] he showed that there is a relation between $x_2(G)$ and a balanced cut respectively a "centre" of the graph G. For example one theorem which is still interesting and important nowadays is:

Theorem 3.37 ([Fie75, p. 630ff.])
Let G be an undirected connected graph and $x_2(G)$ an eigenvector[2] to the Laplacian eigenvalue $\lambda_2(G)$. Then exactly one of the following two cases occurs:

1. *There is a single block[3] B_0 in G which contains both positively and negatively valuated vertices. Each other block has either vertices with positive valuation only, or vertices with negative valuation only, or vertices with zero valuation only. [...]*

2. *No block of G contains both positively and negatively valuated vertices. There exists a single vertex z which has value zero and has a neighbour with a non–zero valuation. This vertex is a point of articulation. Each block contains (with the exception of z) either vertices with positive valuation only, or vertices with negative valuation only or vertices with zero valuation only. [...]*

Theorem 3.37 and some other propositions often build the background for spectral methods or heuristics. In section 4.4 we consider some approaches using the same or similar methods for cluster analysis. In section 8.1 we have a closer look at a procedure directly derived from theorem 3.37.

3.6 Normalised matrices

Determining relations between the spectrum of a graph and its properties is not trivial. Indeed Chung said in [Chu94]:

> "Roughly speaking, half of the main problems of spectral theory lie in deriving bounds on the distributions of eigenvalues. The other half concern the impact and consequences of the eigenvalue bounds as well as their applications [...]"

Therefore we consider normalised matrices in this section. One is derived directly form the Laplacian the other from the adjacency matrix.

[2] FIEDLER used the term *characteristic valuation* instead
[3] a component of a 2–component partition, see definition 2.3

3.6.1 The Normalised Laplacian matrix

Definition 3.38
Let G be an graph without isolated vertices. Then the *normalised Laplacian matrix* $\mathcal{L}_N(G)$ is defined by:

$$\mathcal{L}_N(G) := \mathcal{D}(G)^{-\frac{1}{2}} \mathcal{L}(G) \mathcal{D}(G)^{-\frac{1}{2}},$$

where $\mathcal{D}(G)^{-\frac{1}{2}}$ is a diagonal matrix with entries $\frac{1}{\sqrt{\deg(v)}}$ for $v \in V$.

The normalised Laplacian matrix $\mathcal{L}_N(G)$ of a graph G is also called the *transition Laplacian matrix* of G. It is symmetric and therefore has a real spectrum. Since

$$v^T \cdot \mathcal{L}_N(G) \cdot v = \left(\mathcal{D}(G)^{-\frac{1}{2}} v\right)^T \mathcal{L}(G) \left(\mathcal{D}(G)^{-\frac{1}{2}} v\right) \geq 0 \text{ by proposition 3.31}$$

we know that $\mathcal{L}_N(G)$ is positive semi–definite. By corollary 3.33, we note that $\mathcal{L}_N(G)$ has at least on eigenvalue that equals zero. We use a notation similar to definition 3.32:

Definition 3.39
Let G be a graph with n vertices. The *normalised Laplacian spectrum* of G is defined as:

$$\Lambda_{\mathcal{L}_N}(G) := \Lambda(\mathcal{L}_N(G))$$

and is the set of all eigenvalues of the normalised Laplacian matrix of G. Since this spectrum is real and $\mathcal{L}(G)$ is positive semi–definite we use also

$$0 = {}_N\lambda_1(G) \leq {}_N\lambda_2(G) \leq \cdots \leq {}_N\lambda_n(G)$$

for the eigenvalues of the normalised Laplacian matrix.

One reason why $\mathcal{L}_N$ is called normalised is that its spectrum is contained in $[0, 2]$, see [Chu94, p. 7]. This bounds are in a certain way tight, since always ${}_N\lambda_1 = 0$ and $2 \in \Lambda_{\mathcal{L}_N}$ iff an undirected connected component of the graph is bipartite. Most bounds concerning spectra of the Laplacian or the normalised Laplacian matrix can also be stated for the other. Some of these inequations can be found in [Chu94] or [Moh97].

3.6.2 The Normalised adjacency matrix

Definition 3.40
Let G be a graph such that every vertex has at least one outgoing edge. Then the *normal matrix* $\mathcal{N}(G)$ is defined as:

$$\mathcal{N}(G) := \overrightarrow{\mathcal{D}}(G)^{-1} \mathcal{A}(G),$$

where $\overrightarrow{\mathcal{D}}(G)$ denotes the out–degree matrix of G, this is a diagonal matrix which has the entries $\deg^{\rightarrow}(\cdot)$. The *normal spectrum* of G is defined as:

$$\Lambda_{\mathcal{N}}(G) := \Lambda(\mathcal{N}(G)).$$

The normal matrix is also called the *normalised adjacency matrix*. Since $\mathcal{N}$ is in general not symmetric its spectrum need not be real. We have:

Lemma 3.41
Let G be an graph on n vertices such that every vertex has at least one outgoing edge. Then its normal matrix is stochastic. It has 1 as eigenvalue. If G is undirected we gain additionally:

$$\mathcal{N}(G) = I_n - \mathcal{D}(G)^{-\frac{1}{2}} \mathcal{L}_N(G) \mathcal{D}(G)^{\frac{1}{2}} = I_n - \mathcal{D}(G)^{-1} \mathcal{L}(G) \tag{3.17}$$

Proof:
The normal matrix is nonnegative since it is the product of nonnegative matrices (proposition 3.5). Let v be a vertex. The adjacency matrix $\mathcal{A}(G)$ has exactly $\deg^{\rightarrow}(v)$ entries with value 1 and all other entries are 0 in the v–th row. So every row sum of $\mathcal{N}(G)$ equals 1 and therefore it is stochastic. Since $\mathcal{N}$ is stochastic 1 is an eigenvalue.

If G is undirected we have $\mathcal{L}(G) = \mathcal{D}(G) - \mathcal{A}(G)$, therefore we have

$$\begin{aligned}
\mathcal{N}(G) &= \mathcal{D}(G)^{-1}\Big(\mathcal{D}(G) - \mathcal{L}(G)\Big) \\
&= I_n - \mathcal{D}(G)^{-1}\mathcal{L}(G) \\
&= I_n - \mathcal{D}(G)^{-1}\Big(\mathcal{D}(G)^{\frac{1}{2}} \mathcal{L}_N(G) \mathcal{D}(G)^{\frac{1}{2}}\Big) \\
&= I_n - \mathcal{D}(G)^{-\frac{1}{2}} \mathcal{L}_N(G) \mathcal{D}(G)^{\frac{1}{2}}
\end{aligned}$$

□

Corollary 3.42
Let G be an undirected graph such that every vertex has at least one incident edge. Then its normal spectrum is real and (μ, w) is an eigenpair of $\mathcal{N}(G)$ iff $\mathcal{L}_N(G)$ has $(1-\mu, \mathcal{D}(G)^{-\frac{1}{2}} w)$ as an eigenpair.

Proof:
It is sufficient to show the equivalence, since $\mathcal{L}_N(G)$ has a real spectrum. Let $B = \mathcal{D}(G)^{\frac{1}{2}}$. The matrix $B^{-1}\mathcal{L}_N(G) B$ has the same eigenvalues as $\mathcal{L}_N(G)$ and v is an eigenvector of $\mathcal{L}_N(G)$ iff $B^{-1}v$ is an eigenvector of $B^{-1}\mathcal{L}_N(G) B$. Therefore equation (3.17) implies the equivalence. □

Since every stochastic has spectral radius one, every eigenvalue is located in the unit disc in the complex plane. We use the notation

$$\mu_1, \ldots, \mu_{n-1}, \mu_n = 1$$

for the eigenvalues the μs are sorted non descending with respect to their real parts. If G has no bipartite connected component we have $|\mu_j| < 1$ for all $j = 1, \ldots, n-1$.

Corollary 3.43
Let $G = (V, E)$ be an undirected graph with $\delta(G) \geq 1$. Two eigenvectors to different eigenvalues of $\mathcal{N}(G)$ are orthogonal with respect to the following scalar product:

$$\langle x, y \rangle_N := \sum_{v \in V} (\deg v) \cdot [x]_v [y]_v \tag{3.18}$$

Proof:
Since $\mathcal{L}_N(G)$ is symmetric we know by theorem 3.21 that eigenvectors to different eigenvalues are orthogonal (with respect to $\langle , \rangle_{\mathbf{1}}$). By corollary 3.42 we know that if v is an eigenvector of $\mathcal{L}_N(G)$ then $\mathcal{D}(G)^{-\frac{1}{2}} v$ is an eigenvector of $\mathcal{N}(G)$. Let x', y' be two eigenvectors of $\mathcal{N}(G)$ to different eigenvalues and $x := \mathcal{D}(G)^{-\frac{1}{2}} x'$ and $y := \mathcal{D}(G)^{-\frac{1}{2}} y'$. Then we have:

$$\begin{aligned}
\langle x', y' \rangle_N &= \sum_{v \in V} (\deg v) \cdot [x']_v [y']_v \\
&= \sum_{v \in V} \deg v \cdot \left(\frac{[x]_v}{\sqrt{\deg v}} \right) \cdot \left(\frac{[y]_v}{\sqrt{\deg v}} \right) \\
&= \sum_{v \in V} [x]_v [x]_w = \langle x, y \rangle_{\mathbf{1}}
\end{aligned}$$

Since x and y are eigenvectors of $\mathcal{L}_N(G)$ to different eigenvalues x and y are orthogonal and therefore $\langle x, y \rangle_{\mathbf{1}} = 0$. □

The normal matrix can be used as a model for random walks in a graph. In section 5.1 we describe this model in an application background.

Chapter 4

Cluster analysis

In this chapter we give a brief overview of cluster analysis. We are especially interested in graph clustering respectively clustering with methods from graph theory. The first section considers problem settings and aims of clustering. Then we present some selected methods and discuss advantages and disadvantages. In the last section we deal with a graph clustering method using spectral methods.

4.1 Problem settings and purpose

What is *clustering*? Giving an informal definition we could say:

> Cluster analysis is the study of recognizing natural groups within a class of entities with the help of mathematical tools.

The weak point of this definition is of course the term *natural groups*. In the past many different approaches have been made to formalise this term. This is one reason why today there exists a toolbox of different procedures rather than one method. Summarising we give the following general problem definition:

Problem 4.1 (general clustering)
Input: *a set C of entities, a weighted binary relation on C and a cost function for partitions on C*

Output: *a partition P of C such that P has minimal cost with respect to all partitions of C*

The partition P is also called a *cluster* and its components the *cluster components*. One detail which is hidden in this kind of problem description is the role of the binary relation. Its purpose is measuring the degree of relation between two entities. This can be done in a positive way, for example expressing similarity, or in a negative one, for example modeling dissimilarity or differences. Throughout the rest of this paper we consider only similarity as weighted binary relation between the entities. The binary relation is usually a complete relation. Problem 4.1 can also be seen as a maximising problem, if the cost function is replaced by a weight function or something similar.

Before we describe certain cost or weight functions in detail we state why we think that cluster analysis is a fundamental principle in science. Many scientific disciplines deal with the problem of exploring huge data sets. A general interest is to identify structures within the entities. Consider for example the classification of animals and plants in biology. Using clustering techniques one can reduce the complexity of the exploration space and identify local relations between the entities. In disciplines like biology, psychology, sociology, linguistics, geology and many more there is a general principle to identify new classifications:

1. Examine a small data space and identify certain structures or relations.
2. Formulate a hypothesis. The thesis consists of two parts: first, variables which influence the result, and second, the connection between result and variables.
3. Try to prove or disprove the hypothesis. This can be done for example by searching for a counter–example or probabilistically by considering a larger data space.
4. If the hypothesis was correct, then a new law was found, else reenter step 1 or 2 and modify current results.

In this scheme cluster analysis is often used in step 3 and sometimes also in step 1. For a more detailed introduction into cluster analysis consider for example [JD88], [Zup82], [And73] or [Har37].

4.2 Simple approaches

Considering problem 4.1 we see that the set of entities together with the weighted binary relation forms a graph with an edge weight function. Therefore we restrict ourselves to the following problem:

Problem 4.2 (Graph clustering)
Input: *a graph G with an edge weight s and a cost function c on the partitions of G*

Output: *a partition P with minimal cost with respect to all possible partitions of G*

Note that this problem description is more flexible with respect to the binary relation. The general cluster problem usually involves a complete relation on the entity set. The graph structure offers several possibilities to compact these information without loss. For example a partial relation may be complete by the calculation of shortest paths. We use also the following additional notation:

Definition 4.3
Let $G = (V, E)$ be a graph with an edge weight s and $\mathbb{P} = (C_1, \ldots, C_r)$ a partition of G. An edge which has its source and its targets in different components is called an *inter–cluster edge*. The collection of all such edges is denoted by $\mathbb{I}_{\mathbb{P}}$ and its weight (with respect to s) is defined by:

$$s(\mathbb{I}_{\mathbb{P}}) = \sum_{e \in \mathbb{I}_{\mathbb{P}}} s(e)$$

The quotient

$$\frac{s(\mathbb{I}_{\mathbb{P}})}{\sum_{e \in E} s(e)}$$

is the ratio of uncovered weight to the total weight. Edges which are not inter–cluster edges are *cluster edges* or *inner–cluster edges*.

Although the following approaches were seldom stated in terms of graphs in the literature we will use graph notation to describe and discuss them. Therefore assume for the input: $G = (V, E)$ as a graph with an edge weight s.

Single linkage

Approach 1 (Single linkage)
As additional input let $\Delta \in \mathbb{R}$ be a threshold value. Denote the set of all edges with weight greater than Δ by E'. Then the undirected connected components of (V, E') form the cluster partition.

This method simply eliminates all elements of the binary relation which have a degree of similarity below Δ. It is quite economical since its runtime is $\mathcal{O}(|V| + |E|)$ and we need no additional space. The weak point of this procedure is the choice of Δ, since a bad choice may lead to partitions of too small or too great size. If the component size is too large then it implies that vertices belong to the same component although they have little in common. This effect is called *chaining*, since one connecting path is sufficient to put two vertices into the same component. Consider the following example. Figure 4.1 shows the input graph. The degree of similarity is expressed by the thickness of the edges. So if we choose the threshold value Δ too small then we will obtain one component. But the vertices with a square shape and those with a rhombus shape have little in common. In fact the only connection is a path containing circle shaped vertices. A more intuitive partition would be using all square–shaped vertices as a component, all circular vertices as another component and a final component with all rhombus–shaped vertices.

Because of this chaining effect single linkage is seldom used on its own. Often it is used for a preprocessing step. The graph created by single linkage is also called the Δ–*threshold graph*.

Complete linkage

Chaining effects depend on the existence of few edges with a relatively high degree of similarity. To thwart this we consider more edges. An optimal situation is that every possible edge exists and has a high degree of similarity. This is the idea behind complete linkage and can be best recursively described:

Approach 2 (Complete linkage)
As additional input let $\Delta \in \mathbb{R}$ be a threshold value. Start with a partition where every single vertex is a component on its own. If there are two components such that their union induces a complete subgraph with edges of

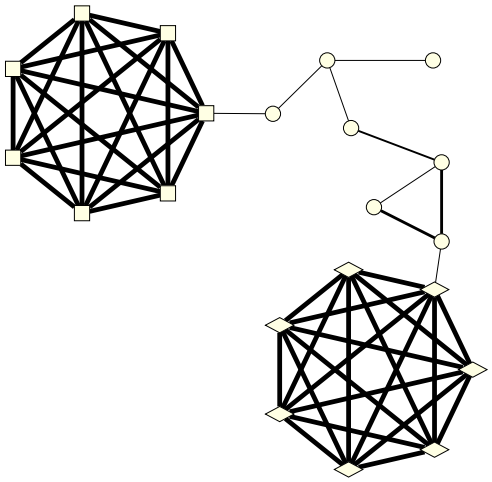


Figure 4.1: Chaining example

weight greater than Δ, then join them into a single component. If no pair of components exists which fulfills the condition output the current partition.

Note that this approach is not equivalent to finding a maximal complete subgraph with a certain edge weight since these maximal complete subgraphs are in general not disjoint. Consider the graph in figure 4.2. The maximal complete subgraphs are $\{1, 2, 3\}$ and $\{3, 4, 5\}$, but they are not disjoint. Therefore clustering by maximal complete subgraph would require overlapping partitions. But this has also a drawback. Overlapping components have many vertices in common and this is usually not desired. Figure 4.3 shows such a graph. Every maximal complete subgraph contains $\{0, 1, 2, 3, 4, 5\}$. The

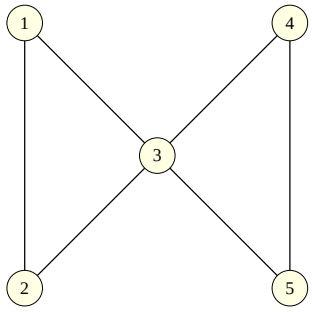


Figure 4.2: An example for a graph with non disjoint maximal complete subgraphs

bold edges are only a visual help and are all edges which connect two elements of $\{0, 1, 2, 3, 4, 5\}$.

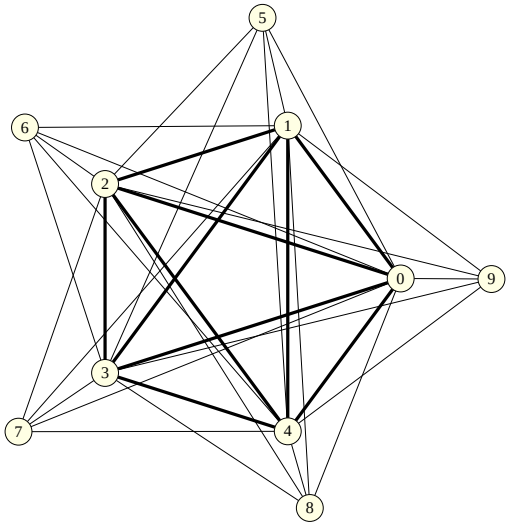


Figure 4.3: Example of a graph where maximal complete subgraphs have many vertices in common

Complete linkage is more complex and requires more run time. Testing if the union of two partition components P_1, P_2 form a complete subgraph requires $\mathcal{O}(|P_1| \cdot |P_2|)$. This implies that complete linkage runs in $\mathcal{O}(|V|^5)$. It needs no additional space. Like single linkage one difficulty is the choice of Δ. Wrong choices may lead to components with relatively small size.

Single and complete linkage are opposed in their methods. Both have difficulties with certain situations. In the next section we will consider such problems. Before we move on we briefly show some other approaches. These use a restricted version of problem 4.2, namely:

Problem 4.4 (*k*–graph clustering)

Input: *a graph G with an edge weight s, an integer k and a cost function c on the partitions of G*

Output: *a partition P of size k with minimal cost with respect to all possible partitions of G with size k*

Note the following difference: In problem 4.4 the size of the final partition is fixed. Sometimes the problem is weakened to bound the size of the partition by k only either as an upper bound or as a lower bound. The following functions are often considered in the literature:

- maximal covered weight:
  As cost function we use the number of inter–cluster edges. This implies that the final partition has as few inter–cluster edges as possible. A weighted version uses the weight of the inter–cluster edges as cost. Minimising this function implies that the sum of degrees of similarity between vertices which are adjacent but in different cluster components is as low as possible.
- inter–cluster distance:
  Here we use the average or the maximal weight of the inter–cluster edges as cost. The intentions are similar to maximal covered weight.
- minimal diameter:
  The diameter d of a graph is the minimal number of edges such that every pair of vertices can be connected by a path of length at least d. As cost function we choose the maximum diameter of all partition components. A similar approach is the content of [HS00].
- minimal average distance:
  The cost is the sum of the average distance of two vertices in a cluster component. This distance can simply be the minimal number of edges connecting them or the weight of such a path.

All approaches have in common that they try to ensure that the cluster components are highly connected. Every method measures this in its own way. For more details and other procedures see also [Fas99] and [Gon85].

4.3 Errors and quality

Error detection

Some complex calculation or data ascertainment is often the basis for the edge weight function s which realises the degree of similarity. These methods are usually not free of errors. Finite arithmetic is another reason for inaccuracies.

Error detection and correction mainly depends on the input data and the applied proceedings. It is a rather technical problem which should be dealt with when considering real data. We present only a very limited point of view. Nevertheless this topic is quite related to quality measurement which is the next addressed topic.

We distinguish between two sorts of errors: positive and negative. The first kind is that two vertices are adjacent or have a high degree of similarity although they have little in common. It is called in this manner since affected relations are overestimated. The second kind is the reversed scenario. Two vertices are not adjacent or have a small degree of similarity although they are highly similar. Comparable to positive errors this sort is called in such a way since affected relations are underestimated.

We state only some general characteristics and only for single and complete linkage. These two are opposite in their approaches and this can also be found in the error behaviour. If there are many positive errors single linkage will create few components but with great size. Hence a final partition will be too raw or improper. On the other hand, if there are many negative errors complete linkage will not find many complete subgraphs, since edges are "missing". Therefore a final cluster will be too fine. This leads to an important question:

> "Given two clusters, which one is the better one?" respectively
> "How can the quality of a cluster be measured?"

