available at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cosrev

Survey

Graph clustering

*Satu Elisa Schaeffer**

Laboratory for Theoretical Computer Science, Helsinki University of Technology TKK, P.O. Box 5400, FI-02015 TKK, Finland

ARTICLE INFO

Article history:
Received 29 January 2007
Received in revised form
8 May 2007
Accepted 28 May 2007

ABSTRACT

In this survey we overview the definitions and methods for graph clustering, that is, finding sets of "related" vertices in graphs. We review the many definitions for what is a cluster in a graph and measures of cluster quality. Then we present global algorithms for producing a clustering for the entire vertex set of an input graph, after which we discuss the task of identifying a cluster for a specific seed vertex by local computation. Some ideas on the application areas of graph clustering algorithms are given. We also address the problematics of evaluating clusterings and benchmarking cluster algorithms.

1. Introduction

Any nonuniform data contains underlying structure due to the heterogeneity of the data. The process of identifying this structure in terms of grouping the data elements is called *clustering*, also called *data classification* [152]. The resulting groups are called *clusters*. The grouping is usually based on some *similarity measure* defined for the data elements. Clustering is closely related to *unsupervised learning* in pattern recognition systems [81]. A basic task in unsupervised learning is to classify a data set into two or more classes based on a similarity measure over the data, without resorting to any *a priori* information on how the classification should be done.

Graphs are structures formed by a set of *vertices* (also called *nodes*) and a set of *edges* that are *connections* between pairs of vertices. *Graph clustering* is the task of grouping the vertices of the graph into clusters taking into consideration the edge structure of the graph in such a way that there should be many edges *within* each cluster and relatively few *between* the clusters. Graph clustering in the sense of grouping the vertices of a given input graph into clusters, which is the topic of this survey, should *not* be confused with the clustering of *sets of graphs* based on structural similarity; such clustering of graphs as well as measures of graph similarity is addressed in other literature [38,124,168,169,202,206], although many of the techniques involved are closely related to the task of finding clusters *within* a given graph.

As the field of graph clustering has grown quite popular and the number of published proposals for clustering algorithms as well as reported applications is high, we do not even pretend to be able to give an exhaustive survey of *all* the methods, but rather an explanation of the methodologies commonly applied and pointers to some of the essential publications related to each research branch.

1.1. Outline of the survey

We begin by providing basic definitions in Section 2. In Section 3 we proceed to defining the task of graph clustering by discussing the different definitions of clusterings and clusters. These definitions lead into definitions of similarity measures discussed in Section 4, global clustering algorithms summarized in Section 5, and local methods presented in

* *Corresponding address:* Universidad Autónoma de Nuevo León, Facultad de Ingeniería Mecánica y Eléctrica, Posgrado en Ingeniería de Sistemas (PISIS), AP 126-F, Ciudad Universitaria, San Nicolás de los Garza, NL 66450, Mexico.
E-mail address: elisa.schaeffer@gmail.com.

1574-0137/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cosrev.2007.05.001

Section 6. Section 7 discusses the difficulty of comparing, evaluating and benchmarking graph-clustering methods. Applications are reviewed in Section 8. In Section 9 we glance at open problems and future directions, and in Section 10 we conclude the survey.

2. Terminology and definitions

In this section we first review the necessary terminology to facilitate discussion in the rest of the survey. We provide some of the basic definitions of computational complexity, approximation algorithms, graph theory, and Markov chains. Readers familiar with these topics are encouraged to proceed directly to Section 3 on page section.

2.1. Computational complexity

The *worst-case running time* of an algorithm for a problem instance of size x is the number of computation steps needed to execute the algorithm for the most difficult instance of size x possible. The instance size is measured in some fixed units, typically integers or bits — the effect of the unit selection will vanish as we proceed to the definitions of complexity used in the survey. Hence x is a positive integer, $x \in \mathbb{Z}^+$.