Quality measure

It is quite hard to define a general quality measure for clusters. Although it may even be possible to prove certain results in special situations. Especially if we have much information about the input, we keep our general viewpoint instead. All presented approaches and cost function can easily be fooled. This means that for every method simple counter–examples exist where "wrong" clusters are calculated. For details see [KVV00]. In this reference we also found an intuitive measure which does not seem to be easily fooled:

Definition 4.5
Let $G = (V, E)$ be a graph with an edge weight s. Let S be a nonempty

proper subset of V. Then the *partition weight* is defined as:

$$\Xi_s(S) := \sum_{\substack{e \in E \\ \text{source}(e) \in S}} s(e).$$

We assume that $\Xi_s(\cdot) \not\equiv 0$ for all nonempty proper subsets of V. Since S is nonempty and proper it induces a cut. So we define the *conductance* of S as

$$\phi(S) := \frac{\sum_{e \in \partial S} s(e)}{\min(\Xi_s(S), \Xi_s(\overline{S}))}$$

and the *conductance* of the graph G as the minimal occurring conductance:

$$\phi(G) := \min_{\emptyset \neq S \subsetneq V} \phi(S).$$

Note that the conductance is invariant under complementation. Let S be a nonempty proper subset of the vertex set of a graph then

$$\phi(S) = \phi(\overline{S}).$$

Let $\mathbb{P} = (C_1, \ldots, C_r)$ be a clustering of G. Let $G_1, \ldots, G_r$ be the subgraphs induced by $C_1, \ldots, C_r$. Then we define the *conductance* of $\mathbb{P}$ as:

$$\phi(\mathbb{P}) := \min_{1 \leq j \leq r} \phi(G_j).$$

Next we state some intuition of the conductance model. Conductance measures a certain quality of cuts. It is similar to minimal cuts, but additionally respects the degree of balance induced by the two cut components. Minimal cut can be seen as global bottle necks, for example how big the smallest common boundary between two parts of a graph is. Recall the computer network in 2.3.1. There, one of the occurred questions was: How many connections must be cut to have at least two separated networks? This corresponds to a bottle neck of the network. It is the smallest part splitting the network at least into two parts. These global bottlenecks often split the graph in very unbalanced pieces. Some vertices into one component and all others in the other component. This lack is removed in the conductance model. Therefore the conductance also measures the degree of balance of the induced components. Note that the conductance of a graph is also called the CHEEGER

constant and has a physical analogue, see [SR97]. Figure 4.4 shows the graph G_3. In this graph the minimal cut and the cut induced by the graph conductance are different. Each edge has weight 1 and so we omit a labeling in the figure. Like in a previous example (see figure 2.5) we use bold lines which cross the cut edges. The solid line is used for the conductance cut and the dashed one for the minimal cut. To emphasise the conductance cut we use different colors for the vertices in different partitions. Figure 4.5 displays all pairs $(|S|, \phi(S))$ for nonempty proper subsets S of the vertex set of G_3.

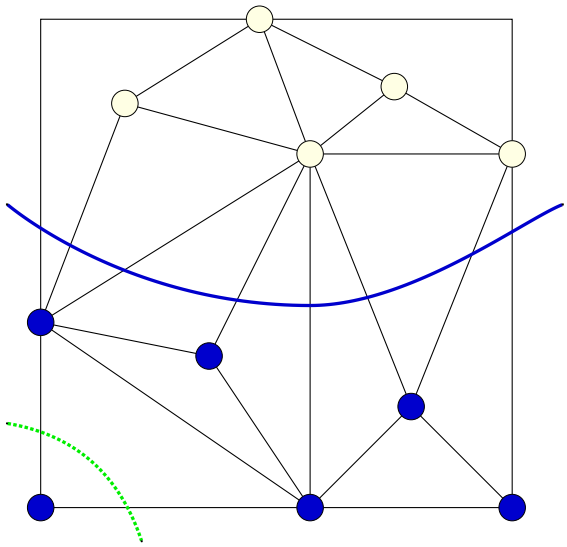


Figure 4.4: Graph G_3 which has different minimal and conductance cuts

In [KVV00] the conductance together with the uncovered weight ratio are used as a cluster measure. The cluster's conductance is maximised while the uncovered weight ratio is simultaneously minimised. The authors used the uncovered weight ratio due to the fact that some clusters may have many components with high conductance and few with very low conductance. They show that this optimisation problem is $\mathcal{NP}$–hard[1]. It is even $\mathcal{NP}$-hard to calculate the conductance of a given graph. They presented a poly–logarithmic approximation algorithm for their cluster optimisation problem.

[1] We use $\mathcal{NP}$–hard for optimisation problems which associated decision problem is $\mathcal{NP}$–complete. For further information see [GJ79].

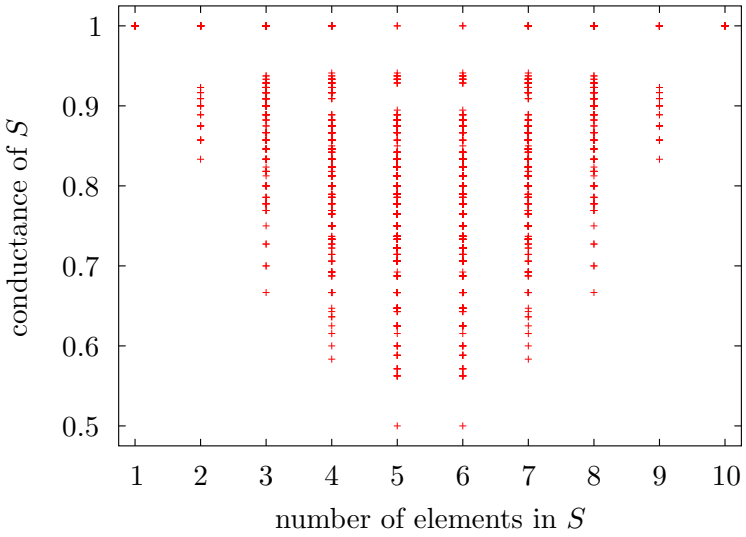


Figure 4.5: All possible conductance values of G_3

4.4 Spectral clustering

In this section we consider first of all a cluster approach which is based on theorem 3.37. We combine problem 3.36 with problem 4.4. Informal this is just placing the vertices of a graph on exactly k different positions of a geometrical construct such that the induced costs are minimal. The formal description is given in problem 4.6.

Problem 4.6 (discrete k–partition d–dimensional placement)
Input: *a graph $G = (V, E)$, a cost function cost: $\{-1, 1\}^{|V| \times d} \to \mathbb{R}$, integers $d, k \leq |V|$ and a fixed vertex–ordering $v_1, \ldots, v_n$ with $n := |V|$*

Output: *a position function $\varrho\colon V \to \{-1, 1\}^d$ such that $\varrho(V)$ has exactly k elements and the matrix A is a global minimum of cost with respect to all position functions with k different elements in the image and*

$$A := \begin{pmatrix} \varrho(v_1) \\ \ldots \\ \varrho(v_n) \end{pmatrix}.$$

Since $\{-1,1\}^d$ has 2^d elements it is necessary that

$$1 \leq k \leq 2^d \qquad \text{or} \qquad \log_2 k \leq d$$

holds. If we just count the number of cross edges (edges which connect two vertices with different positions) problem 4.6 is $\mathcal{NP}$–hard, see for example [CK94, ND13]. Considering a fixed k the problem becomes solvable in polynomial time. For $k = 2$ it is a geometrically version of the minimum cut, problem 2.4. The number of vertices which have the same position is arbitrary in problem 4.6. This often leads to the situation where many vertices are placed on the same position. Some applications and algorithmic procedures, for example divide and conquer techniques, require that the graph is equally split, so we introduce a subproblem where we also fix that:

Problem 4.7 (discrete balanced k–partition d–dim. placement)
Input: *a graph $G = (V, E)$, a cost function cost: $\{-1,1\}^{|V|\times d} \to \mathbb{R}$, integers k, d and a fixed vertex–ordering $v_1, \ldots, v_n$ with $n := |V|$*

Output: *a position function $\varrho\colon V \to \{-1,1\}^d$ such that $\varrho(V)$ has exactly k elements, at least $\lfloor \frac{n}{k} \rfloor$ and at most $\lceil \frac{n}{k} \rceil$ vertices have the same position and the matrix A is a global minimum of cost with respect to all such position functions and*

$$A := \begin{pmatrix} \varrho(v_1) \\ \cdots \\ \varrho(v_n) \end{pmatrix}.$$

Note that problem 4.7 is still $\mathcal{NP}$–hard. It may be seen as combination of Minimum k–Cut ([CK94, ND13]) and Minimum Cut into bounded sets (problem 2.6, [GJ79, ND17]). Both are $\mathcal{NP}$–hard on their own. As cost function we use the total edge length, i.e. the sum of the distances of adjacent vertices and consider the special instance $d = 1$ and $k = 2$ first. If we relax ϱ's range to $[-1, 1]$ we see that a normalised eigenvector v of $\lambda_2(G)$ is a nontrivial solution. Therefore v can be used as approximation:

Approach 3 (Laplacian bisection)
Let $G = (V, E)$ be a graph with n vertices. Calculate an eigenvector x_2 to $\lambda_2(G)$. Sort the vertices according to the entries of x_2 and put the first $\lfloor \frac{n}{2} \rfloor$ vertices in the first component and all others in the second component.

Note that x_2 has always a sorting such that each component has (exactly) one undirected connected component. For details see [Fie75]. In section 8.1 we show that approach 3 can be fooled. Nevertheless it is a helpful method. A little survey about extending approaches can be found in [AK95, ch. 4]. Next we describe some general methods to generalise approach 3:

- recursion:
  The approach 3 is recursively called on the introduced components. This may be only practical for certain choices of k.
- more eigenvectors:
  More eigenvectors are calculated. These are typically normalised and pairwise orthogonal. Often an eigenvector is calculated for each dimension.
- combined eigenvectors:
  Like in more eigenvectors more eigenvectors are calculated. But additionally some are combined to new vectors and the whole set of these vectors is then used to calculate the positions.

These different methods can be summarised in the following approach:

Approach 4 (general spectral clustering)

i.) calculate a matrix M of the input graph G with respect to the weight function s.

ii.) calculate eigenvectors of M: $x_1, \ldots, x_k$

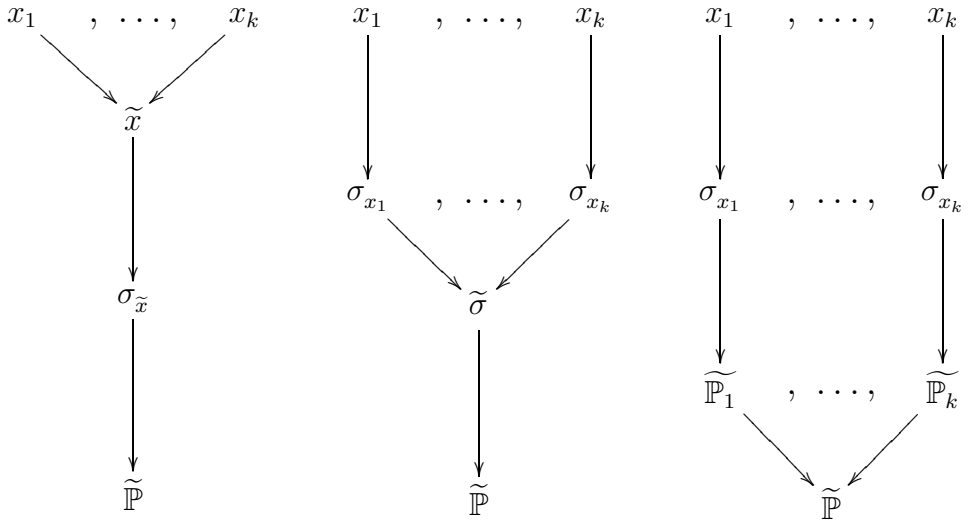
iii.) interpret $x_1, \ldots, x_k$ as cluster

There is a fair number of ways in which the eigenvectors can be interpreted. Next we introduce a helpful notation:

Definition 4.8
Let $G = (V, E)$ be a graph and x a vertex weight function. We associate to x an edge weight function σ_x defined by:

$$\sigma_x(e) := \left| [x]_{\text{target}(e)} - [x]_{\text{source}(e)} \right|.$$

If we interpret x as potential function on the vertices then σ_x can be interpreted as difference between two potentials. The diagram Ms displayed in



(a) combining the potentials

(b) combining the potentials' difference

(c) combining auxiliary clusters

Figure 4.6: General methods to interpret $x_1, \ldots, x_k$ as cluster

figure 4.6 show the three possibilities to calculate the final partition $\widetilde{\mathbb{P}}$ of the clustering.

In chapter 6 we will specify certain instances of approach 4 and the interpretation of $x_1, \ldots, x_k$ as cluster. We have listed some procedures to calculate eigenvalues and eigenvectors in appendix B. Together with some lemmas in appendix C we can state some of these procedures in graph notation.

Chapter 5

MCL process

In the first section of this chapter we introduce another cluster approach, called MCL process. Although being more complex than the previous approaches, it hardly needs any parameters to fix cluster properties like the size of the cluster or the size of the components and still produces "good" clusters. We examine several example and give a brief summary in the middle part. Finally we have a closer look at memory usage.

5.1 MCL process

MCL stands for Markov Cluster and is introduced in [vD00]. Van Dongen considers the general graph cluster problem:

Problem 5.1 (general graph clustering)
Input: *an undirected graph $G = (V, E)$ with an edge weight s, expressing the degree of similarity, and the condition $\forall\, e \in E\colon s(e) \geq 0$.*

Output: *a partition P where the components model the natural groups of G with respect to s*

The problem description 5.1 lacks a cost function which indicates the optimality of the calculated partition. Instead van Dongen used an alternative paradigm to model natural groups:

> A random walk in G that visits a dense cluster (component) will likely not leave the cluster (component) until many of its vertices have been visited. [vD00, p. 6]

Note that this paradigm also tries to ensure that the cluster components are highly connected. This is a common property of the presented cost function in section 4.2 on page 57. Next we consider in which way this paradigm can be realised.

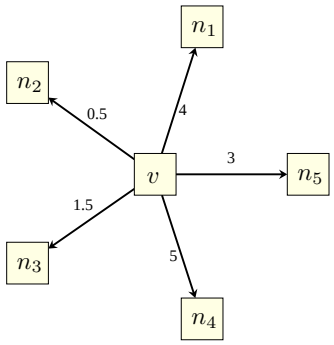


Figure 5.1: Neighbourhood of v

First we fix the term "random walk". A random walk in G is a path starting at a random vertex and then repeatedly moving to a neighbour of the last chosen vertex in a uniform random way with respect to s. For example consider the following situation displayed in figure 5.1. We have chosen the vertex v last and now move on to one of its neighbours. The neighbours then have the following probability to be chosen:

n_1	n_2	n_3	n_4	n_5
$\frac{2}{7}$	$\frac{1}{28}$	$\frac{3}{28}$	$\frac{5}{14}$	$\frac{3}{14}$

We are only interested in the probability to reach a certain vertex from an arbitrary start vertex and not the order of the traversed vertices, so we can use the weighted normal matrix $\mathcal{N}(G)$ of G. Therefore let v be the start vertex and x a real vector indexed with the vertices of G[1] and

$$[x]_w := \begin{cases} 1 & \text{if } w = v \\ 0 & \text{else} \end{cases} .$$

Then we define the following sequence of vectors:

$$x^{(0)} := x \qquad \text{and} \qquad x^{(k)} := \mathcal{N}(G) \cdot x^{(k-1)} \qquad \text{for } k \geq 1.$$

By induction we see that $x^{(k)} = (\mathcal{N}(G))^k \cdot x^{(0)}$. In analogy to lemma 3.25 we know that the entries of $(\mathcal{N}(G))^k$ are the probability of random walks of

[1] We assume $\mathcal{N}(G)$ and x have the same vertex ordering.

length k. So $x^{(k)}$ contains the probabilities to reach a certain vertex with a random walk of length k starting in v. For two reasons it is not satisfying to consider only paths of a fixed length. First, certain graph structures influence these paths. For example bipartite graphs: Paths of even length reach only the same component and paths of odd length only the other component. Therefore the random effect is weakened. Furthermore the information about paths of length k is somehow independent of the information of shorter paths. To avoid both effects VAN DONGEN adds a loop to each vertex in G with a certain degree of similarity. Now the matrix $(\mathcal{N}(G))^k$ contains the probability of random walks of length at most k. This is due to the fact that a random walk may now stay at some vertices for a while and then moves on. Such walks are comparable to walks without such "out-times" at vertices. Since the powers of $\mathcal{N}(G)$ retain the accumulated information it is sufficient to calculate the following sequence:

$$\mathcal{N}(G), (\mathcal{N}(G))^2, (\mathcal{N}(G))^4, \ldots, (\mathcal{N}(G))^{2^k}.$$

To calculate $(\mathcal{N}(G))^{2^k}$ we require $\mathcal{O}(\log_2 k \cdot T_{\text{mul}})$ time, where T_{mul} is the time to multiply two $|V| \times |V|$ matrices. In general we have $T_{\text{mul}} \in \mathcal{O}(|V|^3)$. Note that $\mathcal{N}(G)$ is stochastic and so each power of it.

Next we consider the proportion of those probabilities. A random walk which enters a dense cluster component stays in this component for a while. So There have to be many edges with high probability. If we raised their weight to a power greater than one, then the situation should not really change. Note that if we raise the probabilities to a certain power, we have to linearly rescale them afterwards to keep the matrix stochastic. This method is comparable to enlarge high degree of similarity and to decrease low degree of similarity. This is a kind of natural support, consider for example road networks. Much frequented roads become expanded and roads which are seldom frequented get reduced. With these tools we can describe VAN DONGEN's MCL process, see algorithm 1.

We do not state a formal notation for '*is suitable*', used in algorithm 1. For details concerning this point see [vD00, ch. 5]. The interpretation of $\mathcal{N}$ as clusters is another important step in the MCL process, so we have a closer look. We restrict ourselves to the situation where $e = r = 2$. VAN DONGEN considers certain "limit" cases. He investigates two different limit types: fixed–point and periodicity. For the moment we denote by $\mathcal{N}^\infty$ the last matrix calculated. In the first case, the fixed–point situation, he shows that $\mathcal{N}^\infty$

Algorithm 1: MCL algorithm

Input : a graph $G = (V, E)$, an edge weight s and integers $e, r > 1$
Output: a cluster $\mathbb{P}$ (here an overlapping partition)
begin
  if necessary add self–loops to the vertices in G and update s
  calculate the weighted normal matrix $\mathcal{N}$
  while $\mathcal{N}$ *is not suitable* **do**
    $\mathcal{N} \leftarrow \mathcal{N}^e$
    forall $(u, v) \in V \times V$ **do**
      $[\mathcal{N}]_{u,v} \leftarrow [\mathcal{N}]^r_{u,v}$
    rescale each row of $\mathcal{N}$ such that $\mathcal{N}$ is stochastic
  interpret $\mathcal{N}$ as cluster
end

has a certain structure which is in a natural way similar to an (overlapping) cluster structure. In this structure most rows have only one nonzero entry. Furthermore $[\mathsf{G}(\mathcal{N}^\infty)]$ is almost a forest with very few cycles; thus we could use the connected components of $[\mathsf{G}(\mathcal{N}^\infty)]$ to form the cluster components. Why did we output an overlapping cluster? This is due to the fact that VAN DONGEN did not use the forest structure itself, but something similar. In particular he extracts a subset of the vertex set, which he calls *attractors*. They form the cluster "back bone", this means each attractor creates a cluster component on its own. All other vertices belong to the same component as their neighbours with respect to $\mathsf{G}(\mathcal{N}^\infty)$. Note that $\mathsf{G}(\mathcal{N}^\infty)$ is a directed graph and the neighbours, VAN DONGEN considered, are comparable to the set $\mathrm{N}^{\rightarrow}$ of the vertices in $\mathsf{G}(\mathcal{N}^\infty)$. Using this method it is possible that a vertex is adjacent to two different attractors or two vertices belonging to different components. For the second case, the periodic situation, VAN DONGEN shows that these matrices are often numerically instable or are very special matrices. Therefore he conjectures that "real world data" always results in a fixed–point case.

5.2 Examples for the MCL process

5.2.1 Random geometric graphs

First we consider a random geometric graph. Graphs of this type are constructed in the following way:

- Choose an arbitrary d–dimensional grid and a real threshold value Δ.
- Insert randomly n points $v_1, \ldots, v_n$ into the grid, such that two points do not have the same position. These points form the vertex set.
- Insert an edge if the distance between two points is less than or equal to Δ.
- Each edge receives the distance's reciprocal as edge weight.

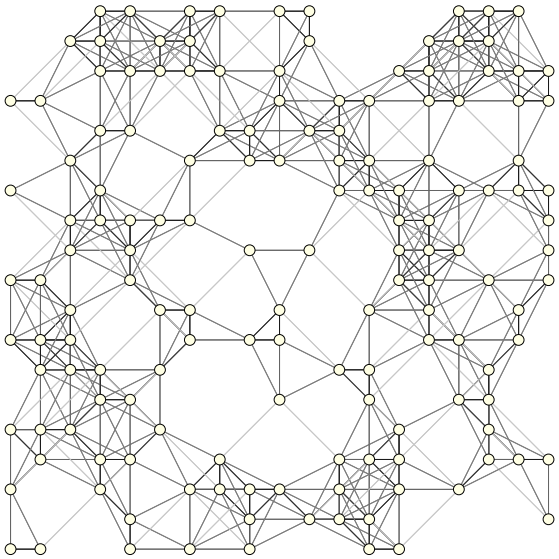


Figure 5.2: An example of a random geometric graph G_4

Figure 5.2 shows such an example graph. We refer to it as G_4. It is constructed with $n = 150$ in a 18×18 grid and $\Delta = \sqrt{8} \cdot u$ where u is the smallest

distance between two grid points. The color of the edges corresponds to their weight, this means dark color expresses high weight and light color expresses low weight. We used algorithm 1 with $e = r = 2$.