Equally, the worst-case memory consumption is the number of memory units that the algorithm will need to *simultaneously* occupy in the worst possible case for an instance of size x. In computational complexity, the interest is in characterizing how the running time and memory consumption grow when x grows. Let $f(x)$ be a function of x that determines the number of computation steps (or alternatively the units of memory) needed in the worst case, given $x \in \mathbb{Z}^+$.

The *worst-case complexity* of an algorithm is denoted by $\mathcal{O}(g(x))$, where $g(x)$ is a function of the input size x such that $f(x)$ *grows no faster than* $g(x)$. This means that there exists a positive constant c such that

$$|f(x)| \leq c \cdot |g(x)| \tag{1}$$

for all sufficiently large values of x. In general, $g(x)$ is formed by ignoring constant multipliers in $f(x)$ and only keeping the highest-order term.

Stating that an algorithm has run time or memory consumption $f(x) = \Omega(h(x))$ in turn means that there exists a positive constant d such that

$$|f(x)| \geq d \cdot |h(x)| \tag{2}$$

for all sufficiently large values of x. The difference is that $f(x) = \mathcal{O}(g(x))$ provides an upper bound, whereas $f(x) = \Omega(h(x))$ is a lower bound on how the complexity grows. Furthermore, we write that $f(x) = \Theta(g(x))$ if both $f(x) = \mathcal{O}(g(x))$ and $f(x) = \Omega(g(x))$ hold.

This kind of "rounding" of the functions and the study of their behaviour for large values of x is called *asymptotic analysis*. For more information on these notations for worst-case and complexity and other related definitions, we recommend the basic textbook on algorithms by Cormen et al. [60].

Decision problems are characterized by a set of *problem instances* and a set of *solutions*, together with a relation associating a particular problem instance to a possibly empty subset of solutions. Essentially, in a decision problem, we ask whether the solution set mapped to a given instance is nonempty. If a solution can be constructed to a given problem instance in time that is polynomial to the length of the representation of the problem instance, the corresponding decision problem is said to be in class **P**. Formally, a problem is in **P** if it has an algorithm with time complexity bounded by some polynomial of the input size x.

For quite few important problems, there are no known polynomial-time algorithms. However, such a problem may still have a polynomial-time *verification algorithm* that can check whether a given *certificate* y that has length polynomial in x provides a feasible solution to a problem instance of size x. Formally, **NP** is the class of all languages that are decided by *nondeterministic Turing machines* in polynomial time [192]. Practically, this means that problems with polynomial-time verification algorithms form the class **NP**. Note that **P** is a subclass of **NP**.

Furthermore, a decision problem S is *reducible* to a decision problem T if there exists a polynomial-time reduction f such that for any $x \in S$, $f(x) \in T$. This is denoted by $S \leq_m^{\mathbf{P}} T$. A problem T is said to be **NP**-*hard* if $S \leq_m^{\mathbf{P}} T$ for all problems $S \in$ **NP**. An **NP**-hard problem T is said to be **NP**-*complete* if additionally $T \in$ **NP**. In this survey, many **NP**-complete problems will be mentioned. For more information on **NP**-completeness and the complexity classes, we recommend the classical reference text of Garey and Johnson [103] and the textbook of Papadimitriou [192].

2.2. Approximation algorithms

In some applications, it may not be worth the effort to compute the best possible solution to the problem at hand, but a not-too-bad solution will suffice. Whenever exact computation is time-consuming, impossible, or simply not justified by the needs of the application, *heuristic* and approximate methods are useful. Many such methods provide a *nondeterministic* output, meaning that the method may output a different solution on different executions. However, one may need to repeatedly execute a heuristic algorithm and then filter the output with respect to some quality measure.

The goal of an *approximation algorithm* is to find efficiently a solution that differs no more than a fixed factor from the exact solution. By efficient, one usually means "in polynomial time". Approximation algorithms are the practical approach for solving large instances of **NP**-complete problems and problems harder than that. A good approximation algorithm should have *provably* polynomial running time.