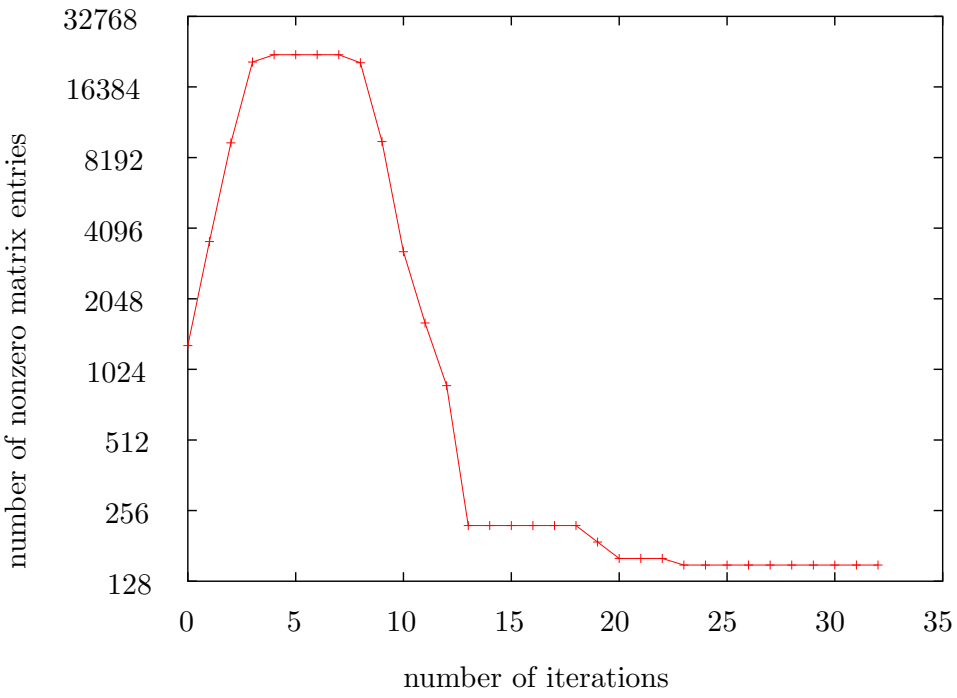
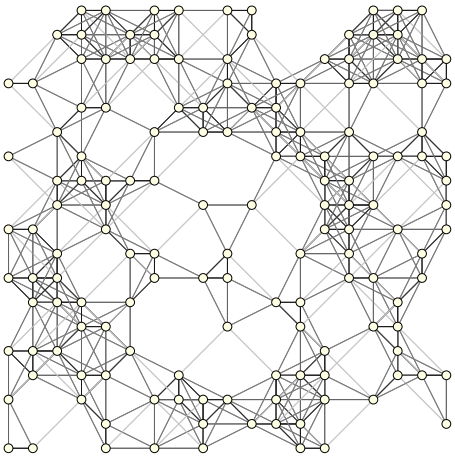
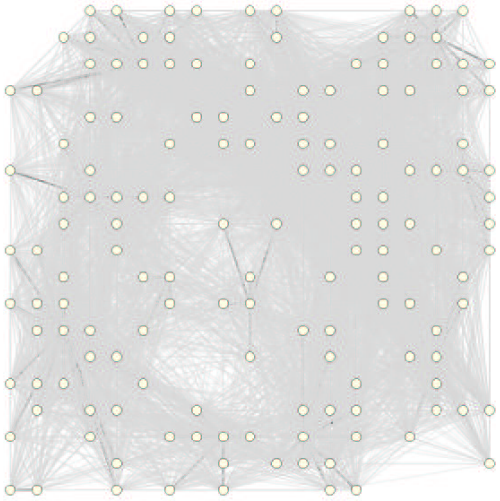


Figure 5.3: number of nonzero entries of $\mathcal{N}$ during the MCL process of graph G_4

Figure 5.3 displays the number of nonzero matrix entries (of $\mathcal{N}$) during the iterations. The matrix has $150^2 = 22500$ entries and the input graph has 647 (undirected) edges. We implemented the MCL process for test purposes. In figure 5.4 and 5.5 different stages of this MCL process are drawn. Note that $\mathsf{G}(\mathcal{N})$ is a directed graph and for the presentability of these stages we omitted the arrow heads and all edges with zero weight in the drawings.

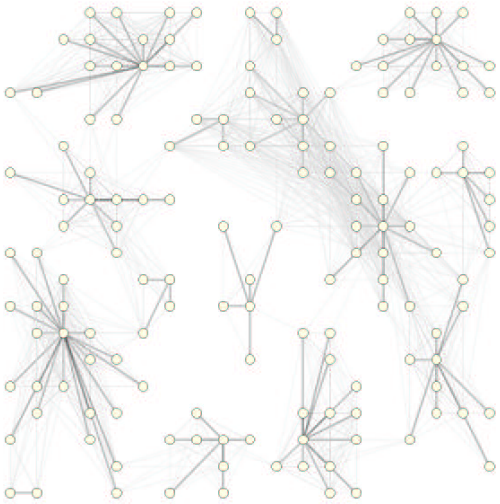


(a) iteration 0, 1294 nonzero entries in $\mathcal{N}$



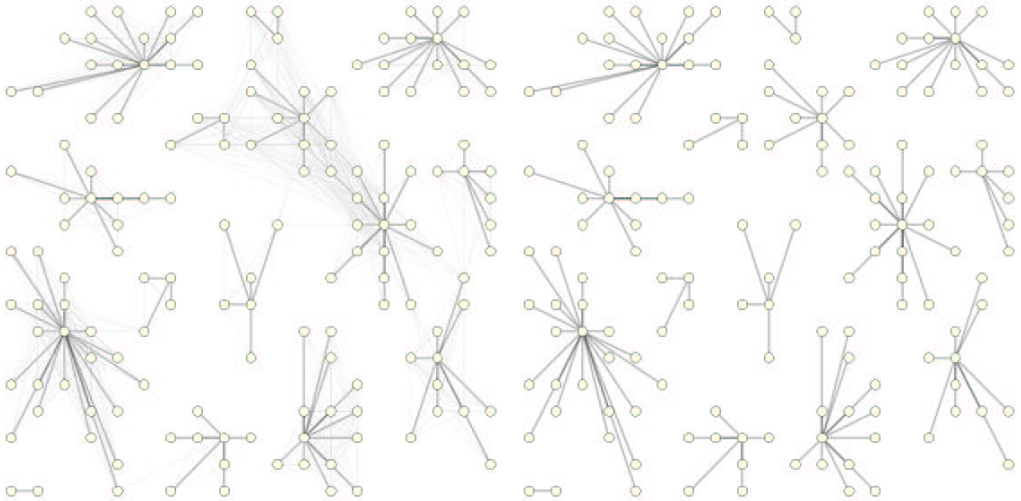
(b) iteration 9, 9583 nonzero entries in $\mathcal{N}$

(c) iteration 10, 3249 nonzero entries in $\mathcal{N}$



(d) iteration 11, 1619 nonzero entries in $\mathcal{N}$

Figure 5.4: Different states of the MCL process of G_4, part I/II



(a) iteration 12, 873 nonzero entries in $\mathcal{N}$

(b) iteration 24, 150 nonzero entries in $\mathcal{N}$

Figure 5.5: Different states of the MCL process of G_4, part II/II

5.2.2 Graphs created by statistical analysis

We also used graphs from other classes than random geometric graphs as input. Within the scope of another project we analysed different sets of entities. The main contribution was to analysis temporary changes in a network. The foundation of the network was based on newspaper articles which were available at `www.yahoo.com`. The network was generated by a centering resonance analysis presented in [CKMD02]. We used the betweenness centrality[2]. to model the importance of the vertices. So to each vertex a vector is associated which contains its centrality indices determined over course of time. We investigated in several different topics. One aspect was quite similar to clustering: "Can these vertices be automatically grouped, such that each group has a similar behaviour over time?" From this input we constructed a graph in the following way:

- For each vertex in the input network we assigned a vertex.

[2]for details see [WF94]

- To each pair of vertices we assigned an edge. Its weight was the correlation coefficient[3] of the corresponding vectors.

The correlation is a measure for linear dependency. Since the correlation is contained in the interval $[-1, +1]$ it is not suitable for the MCL process. Therefore we considered the correlation and added 1. For our example we had used 199 entities. Figure 5.6 displays the number of nonzero entries of $\mathcal{N}$ during MCL processes with different values for r. The exponent e was fixed at 2.

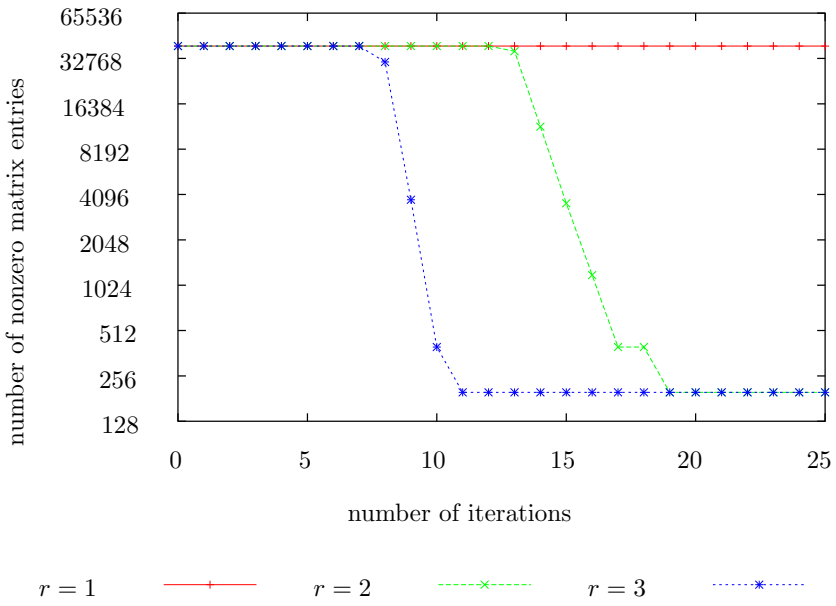


Figure 5.6: number of nonzero entries of $\mathcal{N}$ during MCL processes

5.3 Summary of the MCL process

This almost completes our discussion of the MCL process. Before we continue we briefly summarise why we think that the MCL process is so remarkable:

[3] For details see [DHJ+95, sec. 11.2] or [FKPT01, sec. 3.4].

First we have no parameters which fix any cluster property. Thus we do not have to choose any threshold parameters or anything else that influences the clustering process and can produce undesired artifacts. Van Dongen also showed that the input parameters e, r directly affect the cluster granularity, this means both the cluster size and the size of the cluster components. Regardless there are some parameters in order to influence the cluster properties. Another important point which we did not address, is convergence and stability of the process. Van Dongen proved that the fixed–point case is numerically stable and small perturbations in the input data s create the same cluster or refinements. Informally, he showed that the convergence is quadratically in the fixed–point case. These results can be found in [vD00, ch. 6,7]. Algorithm 1 needs a complex data structure and consists of some simple loops. Since no cost function is minimised the problem to calculate an optimal solution cannot be $\mathcal{NP}$–hard. Its complexity is polynomial in the size of the input parameters and the iteration depth, introduced by *is suitable*. Since the convergence is quadratically in the fixed–point case we can assume that the iteration depth is also polynomial in the size of the input parameters.

Van Dongen's approach has some weak points too. The first is scalability. We require $\mathcal{O}(|V|^2)$ space. Even if $\mathcal{N}(G)$ is sparse its sparsity is not preserved. During the process the matrix $\mathcal{N}$ may have very few zero–entries. Other faults are missing guarantees for the limit cases and optimality for the calculated cluster. Another limitation is non dense cluster components. The paradigm used by van Dongen did not consider non dense cluster components. For example, if a component has a large diameter it is often split into smaller pieces.

5.4 Memory usage

We think that the MCL process is an excellent and very useful method in cluster analysis. It involves only matrices and thus has quadratic memory usage. This space usage may be manageable in most situations, but few scenarios may require approaches which get by with linear space.

5.4.1 Example: "offline" WWW analysis

Consider for example the situation, where we have parts of the WWW[4] "locally" stored and want to analyse this data. Each vertex represents a site and every edge corresponds to a hyperlink between two sites. One important question is: "Can we group these sites automatically such that each group concerns the same topic?" We assume that we somehow calculated a measure for the degree of similarities. This input graph is generally sparse. So the adjacency matrix of it would require much space. Since we need at least $\mathcal{O}(n^{2+\varepsilon})$ for some $\varepsilon > 0$ to multiply two $n \times n$ matrices, we also would have to wait a while for a step in the MCL process. For sparse matrices we can use an algorithm using only the graph structure. Let $H = (V, E)$ be an input graph, then we can calculate $\mathsf{G}\left(\mathcal{A}(H)^2\right)$ in $\mathcal{O}(|V| \cdot \Delta(H)^2)$ time (see lemma C.1). But the matrix $\mathcal{N}$ loses its zero entries quite fast, so that the graph structure holds no more benefit against the matrix structure. Another problem may appear as well: In section 2.4, where we have discussed different graph representations, we have assumed that accessing any entry of a matrix requires one time unit, this is in general true as long as the matrix remains in the main memory[5]. If the matrix reaches a certain size we cannot keep it in main memory and therefore we have a slower access time which is not desired. Even if the graph does not fit into the main memory as well, we will benefit from the fact that we work locally, this means we consider a vertex and its neighbourhood. During its calculation we can load the next vertex and its neighbourhood into the main memory in the background.

Real instances

Such a scenario is not only fictitious. Kleinberg considers in [Kle99] methods to improve search procedures in the WWW and Henzinger considers different algorithms and information retrieval for the Web, for examples see [Hen].

In the course of some other projects we extracted a subgraph of the WWW for ourselves. This was done in a similar way as Kleinberg proclaimed in [Kle99]. This subgraph was built to the search term *"java"* and contained 5414 vertices and 15519 directed edges. Thus this graph had only 0.0529% of all possible edges. Figure 5.7 displays this graph. For the horizontal direction

[4] World Wide Web

[5] Memory which has a very good access time.

we simply used a vertex enumeration and for the vertical direction we used a measure comparable to pagerank[6]. The edges are drawn transparent so we can "recognise" concentrations.

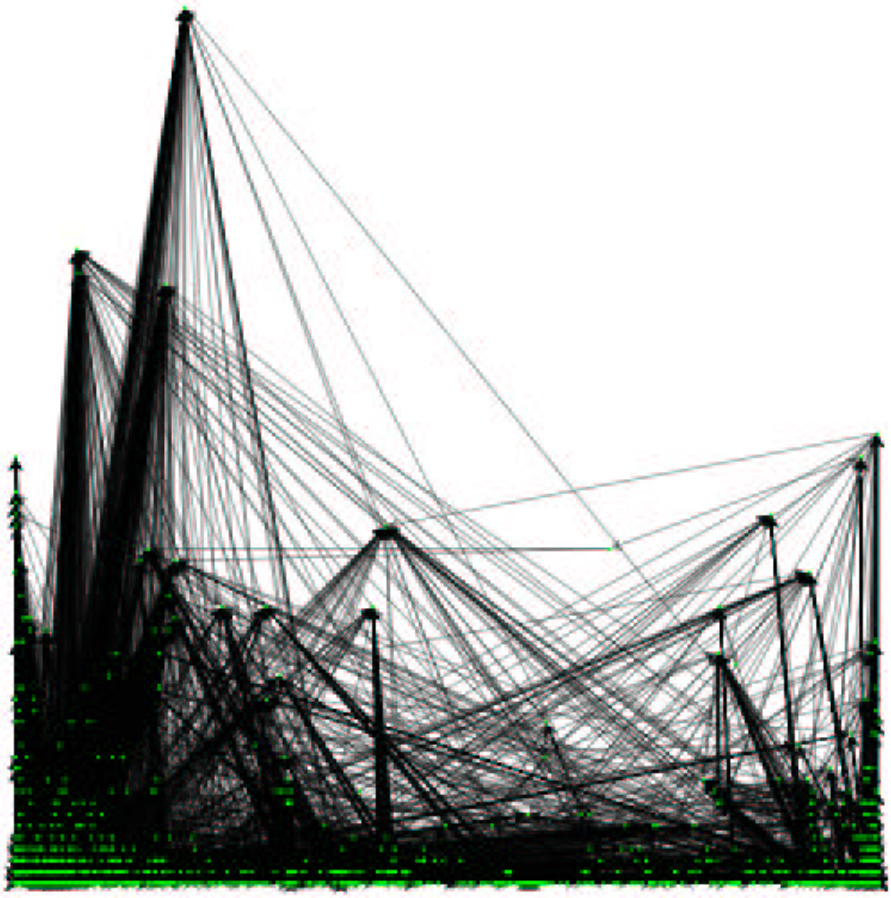


Figure 5.7: Induced subgraph of the WWW

[6] A measure/technique for ranking web pages.

5.4.2 Vector model

In order to keep the space complexity linear we consider a *vector model* for the following techniques. This is just a vertex weight function, respectively a family of vertex weight functions. It is named in this way since these functions are often represented by vectors indexed with the vertices or the entities. It is quite common in cluster analysis because it requires only linear, to be precise $\mathcal{O}(|V|)$, additional space. The degree of similarity function s is often based on such vectors, expressing attribute valuations for the entities. For example in a computer network each terminal has an attribute "age" or "average down time".

Chapter 6

Minimum spanning tree approach

In this chapter we present our own approaches. These have linear space complexity and are special instances of the general spectral approach, approach 4. First we introduce one of these methods and analyse it. Then we show some variations. Here we describe conceptions and abstract procedure. The next chapter, especially section 7.1, presents details concerning implemented algorithms and used options.

6.1 Minimum spanning tree approach

We also use problem description 5.1, but do not state a paradigm. Instead we use the MCL process as orientation and thus use indirectly VAN DONGEN's paradigm. Approach 4 is used as skeleton. Note that this general formulation contains no clue neither for the reduced space requirement nor for a transformation of the MCL process. Although approaches like this have a fundamental weak point. This is step (iii). In this step we have to transform a mapping from the vertex set with range in a non discrete space into a mapping with range in a discrete finite set, namely the cluster set, and should maintain most information. This transformation is comparable to solving another cluster problem. Therefore we have a kind of infinite loop. The only advantage is that we have additional information about certain

eigenvectors of certain matrices. For example consider the partitioning properties of eigenvectors to eigenvalue λ_2 (of the Laplacian matrix of the graph), see theorem 3.37. Approach 3 is such an instance of our general approach 4 where additional information is used for the transformation step.

We consider undirected or bidirected connected input graphs. If the graph is bidirected then the edge weight function s is symmetric, this means that the weight of every edge is equal to the weight of its reversal edge.

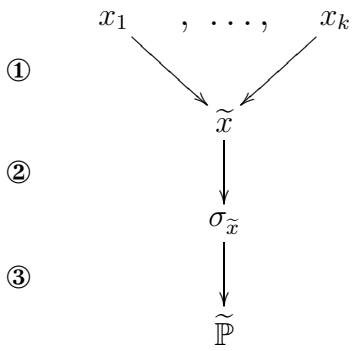


Figure 6.1: Procedure for step (iii)

In analogy to the MCL process we use the normalised matrix $\mathcal{N}$ of the input graph. We choose the eigenvectors to the eigenvalues $\mu_{n-k-1}, \ldots, \mu_{n-1}$. All μs are real (corollary 3.43) and we will only use positive eigenvalues. Typically we have $k = 2$ or $k = 3$. Our aim is to have a method for step (iii) which is mostly free of external parameters. Of course we would like to have some parameters to influence cluster properties like size.

Recall figure 4.6. First we use possibility 4.6(a). By means of figure 6.1 we describe our procedure. Note that ② is defined by definition 4.8 already. We define the combination step ① as a weighted sum:

$$\widetilde{x} := \sum_{i=1}^{k} \alpha_i x_i. \tag{6.1}$$

We specify the α_i as additional parameters for our approach. Interesting is ③, since we have to solve the induced cluster problem in this step. Inspired by the MCL process result, we calculate a minimum spanning tree (MST) of the undirected underlying skeleton of the input graph with respect to the edge weights $\sigma_{\widetilde{x}}$. We can calculate a tree, since the input graph is connected. Next we remove certain tree edges, namely edges which exceed a certain threshold value. Like with single or complete linkage it is quite difficult to choose a threshold value for this process. Undesired artifacts are one reason. For example compare the chaining effects of single linkage. Therefore we need a good suggestion for this threshold value. We would like to have a method which calculates such a suggestion with the help of $x_1, \ldots, x_k$, $\widetilde{x}$ and $\sigma_{\widetilde{x}}$. We

choose Δ depending on the average weight of all edges and all tree edge as well as the maximal weight of all edges and all tree edges. Some external parameters will be used for adaptations. In algorithm 2 we have summarised these parts of ③.

Algorithm 2: general description of ③

Input : input graph G, edge weight s, $x_1, \ldots, x_k$, $\widetilde{x}$, $\sigma_{\widetilde{x}}$ and some external parameters

Output: $\widetilde{\mathbb{P}}$

begin

- calculate a minimum spanning tree T of G with respect to $\sigma_{\widetilde{x}}$
- calculate the average a_T and the maximal weight m_T of all tree edges with respect to $\sigma_{\widetilde{x}}$
- calculate the average a_G and the maximal weight m_G of all edges with respect to $\sigma_{\widetilde{x}}$
- calculate threshold value Δ with a_T, m_T, a_G, m_G and the external parameters
- delete all edges with weight greater as Δ from T
- use T's connected components as partition components of $\widetilde{\mathbb{P}}$

end

Combining these steps we obtain the minimum spanning tree approach:

Approach 5 (minimum spanning tree (MST) approach)
We have as input parameters:

- an undirected or bidirected connected graph G and an edge weight function s measuring the degree of similarity
- parameter k as the number of considered eigenvectors
- weight parameters α_i for $i = 1, \ldots, k$ measuring the importance of the eigenvectors
- external parameters or an external method used to calculate the threshold value Δ in ③

For the procedure itself we obtain:

1. calculate $\mathcal{N}(G)$ with respect to s
2. calculate eigenvectors $x_1, \ldots, x_k$ to $\mu_{n-k-1}(G), \ldots, \mu_{n-1}(G)$
3. interpret $x_1, \ldots, x_k$ as cluster $\widetilde{\mathbb{P}}$, for example with algorithm 2

Return the calculated cluster $\widetilde{\mathbb{P}}$.

Before we analyse this approach with regards to invariance, runtime, space and usability we present our motivation respectively our intuition behind the chosen tools. Three items are very interesting: first the use of $\mathcal{N}$, second involving only eigenvectors to positive eigenvalues and finally the usage of a minimum spanning tree for the induced cluster problem.

We involved the normalised matrix because of several facts. Van Dongen also calculated his partitioning with this matrix. Other reasons are for example presented in [KVV00]. In this paper the matrix is used in a spectral partitioning approach. Indeed the authors showed some approximation guarantees and connections to the conductance measure. Often the Laplacian matrix is chosen for partitioning. We compared the eigen structure of the Laplacian and the normalised matrix and assumed that the normalised matrix is more suitable. This is due to the facts which are issued in corollary 3.42 and 3.43. Linear independent eigenvectors of the Laplacian matrix are orthogonal with respect to the euclidian scalar product in contrast to linear independent eigenvectors of the normalised matrix. These are orthogonal with respect to $\langle , \rangle_N$. The scalar product $\langle , \rangle_N$ is in general a nonuniform weighted one and this weighting directly depends on the input edge weight s. Therefore we assumed that this fact may influence the partitioning properties in a positive way.

We calculated only eigenvectors to positive eigenvalues due to partitioning properties. Recall proposition 3.34. The function π measures the quotient of total edge length to the total space used. Restricting π to nontrivial placements we have that x_2 is a global minimum and x_n is a global maximum. Therefore the entries of x_2 can be interpreted as clustering with minimal inter–cluster distance, but if we used x_n in the same way we would get a clustering with maximal inter–cluster distance. So x_n could be used for the dual cluster problem, which is quite related to the vertex–coloring problem. The situation for the normalised matrix is similar but reversed with respect

to ordering. For example connected bipartite graphs have $\mu_1 = -1$. Considering the associated eigenvector we would gain a placement where the two components are separated. This would be a quite bad cluster since no components contain any edges. So we have to decide which eigenvectors can be used for clustering and which one should not be used. We assumed that eigenvectors to positive eigenvalues are good candidates.

Why do we use spanning trees? We are not the first researchers which use spanning trees for cluster approaches. The basic model is ZAHN*'s Clustering Algorithm*, see for example [JD88, p. 120ff.]. In this reference paper a spanning tree with minimum or maximum weight is calculated first. Afterwards "bad" edges are eliminated. This is often done by something similar to single linkage, approach 1. One weak point is the threshold value for the final reduction. But we also assume that the input edge weight function does not need to contain very much cluster information, for example consider a constant weight function. In this situation every vertex belongs to its own component or all vertices are in the same component. These are very few possibilities. In our situation we assumed two issues which we actually cannot prove but seems to be intuitive. First, the eigenvectors contain more or better cluster information. This may be correct since spectral methods are often used in the area of partitioning and even some properties can be proved respectively there exists an approximation guarantee. Second, two vertices which ideally belong to the same component have similar entries with respect to the eigenvectors. This is again a consequence of proposition 3.34. The vector x_2 cannot be the optimum if highly connected subgraphs are spread over a big area. Since a minimum spanning tree is a structure with minimum weight but maximal reachability we use it for our experiments. Of course the selection of the threshold value is still the critical part. As mentioned before, we assumed that the eigenvectors' valences express the clustering and therefore we conjectured that a minimum spanning tree contains few edges with high weight. To calculate an appropriate threshold value we used the quotient of average edge weight and the maximum edge weight. We calculated this quotient for the whole graph and for the minimum spanning tree. This resolved in a first suggestion for the threshold value. The final value was determined by this value and external choices, which were given by the user. In chapter 7 we describe the exact procedure we used.

In the following section we analyse the minimum spanning tree approach, approach 5. We investigate the following topics: runtime, space and invariance.

We begin by stating some technical informations about minimum spanning trees, which will be necessary for the invariance theorem. Runtime and used space are going to be considered in section 6.3.

6.2 Invariance

Figure 6.2 gives a short sketch of the dependency of the lemmas, the corollary and the theorems which we used to prove the invariance of approach 5.

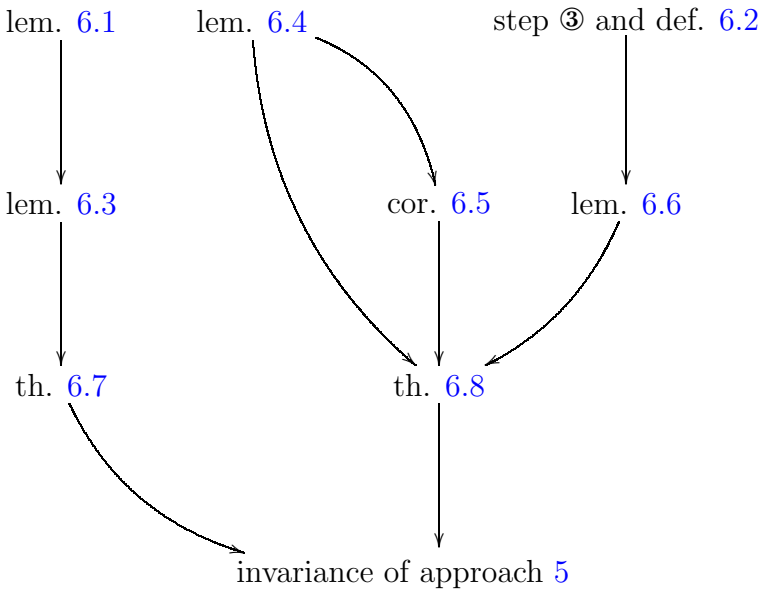


Figure 6.2: Graphical "guide" to the invariance proof of approach 5

Lemma 6.1
Every tree on n vertices has $n-1$ edges.

Proof:
This can be proved by induction on n. □

Definition 6.2
Let G be a graph with an edge weight function σ. By $\mathbb{M}(G,\sigma)$ we denote the set of all minimum spanning trees of G with respect to σ. Let $T \in \mathbb{M}(G,\sigma)$ and Δ be a threshold value. Without loss of generality we assume no edge

has exactly the weight of Δ. A path of tree edges such that every edge has a weight smaller than Δ is called a *component connecting path*. A tree edge which has a weight greater than Δ is called a Δ*–cut edge*.

Lemma 6.3
Let G be a graph with an edge weight function σ. The average weight function for subgraphs $\operatorname{avg}(\cdot)$ *and the maximal weight function for subgraphs* $\max(\cdot)$ *are constant on the set* $\mathbb{M}(G,\sigma)$.

Proof:
Every spanning tree has the same number of edges (see lemma 6.1) and every minimum spanning tree has the same weight. So the quotient is the same for every minimum spanning tree.

The second part, that every minimum spanning tree has the same maximal weight, will be proved by contradiction: let $T_1, T_2 \in \mathbb{M}(G,\sigma)$ be a counter–example and without loss of generality we assume $\max(T_1) > \max(T_2)$. We remove all edges from T_1 which have weight $\max(T_1)$ and gain a forest with components $H_1, \ldots, H_r$. We have at least two components, because T_1 has at least one edge with weight $\max(T_1)$. Since T_2 is a spanning tree, there exist $r-1$ edges $e_1, \ldots, e_{r-1}$ such that every one of them connects a different pair of components in the forest. Let T'' be the graph which is induced by the union of $H_1, \ldots, H_r$ and all edges $e_1, \ldots, e_{r-1}$. Then T'' is a spanning tree and its weight is:

$$\begin{aligned}\sigma(T'') &= \sum_{i=1}^{r}\sum_{e\in H_i}\sigma(e) + \sum_{i=1}^{r-1}\sigma(e_i)\\ &= \sigma(T_1) - (r-1)\max(T_1) + \sum_{i=1}^{r-1}\sigma(e_i)\end{aligned}$$

For every edge e of T_2 we have: $\sigma(e) \le \max(T_2) < \max(T_1)$. So we obtain $\sigma(T_1) > \sigma(T'')$ which is a contradiction to T_1 being a minimum spanning tree. □

Lemma 6.4
Let G be a graph and σ an edge weight function. Let $T \in \mathbb{M}(G,\sigma)$. For every non tree edge e' there exists another minimum spanning tree $T' \in \mathbb{M}(G,\sigma)$ containing e' iff the path p in T which connects source(e') *and* target(e') *satisfies*

$$\sigma(e') = \max\{\sigma(e) : \ p \text{ contains } e\}. \tag{6.2}$$

Proof:
The edge e together with p forms an elementary cycle. If equation (6.2) is fulfilled there exists an edge e in p such that $\sigma(e) = \sigma(e')$. So denote by T' the tree which arises from T by replacing e with e'. Then T' is spanning and has also minimum weight since e and e' has the same weight.

If equation (6.2) does not hold we have $\sigma(e') > \max\{\sigma(e) : p \text{ contains } e\}$, since otherwise we could replace any edge in p with maximal weight by e' and would gain a spanning tree with smaller weight than T which would be a contradiction. So assume there exists a minimum spanning tree $T_{e'}$ which contains e'. Then p contains at least one non tree edge with respect to $T_{e'}$, since otherwise $T_{e'}$ would contain a circle. Let e be such an edge. Then we gain a spanning tree by replacing e' by e in $T_{e'}$. This tree has smaller weight than $T_{e'}$ since $\sigma(e') > \sigma(e)$. Thus $T_{e'}$ can not be a minimum spanning tree. □

Corollary 6.5
Let G be a graph with an edge weight function σ. Let $T, T' \in \mathbb{M}(G, \sigma)$. Then there exists a sequence

$$T = T_0, \ldots, T_{k+1} = T'$$

such that every $T_j \in \mathbb{M}(G, \sigma)$ and every two directly subsequent trees differ in exactly one edge. Therefore T and T' have $n - 2$ common edges.

We omit this proof, since it is rather technical.

Next we consider the connection between the minimum spanning tree and the induced partitions.

Lemma 6.6
Let G be a graph with an edge weight function σ. Let $T \in \mathbb{M}(G, \sigma)$. Let e be a non tree edge and p the path in T which connects source(e) *and* target(e). *Let $e_1, \ldots, e_r$ be edges of p which have maximal weight with respect to σ. If $\sigma(e_j) = \sigma(e)$ and we exchange any one e_j by e in T then the partitions induced by T, see step ③ on page 80, do not change.*

Proof:
Let j be an arbitrary index between 1 and r. Let p be $(v_0, \ldots, v_k)$ and without loss of generality $v_0 = \text{source}(e)$ and $v_k = \text{target}(e)$. Also let s_i and t_i be indices such that

$$v_{s_i} = \text{source}(e_i) \quad \text{and} \quad v_{t_i} = \text{target}(e_i) \qquad \text{for } i = 1, \ldots, r.$$

Without loss of generality we assume

$$0 \leq s_1 < t_1 \leq s_2 < t_2 \leq \ldots \leq s_r < t_r \leq v_k$$

Figure 6.3 shows a sketch with $r = 2$. We set m as the weight of $\sigma(e_j)$.

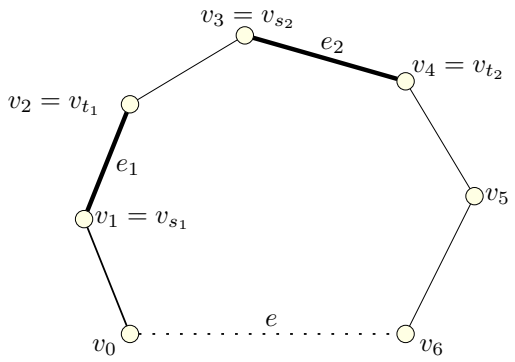


Figure 6.3: Sketch of p and e with $r = 2$

If we replace e_j by e in T we obtain the path $p' = (v_{t_j}, \ldots, v_k, v_0, \ldots, v_{s_j})$. If $m < \Delta$ then p is a component connecting path and so $v_0, \ldots, v_k$ belong to one component. The path p' is also a component connecting path since the maximal weight of the edges is less than Δ. Therefore the partition has not changed. Otherwise if $m > \Delta$, then $e_1, \ldots, e_r$ are split edges. So removing $e_1, \ldots, e_r$ in p we gain a partition $\mathbb{P} = (V_1, \ldots, V_{r+1})$ defined by:

$$\begin{aligned} V_1 &:= \{v_\ell : 0 \leq \ell \leq s_1\} \\ V_{i+1} &:= \{v_\ell : t_i \leq \ell \leq s_{i+1}\} \qquad \text{for } i = 1, \ldots, r-1 \\ V_{r+1} &:= \{v_\ell : t_r \leq \ell \leq k\} \end{aligned}$$

This may not to be the final partition induced by T and restricted to p, since p may contain other edges with weight greater than Δ. Removing the edges with maximal weight in p', namely $e_1, \ldots, e_{j-1}, e_{j+1}, \ldots, e_r, e$, also creates the partition $\mathbb{P}$. So the partition introduced by T does not change if we replace e_j by e. Since j was arbitrarily chosen we proved the lemma. □

Theorem 6.7
The partitions created by approach 5 are independent of the chosen minimum spanning tree.

Proof:
Let G be a graph on n vertices and with an edge weight function σ. We consider $T, T' \in \mathbb{M}(G, \sigma)$. We obtain a sequence $T_0, \ldots, T_{k+1} \in \mathbb{M}(G, \sigma)$ such that $T_0 = T$, $T_{k+1} = T'$ and T_i and T_{i+1} having $n-2$ common edges for $i = 0, \ldots, k$ by corollary 6.5. Thus it is sufficient to show that T_i and T_{i+1} induce the same partitions. So without loss of generality we assume that T and T' have $n-2$ common edges. Let e' be a tree edge of T' and a non tree edge in T. Since $T' \in \mathbb{M}(G, \sigma)$ we can apply lemma 6.4 to T and e' and obtain that the path p which connects source(e') with target(e') in T has an edge e which has maximal weight with respect to σ in p and $\sigma(e) = \sigma(e')$. By lemma 6.6 we obtain that T and T' induce the same partition. □

Theorem 6.8
The threshold value Δ in step ③ on page 80 does not depend on the chosen minimum spanning tree.

Proof:
Only $\operatorname{avg}(T)$ and $\max(T)$ depend on the chosen minimum spanning tree. With lemma 6.3 we know that $\operatorname{avg}(\cdot)$ and $\max(\cdot)$ are constant on $\mathbb{M}(G, \cdot)$, so this completes the proof. □

Thus theorem 6.7 and 6.8 imply that approach 5 always calculates the same cluster for fixed input parameters and fixed eigenvectors. Due to errors in finite arithmetic or high dimensional eigenspaces it is possible that the MST approach calculates different clusters.

6.3 Runtime and space

Recall that we normally have the following input: A undirected or bidirected connected graph $G = (V, E)$ and an edge weight function s.

In appendix B and C we summarised the technical tools for an analysis of spectral part of approach 5. We used a kind of power–method with a constant number of vectors. In the `adjust`–step we ensured convergence toward different eigenvalues respectively linear independent eigenvectors with an orthogonalization process similar to lemma 3.17. Thus the runtime of these steps together is linear in the input size and the iteration depth. We also need only linear additional space.

For the minimum spanning tree part we need a little bit more than linear time. Calculating a minimum spanning tree can be done in $\mathcal{O}(|E| \log |E|)$ time and $\mathcal{O}(|E| + |V|)$ space with the algorithm of KRUSKAL. For details see [Jun99, p. 112ff.]. Calculating $\text{avg}(T)$, $\text{avg}(G)$, $\max(T)$ and $\max(G)$ can be done in $\mathcal{O}(|E|)$ time with constant space. The combining step which creates the threshold value Δ depends on the external method respectively the external parameters; we assume that it needs at most linear time and space[1]. Calculating the partition can be done in $\mathcal{O}(|V| + |E|)$ time with $\mathcal{O}(|V|)$ space. Depth First Search, Breadth First Search or Union Find are possibilities to implement this part.

6.4 Usability

We started our research particularly with regard transforming the MCL process to use only linear memory and to work directly with the graph structure. Currently we do not know whether the structure of our approach can be compared or if any guarantees can be proven. Besides we think that the MST approach is a very flexible method which can be adapted to use additional information of the input. Especially clustering with interactive elements benefits from its simple structure and options. Our technique may also be utilised for online clustering. Here we refer to both topics: first online input, which means the input graph and edge weight function may change during the calculation and second online structure, which refers to models like WWW. Note that online structure can also be used as offline input, see subsection 5.4.1. The biggest profit of online clustering is the linear space usage, so only constant space for each vertex and each edge is needed. Another advantage is the use of the graph structure in such a direct way. During most computation steps only the local composition is accessed, i.e. a vertex and its neighbours. The calculation of the minimum spanning tree may require a more global view of the graph depending on the implementation. In subsection 7.2.1 we consider such an online clustering relating a generic computer network.

[1] Note that especially linear space would be expensive!

6.5 Variants of the minimum spanning tree approach

Next we introduce some variants of approach 5. They have the same background but use the other possibilities drafted in figure 4.6. For each method we briefly state its proceeding and some remarks concerning advantages and disadvantages.

6.5.1 Combining the potentials' difference

Here we consider the combination of the potentials' differences, see figure 4.6(b). The main difference to the original MST approach is that step ① and ② are interchanged. Thus $\widetilde{\sigma}$ is calculated by:

$$\forall\, e \in E\colon\ \widetilde{\sigma}\,(e) := \sum_{i=1}^{k} \alpha_i \cdot \sigma_{x_i}\,(e) \tag{6.3}$$

Equation (6.3) is the analogy to equation (6.1). Of course the combining step may be realised in a different way. We considered only this possibility.

This approach, we called it *edge MST approach*, has a fundamental benefit in contrast to the MST approach: Partitions created with the MST approach are strongly influenced by the signs of the eigenvectors' entities. For certain parts of the graph the signs coincide and thus the eigenvectors enforce the global potential. In the other parts the signs differ and so the eigenvectors enfeeble each other. One result is that isomorphic subgraphs and symmetries are not handled in the same way. In subsection 8.2.1 we present such a behaviour in the context of some path and grid examples. The edge MST approach is almost independent of such problems. This is due to the fact that we accumulate the potentials' difference and not the potentials itself.

The edge MST approach can also be seen as incremental approach. First we calculate some eigenvectors and the induced partition. If this cluster is sufficient in a certain sense then it will be returned as the final result. Otherwise additional eigenvectors are calculated and the partition induced by the previous eigenvectors and the new eigenvectors is considered. Because of equation (6.3) we can calculate the new edge weight function $\widetilde{\sigma}_{\text{new}}$ by the help of $\widetilde{\sigma}_{\text{old}}$ and the new eigenvectors. This may be done until the induced partition is sufficient or a maximal number of eigenvectors are calculated.

6.5.2 Combining auxiliary clusters

Another possibility is the combination of auxiliary clusters, drafted in figure 4.6(c). This version of the MST approach calculates first auxiliary partitions. Each partition can be calculated by either the original MST approach, the edge MST approach or derivations. Then these are combined to form the final cluster. There are quite a lot possibilities ranging from starting with the eigenvector to the auxiliary partitions.

In contrast to the MST and the edge MST approach many induced cluster problems are solved, described in section 6.1. First, we have to solve it for each auxiliary clusters and second, in the combining step. This leads to a great degree of freedom but also to many trap doors. To keep this situation manageable additional input parameters are inserted, but these parameters may create undesired artifacts.

We consider only a quite simple instance using this method. Due to readability we introduce another notation:

Definition 6.9
Let $\mathbb{P} = (X_1, \ldots, X_m)$ be the partition. Then we define $\mathcal{C}_{\mathbb{P}}(u, v)$ as:

$$\mathcal{C}_{\mathbb{P}}(u, v) := \begin{cases} 1 & \text{if an index } i \text{ exists such that } u, v \in X_i \\ 0 & \text{otherwise} \end{cases}.$$

Note that $\mathcal{C}_{\mathbb{P}}(\cdot, \cdot)$ can be interpreted as equivalence relation. Then the equivalence classes are $X_1, \ldots, X_m$.

Our instance of possibility 4.6(c) is described below in approach 6. It uses the edge MST approach to calculate the auxiliary partitions. These are combined by a weighted sum.

Approach 6 (partition minimum spanning tree approach)
We have as input parameters:

- an undirected or bidirected connected graph G and an edge weight function s measuring the degree of similarity
- parameter k as number of considered eigenvectors
- weight parameters α_i for $i = 1, \ldots, k$ measuring the importance of the eigenvectors respectively the importance of the associated minimum spanning tree
- weight parameters Δ_i for $i = 1, \ldots, k$ measuring the threshold level for the i–th minimum spanning tree
- weight parameter $\Delta \in [0, 1]$ measuring the global threshold level

For the procedure itself we obtain:

1. calculate $\mathcal{N}(G)$ with respect to s
2. calculate eigenvectors $x_1, \ldots, x_k$ to $\mu_{n-k-1}(G), \ldots, \mu_{n-1}(G)$
3. calculate σ_{x_i} and a minimum spanning tree $T_i \in \mathbb{M}\left(G, \sigma_{x_i}\right)$.
4. let $\mathbb{P}_i$ be the partition induced by the minimum spanning tree T_i with respect to Δ_i
5. delete all edges $e \in E$ such that:
$$\frac{\sum_{i=1}^{k} \alpha_i \cdot \mathcal{C}_{\mathbb{P}_i}\left(\text{source}(e), \text{target}(e)\right)}{\sum_{i=1}^{k} \alpha_i} \leq \Delta$$
6. the undirected connected components of G form the components of the final partition $\widetilde{\mathbb{P}}$

The parameter Δ influences the cluster granularity. For the values 0 and 1 we obtain the extremal clusters $\mathbb{P}_0 = (V)$ and $\mathbb{P}_1 = (\{v_1\}, \ldots, \{v_n\})$ where the vertex set is $\{v_1, \ldots, v_n\}$.

Chapter 7

Implementation aspects and examples of the minimum spanning tree approach

For some experimental studies we implemented all three variants of the MST approach. First a detailed description is given. In the following parts several examples are presented.

7.1 Implementation

Here some aspects of the implementation are presented. We only describe the technical environment and our choices of different parameters which induce several degrees of freedom. We will not display source code or survey elementary data types and the consequences of their use.

7.1.1 Technical aspects

The implementation was written in the programming language `C++` version 2.95.3. Some data structures and graph algorithms were obtained from `LEDA`[1] version 4.3. Implementation and testing was done on a `Linux` platform.

[1] Library for efficient data structures and algorithms

7.1.2 Fixation of several degrees of freedom

Like VAN DONGEN, we added a self–loop with weight 1 to each vertex.

The combining step and the threshold parameter Δ are the only choices in the MST approach (approach 5) and the edge MST approach. Like suggested previously we used a weighted sum for the combining step. We had two options available for the threshold parameter Δ. One simply used the average weight of all tree edges. The other possibility was a bit more complex: The closed interval $[m, M]$ where m is the minimum weight of all tree edges and the maximum M was used as range. Again the average weight $\operatorname{avg}(\cdot)$ of all tree edges was a kind of base point for the calculation. Then the final threshold value was calculated by:

$$\Delta = \operatorname{avg}(T) + \alpha \cdot \frac{\operatorname{avg}(G) - \operatorname{avg}(T)}{(\max(G) - \max(T)) + 1}, \tag{7.1}$$

where G denotes the input graph, T the minimum spanning tree and α an additional input parameter. This option adjusts the average value to the current situation. Note that the addition of 1 to the denominator is necessary since $\max(G) = \max(T)$ is possible. Note that the first threshold value option is included in the second one by the choice of $\alpha = 0$. Although equation (7.1) may not be the best adjustment we have not investigated further concerning this option.

In the partition MST approach all threshold parameters Δ_i are chosen in the way: the average weight of all tree edges of the minimum spanning tree T_i or according to equation (7.1). The combining step was again a weighted sum. We use the formula of step 5 in approach 6.

7.1.3 Calculation of eigenvectors

We used a modified power iteration method to calculate the eigenvectors. The concept is described in appendix B. The `adjust`–step rescaled the current vectors and ensured orthogonality between them. Since we used the normalised matrix, orthogonality induced by $\langle , \rangle_N$ was used. To implement this method we needed a matrix–vector–multiplication, rescaling and a orthogonality procedure. In appendix C these methods are given in pseudo code.

7.2 Examples

This section covers two parts. In the first part we present a real–world situation where cluster approaches can be used to solve problems. The second part considers graphs and the calculated clusters. We show some graphs which illustrate the differences between our approaches.

7.2.1 Computer network

Recall the situation described in subsection 2.3.1. We consider this model again. Now we distinguish between different kinds of terminals according to their tasks: Routers are terminals which handle communication between subnets. They realise the "global" view for the users. Servers take care of storage jobs and ensure that users can share information. All other terminals are called clients. Routers and servers usually work transparently on the net, this means invisible to users. Most topics concerning security, performance and reliability depend on the network structure. This structure is often predefined due to external factors like administrational ones. The kinds of terminals also have different requirements, for example the hardware for client is cheaper than for servers or routers. So ensuring the efficiency is an important point. The situation is further complicated because dominating factors rapidly change. Figure 7.1 displays a network. Rhombuses correspond to routers, squares represent servers and clients are drawn as circles. The bold edges symbols great connection capacity, the dashed edges medium and thin edges low capacity.

The sketch is drawn hierarchical, this means that the position of the terminal represents its importance or something comparable. Observe for example the subnetwork connected to the middle router. There are four servers present. A central one is connected to the remaining servers. Only two of eight clients have a direct connection to this server. It is quite probable that this station is the main server or a backup device for this part of the network. In both cases it is an important terminal.

Problem settings

Many problems can be solved with cluster techniques in this context. Especially modifications which involve adding or removing terminals or direct

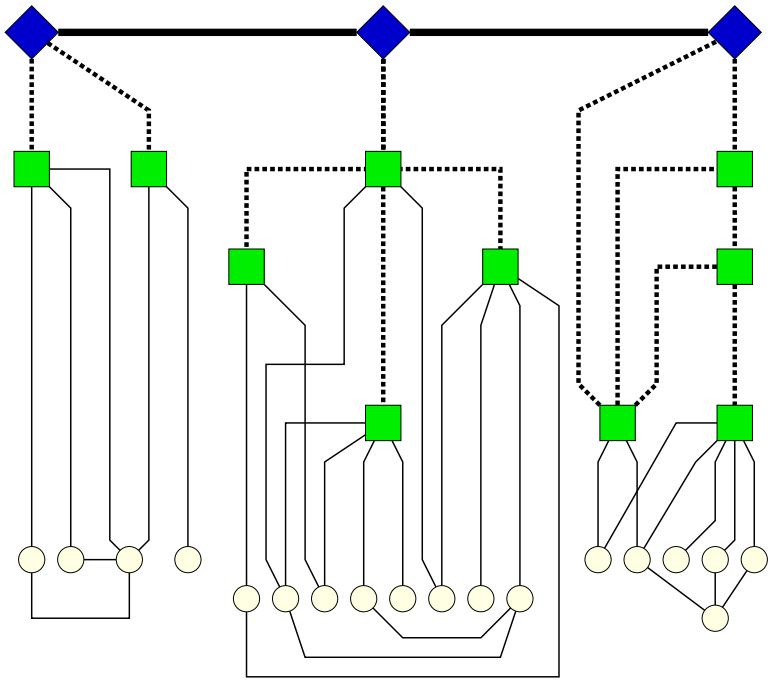


Figure 7.1: An example of a network

links. These can be modeled as cluster problems in a very natural way. But we are more interested in performance tests and reconstruction of subnetworks.

Performance is important to ensure fast access times and a high reliability. It depends very much on network traffic and work balance, therefore an online analysis is often desired. Reconstruction is considered if the general performance is low.

Possible solutions

We assume that the degree of similarity is principally defined by the workload of connections. Therefore we think that the MST approach is helpful in this context. First off all our approach may run directly in the network. For example as an independent program traveling through the whole network and collect the information. Since the edge weight function changes during the process there is no context for eigenvectors, although we can use the same technique. The phase where the eigenvectors $x_1, \ldots, x_k$ were calculated needs to be adjusted a little. These vectors $x_1, \ldots, x_k$ may lack cluster properties, because they no longer are eigenvectors and yet we do not know any reason for their partitioning properties. Thus the solution created by this kind of analysis is generally only a rough estimate but it can be helpful for further planning.

Also the MST approach can be used in the reconstruction part of the planning phase: The online situation can be simulated offline to evaluate varying of possibilities. Under these circumstances our approach truly shines. Interactive elements can be added to make the process more transparent and flexible.

7.2.2 Random geometric graphs

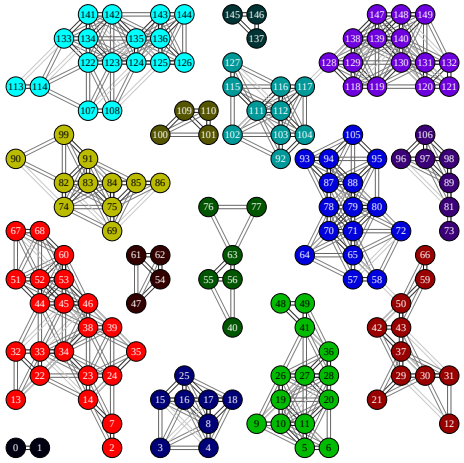
In view of comparison to the MCL process we consider the graph G_4 which is displayed in figure 5.2 on page 69. We interprete G_4 as bidirected graph. For the MST approach we used $k = 3$ and the threshold value is in general the average weight of the MST edges. To compare the various MST approaches the same eigenvectors were used. Figures 7.2 and 7.3 display the results of the MCL process, MST, edge MST and partition MST approach. All drawings respect the same principles:

- The input graph is bidirected but the arrow heads which indicate the direction are omitted.
- All vertices of a component are drawn with the same color.
- Vertices which are a component on their own are drawn with a larger border than the other vertices.
- Inner–cluster edges are grey.
- Inter–cluster edges are omitted except for theses which connect a component of size one with another one and have minimal cost with respect to all outgoing edges; these connecting edges are drawn dashed.
- For MST approaches bold inner–cluster edges represent MST edges.

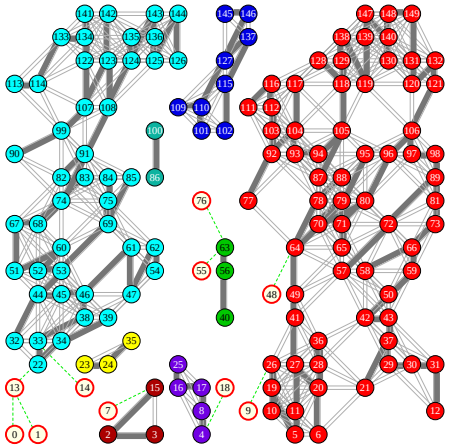
Some simple properties of these clusters are listed in the table displayed in figure 7.4. An asterisk (*) indicates that an additional post–processing phase was excecuted. During this phase all cluster components with size one are merged to the nearest component. We used nearest in the sense that those outgoing edges were chosen to have minimal weight with respect to the potentials' difference. So the minimal size of a component is two.

Next we consider the edge MST approach and threshold value option in detail. Here we use the second option for the threshold value calculation, see equation (7.1). All other parameters were kept ($k = 2$ and $\alpha_1 = \alpha_2 = \alpha_3 = 1$). The post–processing is again used to eliminate components of size one. We present the final clusters for different values of α. Also we kept the same principles for drawings as above. During the MST process we mark certain edges with very high weight with respect to the potentials' difference as inter–cluster edges. These edges are then invisible to remaining steps. The post–processing phase merges certain clusters and so inter–cluster edges become inner–cluster edges. Due to this fact it may be possible that not all inner–cluster edges are drawn. Nevertheless they are counted in the covered weight routine. Such an example is the lower left corner. Figures 7.5(a), 7.5(b), 7.5(c), 7.5(d) and 7.5(d) display the various clusters. The table shown in figure 7.6(b) summarises some simple data about these clusters.

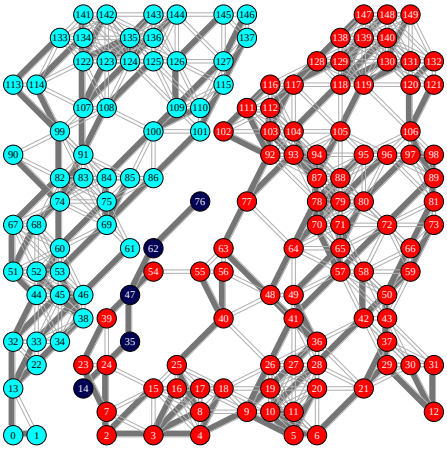
More examples can be found in section 8.2. There are "problematic" structures presented. They are sourced out to this section since they helped us



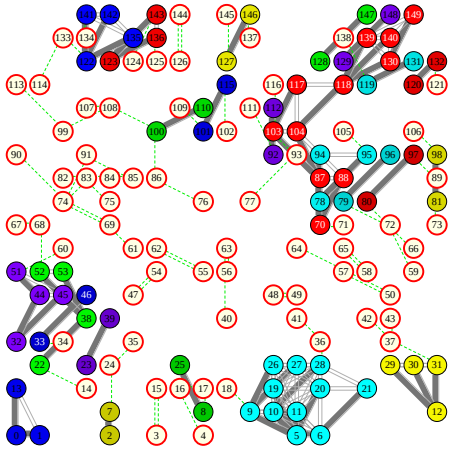
(a) MCL process with $e = r = 2$



(b) MST approach with $\alpha_1 = \alpha_2 = \alpha_3 = 1$



(c) partition MST approach with $\alpha_1 = \alpha_2 = \alpha_3 = 1, \Delta = 0.25$



(d) partition MST approach with $\alpha_1 = \alpha_2 = \alpha_3 = 1, \Delta = 0.5$

Figure 7.2: Clusters created by different approaches

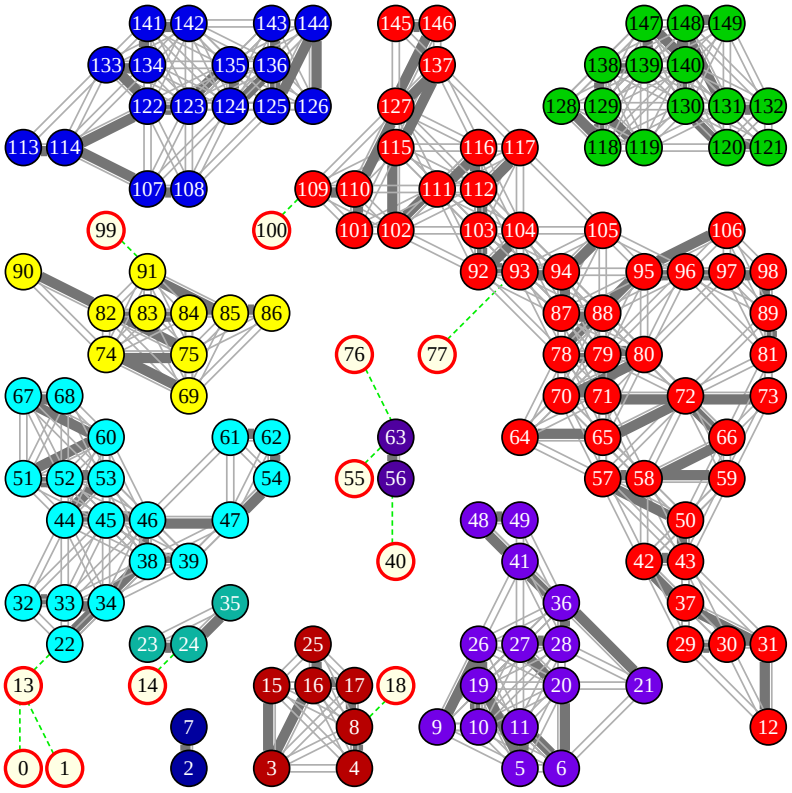
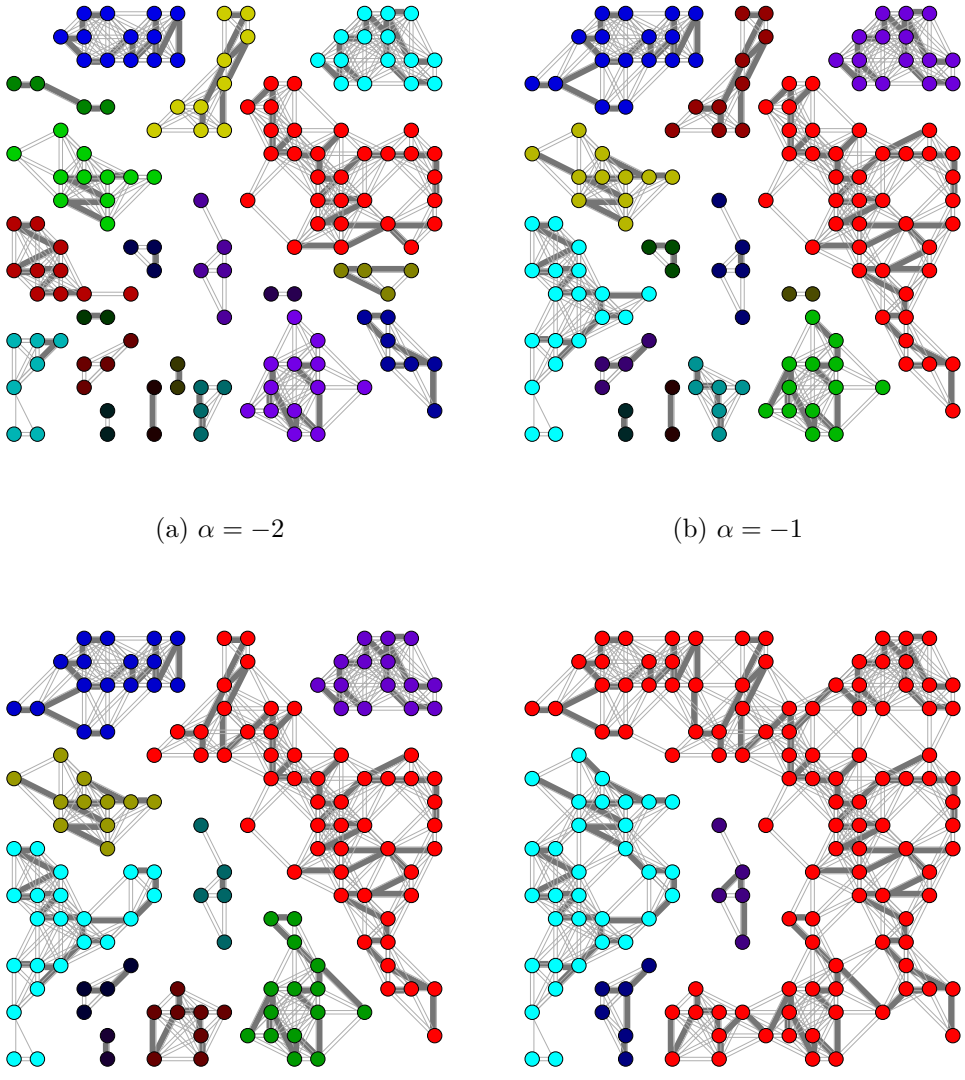


Figure 7.3: A clustering for G_4 with the edge MST approach using $\alpha_1 = \alpha_2 = \alpha_3 = 1$

approach	figure	size of partition	min. size of comp.	max. size of comp.	avg. size of comp.	covered weight
MCL process	7.2(a)	15	2	22	10.0	81.4%
MST approach	7.2(b)	16	1	69	9.3	86.8%
MST approach*	—	7	2	72	21.0	90.9%
edge MST approach	7.3	21	1	49	7.1	83.1%
edge MST appraoch*	—	10	2	51	15.0	87.7%
partition MST approach	7.2(c)	3	5	90	50.0	91.9%
partition MST approach	7.2(d)	91	1	11	1.5	15.8%
partition MST approach*	—	33	2	15	4.5	40.1%

Figure 7.4: Information of the various clusters

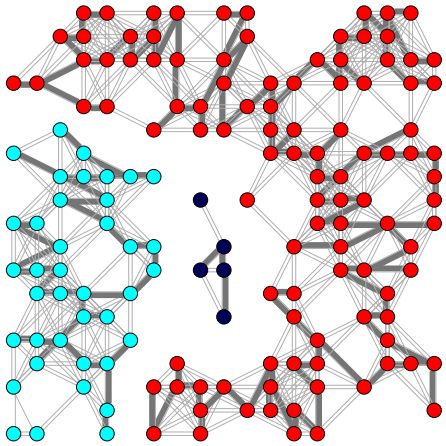
to understand certain behaviour and aided us during the search of a cluster algorithm.



(a) $\alpha = -2$ (b) $\alpha = -1$

(c) $\alpha = 0$ (d) $\alpha = 1$

Figure 7.5: Results of the edge MST approach I/II



(a) $\alpha = 2$

α	Δ	size of partition	min. size of comp.	max. size of comp.	avg. size of comp.	covered weight
-2	0.00667	20	2	30	7.5	74.6%
-1	0.00927	14	2	41	10.7	83.2%
± 0	0.0118	10	2	51	15.0	87.7%
$+1$	0.0144	4	5	106	37.5	94.6%
$+2$	0.0170	3	5	106	50.0	96.6%

(b) some brief information of the clusters

Figure 7.6: Results of the edge MST approach II/II and some brief informations about the clusters

Chapter 8

Counter–examples

In this chapter we present some counter–examples. These are graphs or graph families which have an "obvious" clustering but some approaches calculate different clusters. We will also see that these counter–examples quite depend on the context since these examples often need a certain trade–off between different properties.

8.1 Laplace–Bisection

Laplace–Bisection, approach 3, is a very fundamental method in cluster analysis and graph partitioning. The following counter–example shows that the simple transformation step does not necessary lead to a small balanced cut.

Definition 8.1
Let n be a natural number. The undirected graph $G^{(1)}(n)$ is defined by:

- a vertex set V with $4n$ elements

  $$a_1, \ldots, a_n, b_1, \ldots, b_{2n}, c_1, \ldots, c_n$$

  which we divide into three subsets A, B and C

  $$\begin{aligned} A &:= \{a_i : 1 \leq i \leq n\} \\ B &:= \{b_i : 1 \leq i \leq 2n\} \\ C &:= \{c_i : 1 \leq i \leq n\} \end{aligned}$$

- and an edge set E which is defined by:

$$\begin{aligned} E \;:=\; & \Big\{ \{a_i, a_j\} \,:\, 1 \le i < j \le n \Big\} \cup \\ & \Big\{ \{c_i, c_j\} \,:\, 1 \le i < j \le n \Big\} \cup \\ & \Big\{ \{b_k, b_\ell\} \,:\, 1 \le k < \ell \le 2n \Big\} \cup \\ & \Big\{ \{a_1, b_1\},\ \{b_{2n}, c_1\} \Big\} \end{aligned}$$

The graph family $\mathcal{G}^{(1)}$ is the set of all graphs $G^{(1)}(n)$.

The graph $G^{(1)}(n)$ can be considered as the union of three graphs H_A, H_B and H_C. The graphs H_A and H_C are complete on n vertices while H_B is a complete graph on $2n$ vertices. Also H_B is connected by an edge with H_A and H_C. The endpoints of these two edges are different in H_B. Figure 8.1 shows $G^{(1)}(8)$.

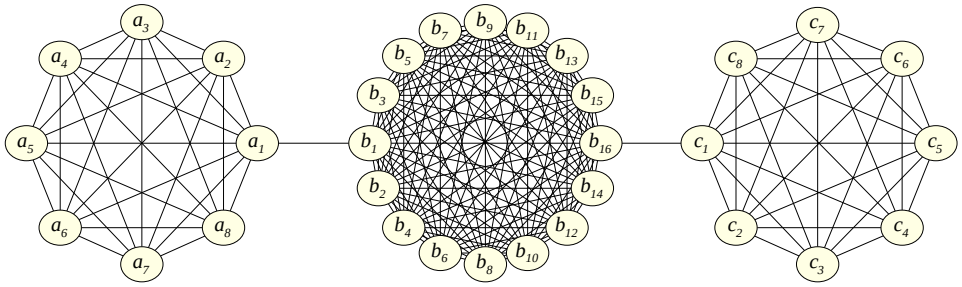


Figure 8.1: $G^{(1)}(8)$

Lemma 8.2
Every graph in $\mathcal{G}^{(1)}$ has a balanced cut of weight two and one of weight n^2.

Proof:
Consider $G = G^{(1)}(n) \in \mathcal{G}^{(1)}$. We define two cuts: one $(S, \overline{S})$ by

$$S := \{b_i \,:\, 1 \le i \le 2n\} = B$$

and another $(T, \overline{T})$ by

$$T := A \cup \{b_i : 1 \leq i \leq n\}$$

Both cuts are balanced since G has $4n$ vertices and S as well as T has $2n$ elements. For the first cut $\{a_1, b_1\}$ and $\{b_{2n}, c_1\}$ are the only cut edges, so the weight of $(S, \overline{S})$ is two. For the other cut, the set of all cut edges is

$$\mathbb{I}_T := \mathbb{I}_{(T,\overline{T})} = \left\{ \{b_i, b_j\} : 1 \leq i \leq n,\ n+1 \leq j \leq 2n \right\}.$$

Since $\mathbb{I}_T$ has n^2 edges, the weight of $(T, \overline{T})$ equals n^2. □

Lemma 8.3
Let n be a natural number. Then the following inequations hold:

$$0 < \lambda_2(G^{(1)}(n)) \leq 1.$$

Proof:
Fix a natural number n. We use λ_2 as short term for $\lambda_2(G^{(1)}(n))$. Since $G^{(1)}(n)$ has one connected component, we have $\lambda_2 > 0$. Equation (3.14) of theorem 3.35 implies that

$$\lambda_2 \leq \kappa(G^{(1)}(n)).$$

Every path which connects a vertex in A with a vertex in C uses the edges $\{a_1, b_1\}$ and $\{b_{2n}, c_1\}$. So $\kappa(G^{(1)}(n)) = 1$. □

The bound of lemma 8.3 can be improved to $\lambda_2(G^{(1)}(n)) \in]0, 1[$.

Lemma 8.4
Let n be a natural number and (λ_2, x_2) be an eigenpair of $\mathcal{L}\left(G^{(1)}(n)\right)$, such that $\lambda_2 = \lambda_2(G^{(1)}(n))$. Let u, w be two vertices such that $\mathrm{N}(u) = \mathrm{N}(w)$, then $[x_2]_u = [x_2]_w$.

Proof:
Fix a natural number n. We use $x_u := [x_2]_u$ and $x_w := [x_2]_w$ as short terms. The graph $G^{(1)}(n)$ is undirected so condition $\mathrm{N}(u) = \mathrm{N}(w)$ extends to $\mathrm{N}^{\leftarrow}(u) = \mathrm{N}^{\leftarrow}(w)$ and $\mathrm{N}^{\rightarrow}(u) = \mathrm{N}^{\rightarrow}(w)$. Since (λ_2, x_2) is an eigenpair

of $\mathcal{L}\left(G^{(1)}(n)\right)$ we obtain the following equations:

$$\begin{aligned}
& \begin{cases} [\mathcal{L}\left(G^{(1)}(n)\right) x_2]_u = \lambda_2 x_u & \text{and} \\ [\mathcal{L}\left(G^{(1)}(n)\right) x_2]_w = \lambda_2 x_w \end{cases} \\
\iff \quad & \begin{cases} \lambda_2 x_u = \deg(u) x_u - \sum_{v \in \mathrm{N}^{\rightarrow}(u)} [x_2]_v & \text{and} \\ \lambda_2 x_w = \deg(w) x_w - \sum_{v \in \mathrm{N}^{\rightarrow}(w)} [x_2]_v \end{cases} \\
\implies \quad & (\lambda_2 - \deg(u)) \cdot (x_u - x_w) = 0
\end{aligned}$$

We obtain $\lambda_2 - \deg(u) \neq 0$ since $\deg(u) = 0$ or $\deg(u) \geq 1$ and $\lambda_2 \in]0, 1[$. Thus we have $x_u = x_w$. □

Proposition 8.5
Let $n > 2$ be a natural number, $\mathcal{L} = \mathcal{L}\left(G^{(1)}(n)\right)$ the Laplacian matrix of $G^{(1)}(n)$ and $\lambda_2 = \lambda_2(G^{(1)}(n))$. If the following equation holds

$$\det \underbrace{\begin{pmatrix} 1-\lambda_2 & -1 & 0 \\ 1-n & n-\lambda_2 & -1 \\ 0 & -1 & 2n+1-\lambda_2 \end{pmatrix}}_{=:M} = 0 \tag{8.1}$$

then there exist positive real numbers γ, α and β such that x defined by:

$$[x]_w := \begin{cases} \gamma & \text{if } w \in A \setminus \{a_1\} \\ -\gamma & \text{if } w \in C \setminus \{c_1\} \\ 0 & \text{if } w \in B \setminus \{b_1, b_{2n}\} \\ \alpha & \text{if } w = a_1 \\ -\alpha & \text{if } w = c_1 \\ \beta & \text{if } w = b_1 \\ -\beta & \text{if } w = b_{2n} \end{cases}$$

is an eigenvector to the eigenvalue λ_2 of $\mathcal{L}$.

Proof:
First note that lemma 8.4 implies a certain structure of the eigenvector x on the subsets $A \setminus \{a_1\}$, $B \setminus \{b_1, b_{2n}\}$ and $C \setminus \{c_1\}$. Our definition of x is consistent with these structures. Next we calculate $\mathcal{L}x$ and show then that the equation

$$\mathcal{L} \cdot x = \lambda_2 x \tag{8.2}$$

holds. To determine $\mathcal{L}x$, we distinguish the following cases with respect to the vertices of $G^{(1)}(n)$:

- $w \in A \setminus \{a_1\}$, $w = a_j$

$$\begin{aligned}
[\mathcal{L}x]_w &= \deg(w) \cdot [x]_w - \sum_{\substack{i=1 \\ i \neq j}}^{n} [x]_{a_i} = (n-1) \cdot [x]_{a_j} - \sum_{\substack{i=2 \\ i \neq j}}^{n} [x]_{a_i} - [x]_{a_1} \\
&= (n-1) \cdot \gamma - (n-2) \cdot \gamma - 1 \cdot \alpha \\
&= \gamma - \alpha
\end{aligned}$$

- $w \in C \setminus \{c_1\}$, $w = c_j$

$$\begin{aligned}
[\mathcal{L}x]_w &= \deg(w) \cdot [x]_w - \sum_{\substack{i=1 \\ i \neq j}}^{n} [x]_{c_i} = (n-1) \cdot [x]_{c_j} - \sum_{\substack{i=2 \\ i \neq j}}^{n} [x]_{c_i} - [x]_{c_1} \\
&= (n-1) \cdot (-\gamma) - (n-2) \cdot (-\gamma) - 1 \cdot (-\alpha) \\
&= -(\gamma - \alpha)
\end{aligned}$$

- $w \in B \setminus \{b_1, b_{2n}\}$, $w = b_j$

$$\begin{aligned}
[\mathcal{L}x]_w &= \deg(w) \cdot [x]_w - \sum_{\substack{i=1 \\ i \neq j}}^{2n} [x]_{b_i} \\
&= (2n-1) \cdot [x]_{b_j} - \sum_{\substack{i=2 \\ i \neq j}}^{2n-1} [x]_{b_i} - [x]_{b_1} - [x]_{b_{2n}} \\
&= (2n-1) \cdot 0 - (2n-3) \cdot 0 - 1 \cdot \beta - 1 \cdot (-\beta) \\
&= 0
\end{aligned}$$

- $w = a_1$

$$\begin{aligned}
[\mathcal{L}x]_w &= \deg(w) \cdot [x]_w - \sum_{i=2}^{n} [x]_{a_i} - [x]_{b_1} \\
&= n \cdot \alpha - (n-1) \cdot \gamma - 1 \cdot \beta
\end{aligned}$$

- $w = c_1$

$$\begin{aligned}[\mathcal{L}x]_w &= \deg(w)\cdot[x]_w - \sum_{i=2}^{n}[x]_{c_i} - [x]_{b_{2n}} \\ &= n\cdot(-\alpha) - (n-1)\cdot(-\gamma) - 1\cdot(-\beta)\end{aligned}$$

- $w = b_1$

$$\begin{aligned}[\mathcal{L}x]_w &= \deg(w)\cdot[x]_w - \sum_{i=2}^{2n}[x]_{b_i} - [x]_{a_1} \\ &= 2n\cdot\beta - (2n-2)\cdot 0 - 1\cdot(-\beta) - 1\cdot\alpha \\ &= (2n+1)\beta - \alpha\end{aligned}$$

- $w = b_{2n}$

$$\begin{aligned}[\mathcal{L}x]_w &= \deg(w)\cdot[x]_w - \sum_{i=1}^{2n-1}[x]_{b_i} - [x]_{c_1} \\ &= 2n\cdot(-\beta) - (2n-2)\cdot 0 - 1\cdot\beta - 1\cdot(-\alpha) \\ &= -(2n+1)\beta + \alpha\end{aligned}$$

The eigenvector equation (8.2) is fulfilled iff the following equations hold:

$$\lambda_2\gamma = \gamma - \alpha \tag{8.3}$$
$$\lambda_2\alpha = n\alpha - (n-1)\gamma - \beta \tag{8.4}$$
$$\lambda_2\beta = (2n+1)\beta - \alpha \tag{8.5}$$

These equations (8.3), (8.4) and (8.5) form a homogeneous linear equation system:

$$\underbrace{\begin{pmatrix} 1-\lambda_2 & -1 & 0 \\ 1-n & n-\lambda_2 & -1 \\ 0 & -1 & 2n+1-\lambda_2 \end{pmatrix}}_{=M} \cdot \begin{pmatrix}\gamma\\ \alpha\\ \beta\end{pmatrix} = \begin{pmatrix}0\\0\\0\end{pmatrix} \tag{8.6}$$

and so it is sufficient to show that equation (8.6) has a non-trivial solution. Since $\det M$ vanishes, M is singular and so equation (8.6) has a non-trivial solution.

None of γ, α and β can be zero. Otherwise consider the following cases:

- If $\gamma = 0$ then it follows by equation (8.3) that $\alpha = 0$ and thus equation (8.4) implies $\beta = 0$. So $(\gamma, \alpha, \beta)^T$ is the trivial solution of equation (8.6) which is a contradiction.
- If $\alpha = 0$ and $\beta \neq 0$ then equation (8.5) implies that λ_2 equals $2n + 1$. Since n is a natural number this is a contradiction to lemma 8.3.
- If both α and β are zero then equation (8.4) implies $n = 1$ or $\gamma = 0$. Since $n > 1$ we have that $(\gamma, \alpha, \beta)^T$ is the trivial solution which is a contradiction.
- Otherwise $\beta = 0$ and by using equation (8.5) we get $\alpha = 0$. As in the previous case we obtain a contradiction.

If γ, α and β have the same sign we are finished. If $\gamma, \alpha, \beta < 0$ we consider the vector $(-\gamma, -\alpha, -\beta)^T$. This yields the vector $-x$, and since x is an eigenvector of λ_2, so is $-x$. Otherwise we distinguish the following cases:

- γ and α have different signs:

  The subgraphs induced by A and C are maximal components of a 2–component partition, because they are complete graphs on n vertices. Because γ and α have different signs, both components have vertices with different signs. This is a contradiction to theorem 3.37.

- α and β have different signs:

  By proposition 3.34 we know that for

  $$\pi(z) = \frac{\sum_{(u,w)\in E}([z]_u - [z]_w)^2}{\sum_{w\in V}([z]_w)^2}$$

  we have

  $$\min_{0\neq z\perp \mathbf{1}} \pi(z) = \pi(x) = \lambda_2.$$

  Now we show that there is a vector y such that $0 \neq \pi(y) < \lambda_2$, y is orthogonal to $\mathbf{1}$ and y has the same length as x. We define y by:

  $$[y]_w := \begin{cases} -\beta & \text{if } w = b_1 \\ \beta & \text{if } w = b_{2n} \\ [x]_w & \text{otherwise} \end{cases}$$

Then y is orthogonal to $\mathbf{1}$ and x and y have the same length. Let F be the set of all edges except for $\{a_1, b_1\}$ and $\{b_{2n}, c_1\}$. By a simple calculation, we see that

$$\sum_{\{u,w\}\in F} ([x]_u - [x]_w)^2 = \sum_{\{u,w\}\in F} ([y]_u - [y]_w)^2.$$

Because of the symmetry of x, y and $\mathcal{L}$ it is sufficient to consider only the edge $\{a_1, b_1\}$. We have

$$\begin{aligned} & ([x]_{a_1} - [x]_{b_1})^2 &=& (\alpha - \beta)^2 \\ & ([y]_{a_1} - [y]_{b_1})^2 &=& (\alpha + \beta)^2 \\ \Longrightarrow \quad & ([y]_{a_1} - [y]_{b_1})^2 - ([x]_{a_1} - [x]_{b_1})^2 &=& -4\alpha\beta \end{aligned}$$

Since α and β have different signs $-4\alpha\beta$ is negative and thus $\pi(y)$ is smaller than $\pi(x)$ which is a contradiction.

- γ and β have different signs:

  Since none of γ, α and β are zero α has a different sign from either γ or β. This is a contradiction to the last two cases. □

Now we reformulate the condition $\det M = 0$ of proposition 8.5. We interpret $\det M$ as a function in two parameters and call it ψ:

$$\begin{array}{ccccccc} \psi & : & [1,\infty[& \times & [0,1] & \longrightarrow & \mathbb{R} \\ & & \zeta & , & \lambda & \longmapsto & \det \begin{pmatrix} 1-\lambda & -1 & 0 \\ 1-\zeta & \zeta-\lambda & -1 \\ 0 & -1 & 2\zeta+1-\lambda \end{pmatrix} \end{array} \tag{8.7}$$

We can calculate ψ explicitly:

Lemma 8.6
Let $\zeta \geq 1$ and $0 \leq \lambda \leq 1$. Then the following equation holds:

$$\psi(\zeta, \lambda) = -\lambda^3 + (2 + 3\zeta)\lambda^2 - (1 + 3\zeta + 2\zeta^2)\lambda + 2\zeta. \tag{8.8}$$

Proof:
We just use the formula for determinants of 3×3 matrices and obtain:

$$\begin{aligned}
\psi(\zeta,\lambda) &= +(1-\lambda)\cdot(\zeta-\lambda)\cdot(2\zeta+1-\lambda) \\
&\quad -\Big((1-\lambda)+(-1)\cdot(1-\zeta)\cdot(2\zeta+1-\lambda)\Big) \\
&= -\lambda^3+\lambda^2(3\zeta+2)+\lambda(-2\zeta^2-4\zeta-1)+2\zeta^2+\zeta \\
&\quad +\lambda-1+\lambda(\zeta-1)-2\zeta^2+\zeta+1 \\
&= -\lambda^3+(2+3\zeta)\lambda^2-(1+3\zeta+2\zeta^2)\lambda+2\zeta
\end{aligned}$$

This also shows that ψ's domain can be expanded to $\mathbb{R}^2$. □

Because of proposition 8.5, we need to know where ψ vanishes. The next proposition (8.7) handles this:

Proposition 8.7
Let $\zeta \in \mathbb{R}$ with $\zeta \geq 3$. Then there exists exactly one real number $\widetilde{\lambda} \in \mathbb{R}$ with $0 < \widetilde{\lambda} < 1$ such that

$$\psi(\zeta,\widetilde{\lambda}) = 0. \tag{8.9}$$

Proof:
Using the formula provided by lemma 8.6 we see that ψ is a continuously differentiable function in both arguments. So it is sufficient to show the following two statements:

$$\forall\,\zeta \geq 3,\ \forall\,\lambda \in [0,1]: \qquad \frac{\partial\psi(\zeta,\lambda)}{\partial\lambda} < 0 \tag{8.10}$$

$$\forall\,\zeta \geq 3: \qquad \psi(\zeta,0) > 0 \quad \text{and} \quad \psi(\zeta,1) < 0 \tag{8.11}$$

First we prove (8.10), so fix $\zeta \geq 3$.

$$\begin{aligned}
\frac{\partial\psi(\zeta,\lambda)}{\partial\lambda} &= -3\lambda^2+(4+6\zeta)\lambda-1-3\zeta-2\zeta^2 \\
&\leq 4+6\zeta-1-3\zeta-2\zeta^2 \qquad (8.12)\\
&= -2\zeta^2+3\zeta+3 \\
&< 0 \qquad (8.13)
\end{aligned}$$

Inequation (8.12) is true since λ is contained in $[0,1]$ and there we have

$$\lambda \geq 0 \qquad \text{and} \qquad -\lambda \geq -1.$$

Inequation (8.11) is true since $\zeta \geq 3$ and therefore we have

$$2\zeta^2 \underbrace{-3\zeta - 3}_{\geq -4\zeta} \geq 6\zeta - 4\zeta \geq 2\zeta > 0$$

This completes the proof of the first statement (8.10). The second statement can be proved by a simple computation. We have:

$$\begin{aligned} \psi(\zeta, 0) &= 2\zeta > 0 \\ \psi(\zeta, 1) &= -1 + 2 + 3\zeta - 1 - 3\zeta - 2\zeta^2 + 2\zeta \\ &= -2\zeta^2 + 2\zeta \end{aligned}$$

Since $\zeta > 1$ we have $-2\zeta^2 + 2\zeta < 0$ and so we finished the whole proof. □

Unfortunately we lack a proof which shows that for any natural number n the solution $\widetilde{\lambda}$ (with $\zeta = n$) of proposition 8.7 coincides with $\lambda_2(G^{(1)}(n))$. But we have several clues. First, proposition 8.7 still holds even if the range is restricted of $\widetilde{\lambda}$ to $[\frac{1}{5\zeta}, \frac{2}{\zeta}]$. These bounds were calculated with the help of equations (3.15) and (3.16). The diameter of $G^{(1)}(n)$ is 5 and we used the balanced cut of size 2 in equation (3.15). The so computed value may not be the minimal value, but is an upper bound. Another clue is lemma 8.4. It implies a certain structure in x_2 and a certain kind of reduction. We assume that the eigen structure of $\mathcal{G}^{(1)}$ is somehow contained in the eigen structure of G_5 which is displayed in figure 8.2.

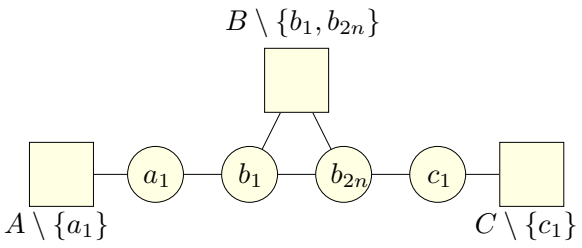


Figure 8.2: Reduction graph G_5

We calculated all eigenvalues and corresponding eigenvectors of $\mathcal{L}(G_5)$. Figure 8.3 shows these pairs.

0	(0.3779, 0.3779, 0.3779, 0.3779, 0.3779, 0.3779, 0.3779)
0.3003	(−0.5725, −0.4005, −0.1082, 0, 0.1082, 0.4005, 0.5725)
0.7530	(−0.4815, −0.1189, 0.3332, 0.5345, 0.3332, −0.1189, −0.4815)
2.2391	(−0.4061, 0.5033, 0.2858, 0, −0.2858, −0.5033, 0.4061)
2.4450	(0.3332, −0.4815, −0.1189, 0.5345, −0.1189, −0.4815, 0.3332)
3.8019	(−0.1189, 0.3332, −0.4815, 0.5345, −0.4815, 0.3332, −0.1189)
4.4605	(−0.0848, 0.2936, −0.6376, 0, 0.6376, −0.2936, 0.0848)

Figure 8.3: Table with all eigenpairs of G_5

The first column contains the eigenvalue and the second the corresponding eigenvector. Due to readability we omitted the transpose symbol for these vectors. The entries of the vectors correspond to the following vertex order:

$$A \setminus \{a_1\},\ a_1,\ b_1,\ B \setminus \{b_1, b_{2n}\},\ b_{2n},\ c_1,\ C \setminus \{c_1\}.$$

Note that G_5 does not belong to $\mathcal{G}^{(1)}$. We calculated the spectrum and the basis consisting of eigenvectors for few $G^{(1)}(n)$ with small $n \in \mathbb{N}$. Then we compared the $+, -, 0$–structure of these eigenvectors with the eigenvectors of G_5 and noted a great set of matches. For example, the only mismatches between the eigenstructure of G_5 and $G^{(1)}(3)$ were eigenvectors of $\mathcal{L}\left(G^{(1)}(3)\right)$ which balanced $A \setminus \{a_1\}$, $B \setminus \{b_1, b_6\}$ or $C \setminus \{c_1\}$ and set the rest to zero. These vectors cannot have a counterpart in G_5 because there these sets are represented by one vertex.

Therefore we assume that the conditions of proposition 8.5 hold for every natural number $n > 2$. Thus proposition 8.5 implies that approach 3 yields the cut T (see lemma 8.2). Obviously this is a negative result in graph partitioning. But not necessarily for cluster analysis. It is quite clear that the minimum balanced cut has size 2 and corresponds to a clustering with two (balanced) components and minimal number of inter–cluster edges. But the component which contains both A and C is not connected. Therefore we have a component such that half of the elements may be very different from the other half. In general this may not be desired. Considering the other balanced cut T we have many inter–cluster edges, but both components are at least connected. This example shows that a trade–off can quite hard to find.

Next we consider the graph family $\mathcal{G}^{(1)}$ in the context of approach 5. We have currently no proof that the MST approach calculates a better partitioning. Although lemma 8.4 holds for the normalised matrix too and lemma 8.6 can be stated in a similar way. We lack a proof for the cluster interpretation part. Thus we have not presented the other part. Using our implementation presented in chapter 7 we tested several examples of this graph family $\mathcal{G}^{(1)}$. The result was always the partition (A, B, C).

8.2 Experimental remarks

Several graphs with interesting and sometimes confusing cluster properties are presented in this part. These graphs appeared during our experimental phases. Thus we give only a short characterisation including the "problems" we have had with this structure. But we neither show comparisons to other approaches nor proofs concerning optimality.

We use the same principles for the drawings as in subsection 7.2.2.

8.2.1 Paths and grids

From our point of view both paths and grids, especially with constant edge weight function, without question lack a cluster. Nevertheless they have a very regular structure and may often occur in applications. Computer networks which are designed for parallel computation often have a grid–like composition.

When we started to cluster paths we expected a cluster which is symmetric. This is due to the fact that paths are axial symmetric. Using the original MST approach with two or more eigenvectors the result was highly asymmetric. Figure 8.4 shows such a cluster with $k = 2$ and $\alpha_1 = \alpha_2 = 1$. The edge weight function s was constant.

The final cluster has a component of size six, two, one and one. In our opinion this was really strange and so we considered the potentials on their own.

Figure 8.5 displays the various potentials. The x–axes represents a vertex–ordering. The y–axes shows the valuation of the eigenvectors. The two solid lines with little marks ($+$ and $\times$) correspond to the potential of the first and

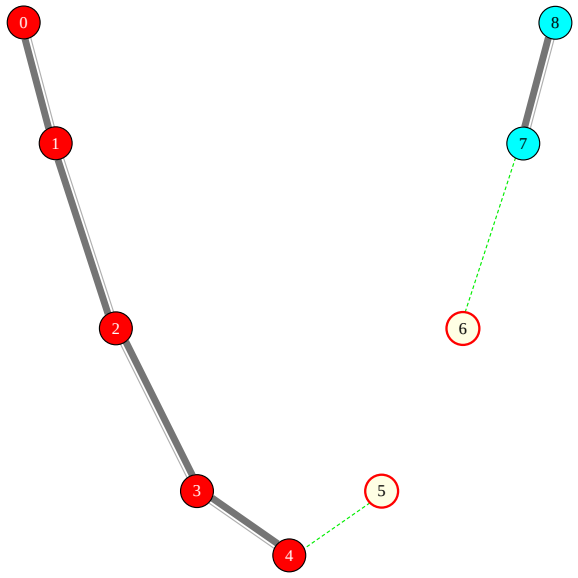


Figure 8.4: Clustering of a path calculated by a MST approach

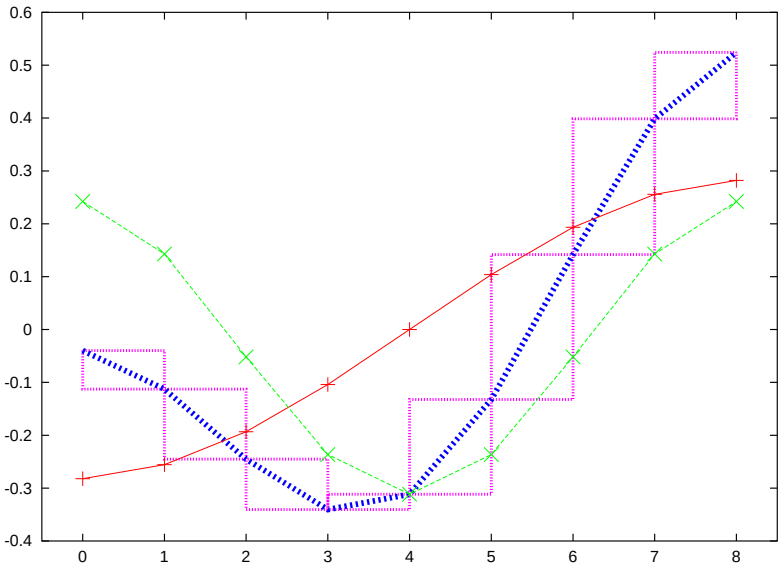


Figure 8.5: Potential of the eigenvectors x_1 and x_2

second eigenvector. The dashed line represents the combination of these two potentials. The dotted boxes mark the differences of two successive vertices according to the final potential. Figure 8.6 displays the cluster calculated with the edge MST approach to the same input parameters. We use the first option (or equivalent $\alpha = 0$) for the threshold parameter.

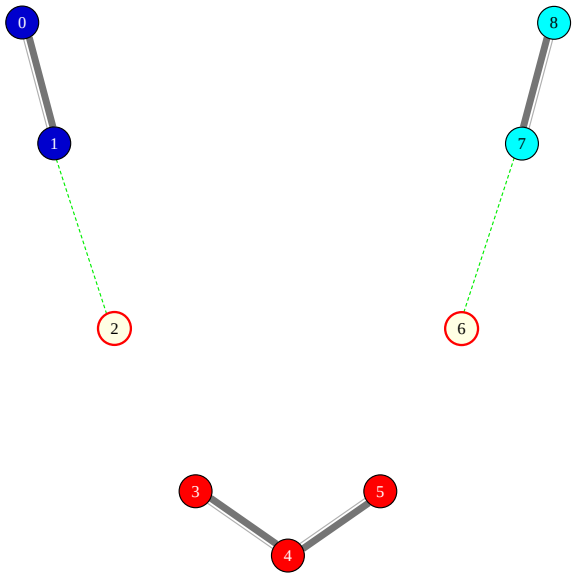


Figure 8.6: Clustering of a path calculated by an edge MST approach

Like mentioned in subsection 6.5.1 this little example clarifies that simply combining the potential may lead to strange partitions. Nevertheless the MST approach may be used to identify parts of the input graph where the entries of certain eigenvectors have similar absolute values. For example the vertex subset $V' := \{0, 1, 2, 3, 4\}$. Each edge in the subgraph induced by V' has very low weight with respect to the potentials' difference.

For grids the situations is quite similar. Among other things this is due to the fact that a grid can be seen as Cartesian product of two paths. Some spectral properties carries over from the factors to the product at least when concerning the Laplacian matrix. We consider only grids with constant edge weight function. Both approaches, the MST and the edge MST approach, create similar clusters for non–square grids. The components are the "axes"

with shorter length. Figure 8.7 displays such a clustering. We used an edge MST approach with $k = 2$, $\alpha_1 = \alpha_2 = 1$ and $\alpha = 0$.

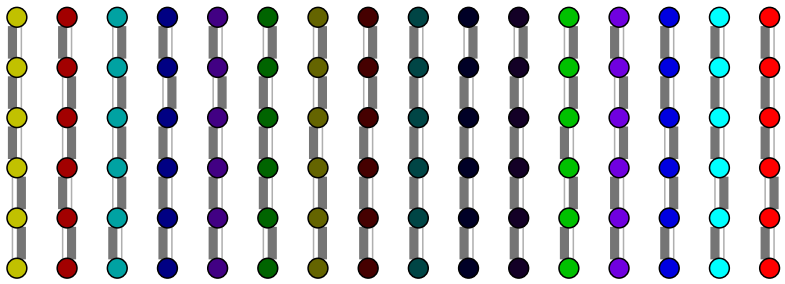


Figure 8.7: A clustering of a 16×6 grid with an edge MST approach

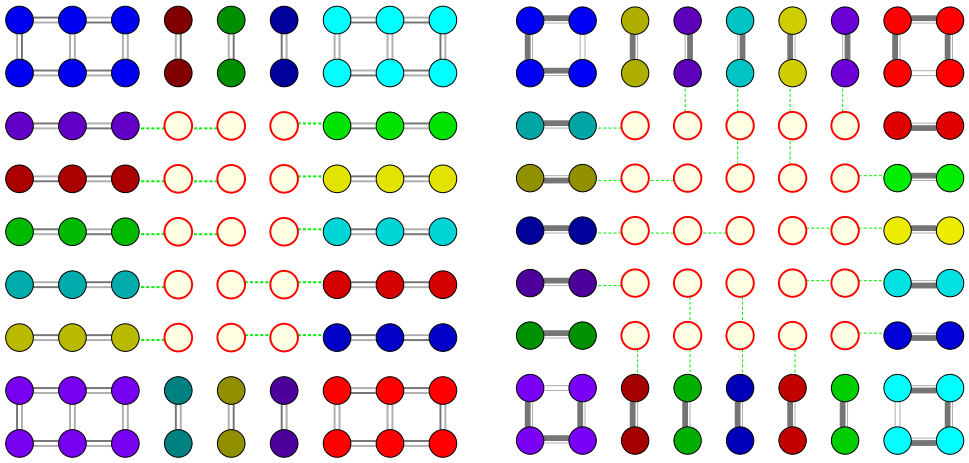
The interesting effects involved square grids. There the MST approach still calculated the "axes" as components while the edge MST approach started with the four corners and created layers around them. Figure 8.8 displays this situation. The MST approach used $k = 1$, $\alpha_1 = 1$ and $\alpha = 0$ and the edge MST approach used $k = 3$, $\alpha_1 = \alpha = 2 = \alpha_3 = 1$ and $\alpha = 0$.

At first glance it may seem strange to compare an approach using one eigenvector with another using three eigenvectors. But the differences between the edge MST approach using one and the one using three eigenvectors are so small that we assume that they can be neglected. The interesting part is the behaviour while increasing the threshold value. The MST approach increases the outer most components with an axes each. The edge MST approach does not always add surrounding layer to the extremal components.

Figure 8.9 and figure 8.10 show different stages during the increasing process. We remark that the edge MST approach has components of size one during almost the whole process while the MST approach does not.

8.2.2 Warts and satellites

We skipped this effect in our research. Nevertheless we are aware of the fact that warts and satellites often conceal important structural elements.



(a) MST approach (b) Edge MST approach

Figure 8.8: A 9×9 grid as input for a MST and an edge MST approach

Warts and satellites are subgraphs which have "bad" cluster properties and are attached to "good" clusters.

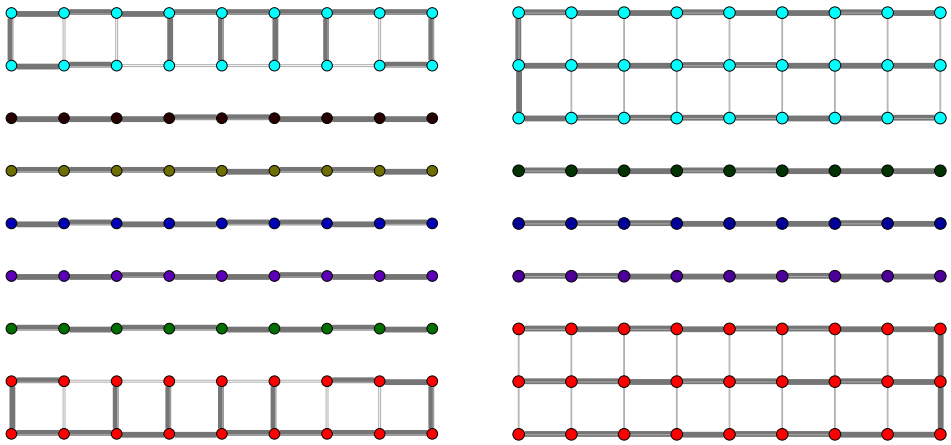
Figure 8.11 shows an example of a wart. The thickness of the edges corresponds to the degree of similarity. The subgraph formed by all vertices which have a circled shape is complete. The wart part is the subgraph induced by all square shaped vertices.

A related phenomenon was the chaining effect of single linkage. This can be interpreted as the connection of warts or satellites. It can alternatively be seen in the way that two components are connected by a wart or a satellite.

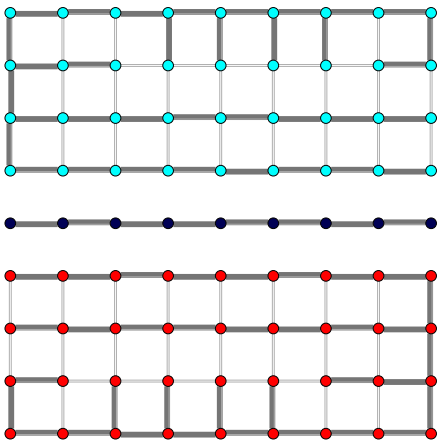
We assume that most warts or satellites can be removed in a pre–processing step.

8.2.3 Pattern matching

Until now we assumed that most cluster properties occur simultaneously. For example: Subgraphs are dense and have a high degree of similarity. The correctness of this assumption depends mostly on the input.

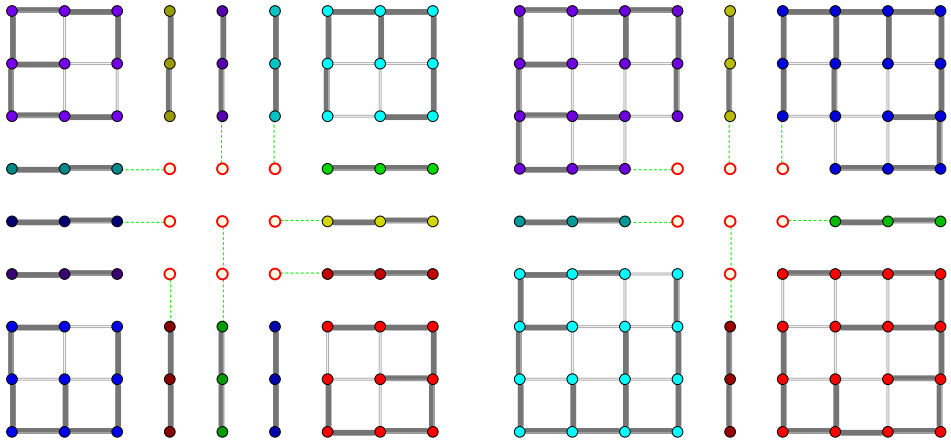


(a) $\alpha = 1$ (b) $\alpha = 2.5$

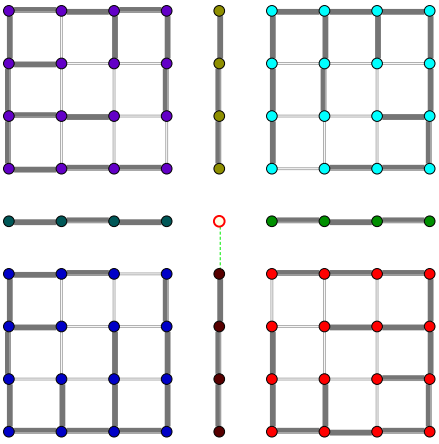


(c) $\alpha = 3$

Figure 8.9: Cluster created by the MST approach with various threshold values Δ



(a) $\alpha = 1$ (b) $\alpha = 2.5$



(c) $\alpha = 3$

Figure 8.10: Cluster created by the edge MST approach with various threshold values Δ

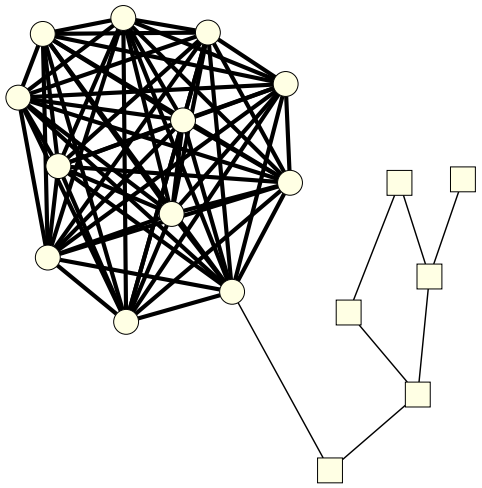


Figure 8.11: An example of a wart

We considered the following example: We take three copies of a complete graph on 16. Since vertices with identical neighbourhoods have identical entries in some eigenvectors a number of edges is removed. We remove three in the first copy, seven in the second and nine in the third copy. The remaining edges receive 1 as degree of similarity. Next we insert a new vertex v_c and choose one vertex in each copy. Finally we connect the chosen vertices with v_c. Each edge which is incident to v_c receives 100 as degree of similarity. Figure 8.12 shows the final clustering created by an edge MST approach with $k = 2$, $\alpha_1 = \alpha_2 = 1$ and $\alpha = 0$.

We note that the average weight function and the maximal weight function for both the MST and the graph are very close to zero ($\approx 10^{-16}$). So the finite arithmetic may cloud this result a bit. Through similar experiments we conjecture that the eigenvectors of the normalised adjacency matrix take more account of dense structure than sparse parts with high degree of similarity. We also assume that there is a graph family where both patterns are regarded with the same effort.

Another interesting question is: Should a cluster approach search for local patterns or global ones? Assume two copies of one structure are weighted with different degree of similarity and the copies are connected by few edges

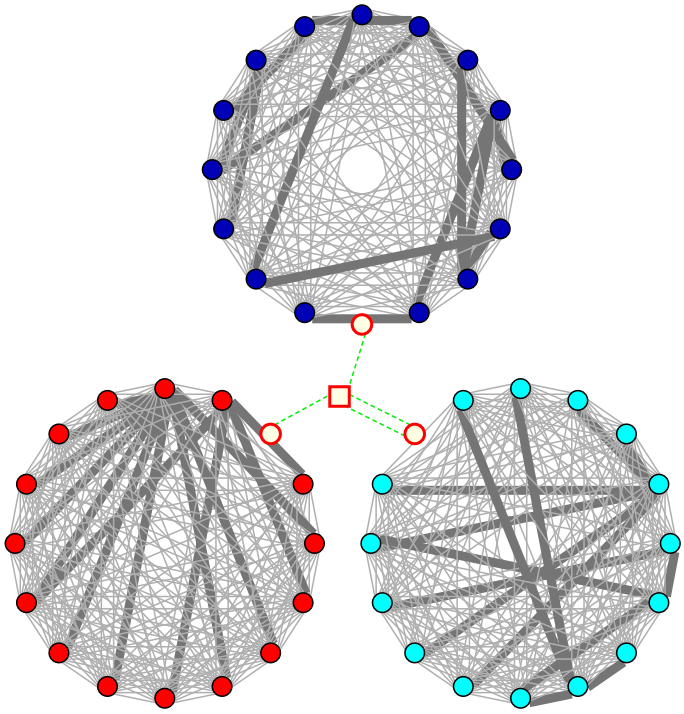


Figure 8.12: Example for: density versus high degree of similarity

with low degree of similarity. What would be a good choice for the cluster? If the copy with the higher degree of similarity is used as "reference" the other copy is split into smaller parts. Another possibility is that both parts are independently clustered. The first scenario describes an global view. The "density" is more or less equal for each component and independent form its neighbourhood. The second version presents a local view. Thereby components take into account more the structure than the values.

For our experiment we used the graph G_4 present on page 69 in figure 5.2. In one copy we scaled the degree of similarity by a factor 10. The other copy kept the original values; these ranges in $[0.353, 1]$. We added four edges between the two copies all with a degree of similarity of 0.00025. (Bold edges which are alternatingly drawn dashed and dotted.)

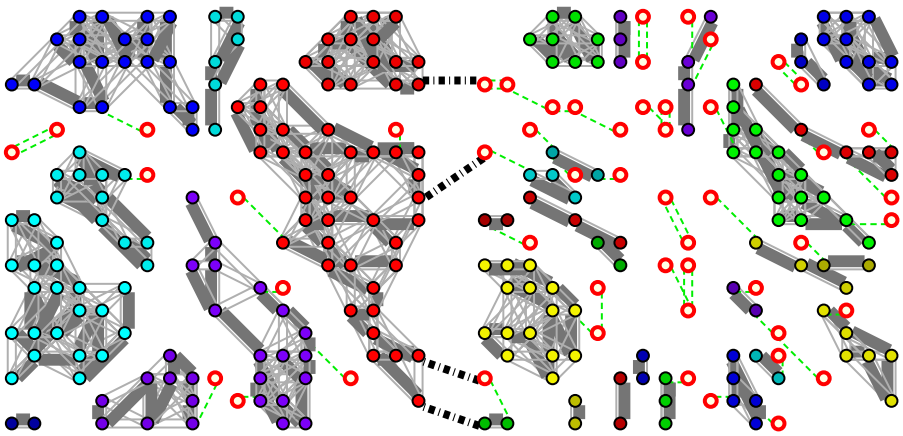


Figure 8.13: Example for: global versus local view

Figure 8.13 shows the result. The cluster was calculated with an edge MST approach with $k = 2$, $\alpha_1 = \alpha_2 = 1$ and $\alpha = 0$. The left copy is the one with the higher degree of similarity. Thus we assume that the edge MST approach combines local and global pattern.

8.2.4 Sparse graphs

A very important issus which also could be addressed in the view of pattern matching is the problem of super–sparse graphs. We think in first order of trees and planar graphs. In general we conjecture that graphs where the number of edges is linear in the number of vertices and the constant is very low can hardly be clustered with a general cluster approach. This is based on our experiences with some random planar graphs.

But no rule without exceptions. Our counter–example is the graph which models a soccer ball. This graph is planar, has vertex–connectivity three and every facet is a pentagon or a hexagon. We used the edge MST approach with $k = 3$, $\alpha_1 = \alpha_2 = \alpha_3$ and $\alpha = 0$. The result is show in figure 8.14.

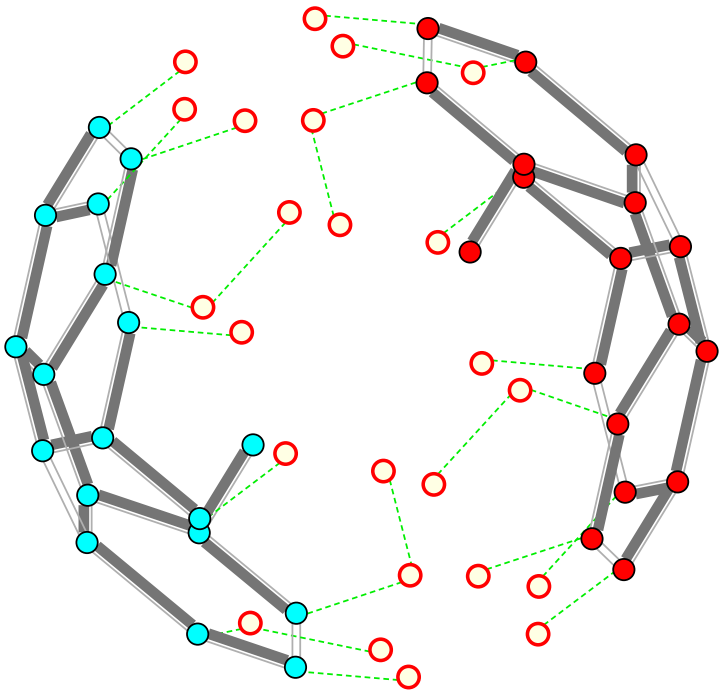


Figure 8.14: Soccer ball graph

This completes our experimental remarks.

Chapter 9

Final remarks

9.1 Conclusion

The main contribution of this thesis is a new graph clustering method with linear space requirement. It is a modification of spectral bisection based on the same intuition as the MCL process. Furthermore it works directly on the graph structure and the spectral information is used as basic partitioning guide.

Therefore we think that the opening question can be affirmed. Although we have neither proved nor presented a fundamental connection between the MCL process and the MST approach we conjecture that the MST approach and its derivates are worthwhile to be considered on their own. In various examples we showed their usability and flexibility. Especially online and interactive clustering benefits from their composition.

Finally the counter–examples presented cases where users have to make trade–offs and adapt the approaches.

9.2 Outlook

For the future we plane to have several benchmark and optimality tests. We hope to show some optimality guarantees or connections to the MCL process by the help of these data. The data can also be used to empirically prove the usability and flexibility.

Other interesting points are finding better automatical methods for the threshold parameters, testing other combination procedures, making the MST approach more adaptable for online and interactive clustering and the influences of exchanging eigenvectors.

Some open problems

We restricted the input graph to be undirected or bidirected. Concerning the case of a digraph which is not undirected. What does the MST approach calculated? What about the calculated vectors? Are they still eigenvectors having some partitioning properties? Does the connection between the Laplacian and the normalised adjacency matrix still hold?

These problems illustrate that there are a lot of uncovered cases to investigate and a lot of approach to discover.

Appendix A

Matrices and weighted graphs

As mentioned in section 3.4 we state some definitions for weighted graphs. These concern basic elements like: degree and adjacency structure.

Throughout the rest of this part we consider a graph $G = (V, E)$ together with an edge weight function s. To handle associated matrices and vectors we assume a fixed vertex ordering.

Definition A.1
Let $v \in V$. The weighted in–degree, out–degree and degree of v is defined as:

$$\begin{aligned}
\deg_s^{\rightarrow}(v) &:= \sum_{\substack{e \in E \\ \text{source}(e)=v}} s(e) \\
\deg_s^{\leftarrow}(v) &:= \sum_{\substack{e \in E \\ \text{target}(e)=v}} s(e) \\
\deg_s(v) &:= \sum_{\substack{w \in V \\ (v,w) \in E}} s((v,w)) + \sum_{\substack{u \in V \\ (u,v) \in E}} s((u,v))
\end{aligned}$$

The equations valid for the undirected case hold in the directed case too:

$$\begin{aligned}
\deg_s(v) = \deg_s^{\leftarrow}(v) + \deg_s^{\rightarrow}(v) \qquad & \text{for directed graphs} \\
\deg_s(v) = \deg_s^{\leftarrow}(v) = \deg_s^{\rightarrow}(v) \qquad & \text{for undirected graphs}
\end{aligned}$$

The next definition is just the formal description of the weighted adjacency matrix. Informally, it was mentioned in section 2.4.

Definition A.2
The weighted adjacency matrix $\mathcal{A}_s(G)$ is a square matrix of order $|V|$ and its entries are defined by:

$$[\mathcal{A}_s(G)]_{u,v} := \begin{cases} s(e) & \text{if } e = (u,v) \\ 0 & \text{otherwise} \end{cases}.$$

These definitions A.1 and A.2 are sufficient to carry over the remaining definitions of section 3.4. Usually we assume that no edge has zero weight. Some properties of the weighted Laplacian matrix can be found in [CO00] and [vdHP00].

Appendix B

Summary of numerical calculation in eigen theory

The following results in [QSS02], [EMU96] and in [GvL96].

First we need to fix the aim of the calculation. There exist different approaches for:

- the calculation of some eigenvalues only
- the calculation of all eigenvalues only
- the calculation of some eigenvalues with associated eigenvector
- the calculation of all eigenvalues and associated eigenvectors
- the calculation of some eigenvectors only

For our cluster approaches we are very interested in the last problem. We can calculate an eigenvector x to an eigenvalue λ of a matrix A by finding a nontrivial solution to the linear equation system: $(A - \lambda I)x = 0$.

B.1 General problem

The calculation of eigenvalues of a matrix of order n is equivalent to calculate roots of polynomials of degree n. One application of Galois–theory is the

ABEL theorem, which states that the roots of a general polynomial cannot be expressed by a finite expression built by addition, multiplication and k–th roots of the coefficients of the polynomials and constants, if its degree is 5 or higher. For details see [Bos99]. Therefore every approach is an indirect method, which only converges towards an eigen solution. Since we use finite arithmetic, it is sufficient to calculate an approximate solution.

B.2 Local approach

The power–method is a very simple and easily implementable method calculating one eigenpair.

Algorithm 3: Power method

Input : a matrix A and a start vector x
Output: an eigenpair
begin
  set $k \leftarrow 0$, $\lambda^{(0)} \leftarrow 0$ and $x^{(0)} \leftarrow v$
  while $(\lambda^{(k)}, x^{(k)})$ *is not a suitable eigenpair* **do**
    $k \leftarrow k + 1$
    $y^{(k)} \leftarrow A \cdot x^{(k-1)}$
    $x^{(k)} \leftarrow \texttt{adjust}(y^{(k)})$
    $\lambda^{(k)} \leftarrow \left(x^{(k)}\right)^T A x^{(k)}$
  output $(\lambda^{(k)}, x^{(k)})$
end

Algorithms 3 shows the generic power–method. The `adjust`–step is usually only a rescaling step, such that $q^{(k)}$ has unit length. In this situation we have, that the power–method calculated an eigenpair, such that $|\lambda| = \varrho(A)$. In this case convergence is not guaranteed, if more eigenvalues have absolute norm $\varrho(A)$. For details see [QSS02] or [EMU96, sec. 7.3]. The `adjust`–step can be modified such that convergence towards certain eigenpairs is ensured. This is based on the GRAM–SCHMIDT orthogonalisation process, see lemma 3.17. Since this method is popular many similar procedures were developed; for more information see [QSS02, sec. 5.3].

B.3 Global approach

One well–known approach which calculates all eigenvalues is the QR–method and its derivations. It is named in this way since the fundamental idea is a similarity transformation which is calculated with the help of a matrix decomposition called QR–decomposition. Due to several aspects, among other things like manageability and convergence, the input matrix A is transformed. For more details see [QSS02, sec. 5.4–7] or [EMU96, sec. 7.7]. Since the matrix's sparsity is not preserved during the initial transformation step as well as the similarity transformation, this procedure is not suitable.

B.4 Extreme eigenvalues

The last approach is suitable for calculating few extreme eigenvalues. It is called Lanczos approach and is originally used to solve linear equation systems. For details see [QSS02, sec. 5.11]. It gained its popularity since it maintains the matrix's sparsity and has an acceptable runtime as well as needed space. Some versions lack numerically stability.

Literature note

During the study of the computation of eigenvalues and eigenvectors we also used [Wil72], [GvL96] and [TDB97]. In these references we found different introductions to this topic as well as various versions and different analysis tools.

Appendix C

Some basic calculation regarding the graph structure

In this part we have collected some standard calculations of linear algebra which can be expressed by the use of graph notation.

Lemma C.1
Let $H = (V, E)$ be a graph and s an edge weight function. Then the graph $H' := \mathsf{G}\left(\mathcal{A}\left(H\right)^2\right)$ can be calculated in $\mathcal{O}\left(|V| \cdot \Delta(H)^2\right)$ time.

Proof:
With the weighted analogue of lemma 3.25 it is sufficient to calculate all paths of length two between all pairs of vertices. Let us fix an arbitrary vertex v. Then we reach all vertices in N with a path of length 2 with

$$N := \bigcup_{w \in \mathrm{N}^{\rightarrow}(v)} \mathrm{N}^{\rightarrow}(w).$$

The set N contains at most $\Delta(H)^2$ vertices, since $\deg^{\rightarrow}(u) \leq \Delta(H)$ for all vertices $u \in V$.

To calculate H' we override the original input graph H and add an edge weight function t to H'. Only the edge set E' of H' differs from the edge set E of H. The vertex set is the same for both H and H'. Both H' and t will be calculated in an incremental way.

We have to keep track of newly inserted edges. Therefore we use an additional edge weight ϵ initialised to:

$$\forall\, e \in E \colon \epsilon(e) = 1.$$

Also t is initialised to be zero for all elements in E. To update one vertex we use algorithm 4. We omit to prove the correctness of it.

Algorithm 4: Update vertex v

Input : H', s, t, ϵ and v
Output: update of H', s, t and ϵ
begin
  forall $w \in \mathrm{N}^{\rightarrow}(v)$ **do**
    let e_w be the edge connecting v with w in H'
    if $\epsilon(e_w) \neq 0$ **then**
      forall $u \in \mathrm{N}^{\rightarrow}(w)$ **do**
        let e_u be the edge connecting w with u in H'
        if $\epsilon(e_w) \neq 0$ **then**
          if $(v, u) \notin E'$ **then**
            insert edge e' connecting v with u in H'
            $\epsilon(e') \leftarrow 0$
            $t(e') \leftarrow 0$
          let e' be the edge connecting v with u in H'
          $t(e') \leftarrow t(e') + s(e_w) \cdot s(e_u)$
end

Since every vertex needs only to be updated once, we call algorithm 4 for each vertex. Since algorithm 4 considers each element of N once and uses constant time for the inner most loop, we have proved the postulated time complexity. □

Next we state some lemmas containing edge weight functions and vectors indexed by the vertex set. Therefore let T_x be the access time for one element using a vector and T_s the access time for one element using an edge weight function.

Lemma C.2
Let $H = (V, E)$ be a graph, s an edge weight function and x a vector indexed by the vertices of V. Let $\mathcal{A}$ be the weighted adjacency matrix of H with respect to s and having the same vertex ordering as x. Then the vector $\mathcal{A} \cdot x$

can be calculated in $\mathcal{O}(|E| \cdot T_s \cdot T_x + |V| \cdot T_x)$ *time and by the use of* $\mathcal{O}(|V|)$ *space.*

Proof:
Algorithm 5 does the computation. It requires $\mathcal{O}(|V|)$ space to save the result and runs in the desired time, since every edge is considered twice and we need its weight and the weight of its source and its target.

Algorithm 5: Matrix–vector multiplication

```
Input  : A, s and x
Output: Ax
begin
    allocate a new vector y
    // fixing one entry of y
    forall v ∈ V do
        [y]_v ← 0
        // calculating the addends of entry of [y]_v
        forall w ∈ N→(v) do
            let e_w be the edge connecting v with w
            [y]_v ← [y]_v + s(e_w) · [x]_w
    output y
end
```

Algorithm 5 is correct, because it is just a serial implementation of the definition of a matrix–vector multiplication. □

If we assume $T_x, T_s \in \mathcal{O}(1)$ then we can replace $\mathcal{A}$ by $\mathcal{L}(H)$ or $\mathcal{N}(H)$ without changing the runtime of lemma C.2. This is due to the fact that these matrices respects the adjacency structure and add a diagonal element or rescale each row.

Lemma C.3
Let $H = (V, E)$ *be a graph and* x, y *to vectors. Then any weighted scalar product of these two vectors can be calculated in* $\mathcal{O}(|V| \cdot T_x^3)$ *time and constant space. If the weight is constant we require only* $\mathcal{O}(|V| \cdot T_x^2)$ *time. Scaling one vector can be done in* $\mathcal{O}(|V| \cdot T_x)$ *time and requires* $\mathcal{O}(|V|)$ *space for saving the result.*

Proof:
Simply use the definition in lemma 3.13 and the scalar–vector multiplication. □

Lemma C.4
Let $H = (V, E)$ be a graph, $x_1, \ldots, x_k$ orthogonal vectors and x an arbitrary vector not contained in the span of $x_1, \ldots, x_k$. Let x_{k+1} be the orthogonal projection, see lemma 3.17. Then x_{k+1} can be calculated with $\mathcal{O}(|V|)$ space and in $\mathcal{O}(|V| \cdot (2k) \cdot T_x^2 + |V| \cdot (k+1) \cdot T_x)$ time.

Proof:
Algorithm 6 does the computation. It requires $\mathcal{O}(|V|)$ space. It has the desired runtime, since we have to compute the sum of $k+1$ vectors and $2k$ scalar products.

Algorithm 6: Vector orthogonalisation

Input : V, $x_1, \ldots, x_k$ and x
Output: x_{k+1}
begin
  allocate a new vector y
  forall $v \in V$ **do**
    $[y]_v \leftarrow [x]_v$
  for $i = 1, \ldots, k$ **do**
    $Z \leftarrow 0$
    $N \leftarrow 0$
    // calculating the coefficients for the linear combination
    forall $v \in V$ **do**
      $Z \leftarrow Z + [x]_v \cdot [x_i]_v$
      $N \leftarrow N + ([x_i]_v)^2$
    $Z \leftarrow Z/N$
    // calculating the next part for the linear combination
    forall $v \in V$ **do**
      $[y]_v \leftarrow [y]_v - Z[x_i]_v$
  output y
end

Algorithm 6 is correct, since it is an implementation of the lemma 3.17. □

With these lemmas we can evaluate the power–method, see appendix B, in the graph situation. Therefore we assume $T_x, T_s \in \mathcal{O}(1)$.

Proposition C.5
Let $H = (V, E)$ be a graph, s an edge weight function and x a vector. If we calculate k iterations of the power–method with $A \in \{\mathcal{A}, \mathcal{N}, \mathcal{L}\}$ then we require $\mathcal{O}(k \cdot (|E| + |V| + \mathcal{T}))$ time, where $\mathcal{T}$ is the runtime of the `adjust`*–step. In addition we need $\mathcal{O}(|V|)$ space.*

Proof:
For each step we first calculate a matrix–vector multiplication, second call `adjust` and finally calculate a matrix–vector multiplication and a scalar product. Lemma C.2 implies that we require $\mathcal{O}(|E| + |V|)$ time for a matrix–vector. By lemma C.3 we know that we need $\mathcal{O}(|V|)$ time to calculate a scalar product. Therefore we have the following runtime for each step:

$$\mathcal{O}\left(2 \cdot (|E| + |V|) + |V| + \mathcal{T}\right) \subseteq \mathcal{O}(|E| + |V| + \mathcal{T})$$

This completes the proof. □

Appendix D

List of symbols

D.1 Graph notation

$[G]$	underlying undirected skeleton of G	page 11
$\mathrm{source}(e)$	source of the edge e	page 12
$\mathrm{target}(e)$	target of the edge e	page 12
$\mathrm{N}(v)$	neighbourhood of v	page 12
$\mathrm{N}^{\leftarrow}(v)$	all neighbours which are adjacent to v	page 12
$\mathrm{N}^{\rightarrow}(v)$	all neighbours to which v is adjacent	page 12
$\deg(v)$	degree of v	page 13
$\deg^{\leftarrow}(v)$	in–degree of v	page 13
$\deg^{\rightarrow}(v)$	out–degree of v	page 13
$\delta(G)$	minimal degree in G	page 13
$\Delta(G)$	maximal degree in G	page 13
$u \overset{p}{\rightsquigarrow} v$	path p connected u with v	page 13
$\eta(v_s, v_t)$	number of edge disjoint path connecting v_s with v_t	page 13
$\kappa(v_s, v_t)$	number of vertex disjoint path connecting v_s with v_t	page 13
$\eta(G)$	minimal number of edge disjoint paths connecting two (different) vertices in G	page 14
$\kappa(G)$	minimal number of vertex disjoint paths connecting two (different) vertices in G	page 14

$E\|V'$	the subset of all edges of E such that both source and target if contained in V'	page 14
$\overline{S}$	the complement of S, i.e. $V \setminus S$	page 16
$E(V_i, V_j)$	the set of edges with source in V_i and target in V_j	page 17
∂S	the set of all cut edges of $(S, \overline{S})$	page 17

D.2 Matrix notation

$[A]_{i,j}$	the ij–the entry of A	page 23
$[x]_i$	the i–the entry of x	page 23
A^T	the transpose of A	page 23
I_n	unit matrix of order n	page 23
$\mathbf{1}$	vector with all entries equal to one	page 23
E_λ	eigenspace of eigenvalue λ	page 24
p_A	characteristic polynomial of A	page 24
$\Lambda(A)$	the spectrum of A	page 24
$\varrho(A)$	the spectral radius of A	page 24
$A \geq 0$	A is nonnegative	page 25
$A > 0$	A is positive	page 25
$\mathcal{N}_0(n)$	the set of all nonnegative $n \times n$ matrices	page 25
$\mathcal{N}_+(n)$	the set of all positive $n \times n$ matrices	page 25
$\mathcal{N}_S(n)$	the set of all stochastic $n \times n$ matrices	page 27
$\mathcal{S}(n)$	the set of all symmetric $n \times n$ matrices	page 28
$\|x\|$	the (euclidian) length of x	page 30
$\mathsf{G}(A)$	to A associated digraph	page 34
$\mathcal{I}(G)$	a incidency matrix of G	page 34
$\mathcal{L}(G)$	the Laplacian matrix of G	page 39
$\mathcal{D}(G)$	the degree matrix of G	page 39
$\Lambda_{\mathcal{L}}(G)$	the Laplacian spectrum of G	page 43
$\mathcal{L}_N(G)$	the normalised Laplacian matrix of G	page 47
$\Lambda_{\mathcal{L}_N}(G)$	the normalised Laplacian spectrum of G	page 47
$\mathcal{N}(G)$	the normal matrix of G	page 48
$\Lambda_{\mathcal{N}}(G)$	the normalised spectrum of G	page 48

D.3 Cluster notation

$\mathbb{I}_{\mathbb{P}}$	set of all inter–cluster edges	page 53
$\Xi_s(S)$	partition weight with respect to edge weight function s and nonempty proper vertex subset S	page 59
$\phi(S)$	conductance of nonempty proper vertex subset S respectively a cut $(S, \overline{S})$	page 59
$\phi(G)$	conductance of a graph	page 59
$\phi(\mathbb{P})$	conductance of a partition	page 59
$\sigma_x(\cdot)$	induced edge weight function by a vertex weight function x	page 63

D.4 Minimal spanning tree

$\mathbb{M}(G, s)$	set of all minimal spanning trees of G with respect to edge weight function s	page 84

Bibliography

[AK95] C. J. Alpert and A. B. Kahng. Recent directions in netlist partitioning: A survey. *Integration: The VLSI J.*, 19:1–18, 1995.

[And73] Michael R. Anderberg. *Cluster analysis for applications*, volume 19 of *Probability and mathematical statistic*. Academic press, 1973.

[Bos99] Siegfried Bosch. *Algebra*, chapter 6, pages 256–274. Springer–Verlag, 3. edition, 1999.

[Chu94] Fan R. K. Chung. *Spectral Graph Theory*. Number 52 in Conference Board of the Mathematical Sciences. American Mathematical Society, 1994.

[CK94] Pierluigi Crescenzi and Viggo Kann. A compendium of NP optimization problems, 1994. `http://www.nada.kth.se/~viggo/problemlist/compendium.html`.

[CKMD02] Steven R. Corman, Timothy Kuhn, Robert D. McPhee, and Kevin J. Dooley. Studying complex discursive systems: Centering resonance analysis of communication. *Human Communication Research*, 28(2), 2002. In print.

[CO00] Fan Chung and Kevin Oden. Weighted Graph Laplacians and Isoperimetric Inequalities. *Pacific Journal of Mathematics*, 192:257–274, 2000.

[DHJ+95] F. Daly, D.J. Hand, M.C. Jones, A.D. Lunn, and K.J. McConway. *Elements of Statistics*. The Open University, Addision–Wesley, 1995.

[EMU96] Giesela Engeln-Müllges and Frank Uhlig. *Numberical Algorithms with C*. Springer–Verlag, 1996.

[Fas99] Daniel Fasulo. An analysis of recent work on clustering algorithms. Technical Report 01-03-02, Department of Computer Science & Engineering, Box 352350, University of Washington, Seattle WA 98195, April 1999.

[Fie73] Miroslav Fiedler. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal*, 23(98):298–305, 1973.

[Fie75] Miroslav Fiedler. A property of eigenvectors of nonnegative symmetric matrices and its applications to graph theory. *Czechoslovak Mathematical Journal*, 25(100):619–633, 1975.

[Fie86] Miroslav Fiedler. *Special matrices and their applications in numerical mathematics*. Martinus Nijhoff Publishers and SNTL, 1986.

[Fis97] Gerd Fischer. *Lineare Algebra*. Vieweg, 11. edition, 1997.

[FKPT01] Ludwig Fahrmeir, Rita Künstler, Iris Pigeot, and Gerhard Tutz. *Statistik, Der Weg zur Datenanalyse*. Springer–Verlag, 3. edition, 2001.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability, A Guide to the theory of NP–Completeness*. W. H. Freeman and Company, 11 edition, 1979.

[Gon85] Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

[GR01] Chris Godsil and Gordon Royle. *Algebraic Graph Theory*. Springer–Verlag, 1. edition, 2001.

[GvL96] Gene H. Golub and Charles F. van Loan. *Matrix Computation*. The John Hopkins University Press, 3. edition, 1996.

[Har37] John A. Hartigan. *Clustering Algorithms*. Wiley series in probability and statistics. John Wiley & Sons, 1937.

[Hen] Monika Henzinger. `http://www.henzinger.com/monika/↪index.html`.

[HS00] Erez Hartuv and Ron Shamir. A clustering algorithm based on graph connectivity. *Information Processing Letters*, 76(4-6):175–181, 2000.

[JD88] Antil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.

[Jun99] Dieter Jungnickel. *Graphs, Networks and Algorithms*, volume 5 of *Algorithms and Computation in Mathematics*. Springer–Verlag, 1999.

[Kle99] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999. `http://www.cs.cornell.edu/home/kleinber/auth.ps`.

[KVV00] Ravi Kannan, Santosh Vampala, and Adrian Vetta. On Clustering — Good, Bad and Spectral. In *FOCS 2000*, 2000.

[Lan72] Serge Lang. *Linear algebra*. Addison–Wesley Publishing Company, 2. edition, 1972.

[Moh91] Bojan Mohar. The laplacian spectrum of graphs. In Y. Alavi, G. Chartand, O.R. Oellermann, and A.J. Schwenk, editors, *Graph Theory, Combinatorics and Applications*, volume 2, pages 871–898. John Wiley & Sons, Inc., 1991.

[Moh97] Bojan Mohar. Some Applications of Laplace Eigenvalues of Graphs. In G. Hahn and G. Sabidussi, editors, *Graph Symmertry: Algebraic Methods and Applications*, volume 497 of *NATO ASI Series C*, pages 227–275, 1997.

[QSS02] Alfio Quarteroni, Riccardo Sacco, and Fausto Saleri. *Numerische Mathematik*, volume 1. Springer–Verlag, 2002.

[SR97] Andres Seary and William Richards. The physics of networks, 1997. `http://www.sfu.ca/~richards/Pdf-ZipFiles/↪physics.pdf`.

[TDB97] Lloyod N. Trefethen and III David Bau. *Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, 1997.

[vD00] Stijn Marinus van Dongen. *Graph Clustering by Flow Simulation*. PhD thesis, University of Utrecht, 2000.

[vdHP00] Jan van den Heuvel and Snežana Pejić. Using Laplacian Eigenvalues and Eigenvectors in the Analysis of Frquency Assignment Problems. Technical Report LSE-CDAM-2000-20, Centre for Discrete and Applicable Mathematics, Department of Mathematics, London School of Economics, Houghton Street, London WC2AE, UK, December 2000.

[WF94] Stanley Wasserman and Katherine Faust. *Social Network Analysis Methods and Applications*. Cambridge University Press, 1994.

[Wil72] J.H. Wilkinson. *The Algebraic Eigenvalue Probelm*. Clarendon Press, 1972.

[Zup82] Jure Zupan. *Clustering of Large Data Sets*. Research Studies Press, 1982.

Index

Index