The problems for which approximation algorithms are most commonly used are *optimization* problems. In an optimization problem the task is to choose from a large set of possible solutions the one that gives the best value for a certain function. The goal may either be the minimization of a cost function or the maximization of a fitness function.

When searching for an approximate solution to an optimization problem, it is a matter of application to define how much the approximate solution may differ from

the exact solution to be acceptable. As the approximation algorithm does not aim to find the best possible solution, but rather one that is not very far from being the best, one should present *provable* bounds on how far from the optimal solution can the approximation solution be. Such a bound is called the *approximation factor* and is less than one for maximization problems and greater than one for minimization problems.

The extremal value of the factor over all problem instances is (minimum for maximization problems and maximum for minimization problems) called the *approximation ratio*. A *constant-factor* approximation algorithm is one where the value of the solution found is at most a constant-multiple of the optimum solution. If there exists a systematic method to approximate the solution to arbitrary factors, the method is called a (polynomial-time) *approximation scheme*, abbreviated **PTAS**. The complexity class of problems that have a **PTAS** is also denoted by **PTAS**.

The complexity class **NPO** is a function problem class: find an n-bit string x that maximizes a given cost function $C(x)$, where the function C is computable in polynomial time by a deterministic Turing machine. The class **NPO** is called **NP**-*optimization*. The complexity class **APX** is a subclass of **NPO** such that each problem in **APX** allows constant-factor polynomial-time approximation algorithms. There exist problems that are in **APX** but not **PTAS**, unless **P** = **NP**. **APX**-hard problems have a **PTAS**-reduction from every other problem in **APX**. Assuming **P** ≠ **NP**, no **APX**-hard problem can have a **PTAS**. We recommend the book of Vazirani [226] on approximation algorithms as well as the book of Ausiello et al. [14] on the complexity of approximation algorithms.

2.3. *Graph theory*

A *graph* G is a pair of sets $G = (V, E)$. V is the set of *vertices* and the number of vertices $n = |V|$ is the *order* of the graph. The set E contains the *edges* of the graph. In an *undirected graph*, each edge is an unordered pair $\{v, w\}$. In a *directed* graph (also called a *digraph* in much literature), edges are ordered pairs. The vertices v and w are called the *endpoints* of the edge. The edge count $|E| = m$ is the *size* of the graph. In a *weighted graph*, a weight function $\omega : E \to \mathbb{R}$ is defined that assigns a weight on each edge. A graph is *planar* if it can be drawn in a plane without any of the edges crossing.

In this survey, we define the *density* of a graph $G = (V, E)$ as the ratio of the number of edges present to the maximum possible,

$$\delta(G) = \frac{m}{\binom{n}{2}}. \tag{3}$$

For $n \in \{0, 1\}$, we set $\delta(G) = 0$. A graph of density one is called *complete*.

If $\{v, u\} \in E$, we say that v is a *neighbour* of u. The set of neighbours for a given vertex v is called the *neighbourhood* of v and is denoted by $\Gamma(v)$. A vertex v is a member of its own neighbourhood $\Gamma(v)$ if and only if the graph contains a reflexive edge $\{v, v\}$.

The *adjacency matrix* $\mathbf{A}_G$ of a given graph $G = (V, E)$ of order n is an $n \times n$ matrix $\mathbf{A}_G = (a^G_{v,u})$ where

$$a^G_{v,u} = \begin{cases} 1, & \text{if } \{v, u\} \in E, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

The number of edges incident on a given vertex v is the *degree* of v and is denoted by $\deg(v)$. A graph is *regular* if all of the vertices have the same degree; if $\forall v \in V$ in $G = (V, E)$ we have $\deg(v) = k$, the graph G is k-regular. The diagonal *degree matrix* of a graph $G = (V, E)$ is

$$\mathbf{D} = \begin{pmatrix} \deg(v_1) & 0 & 0 & \dots & 0 & 0 \\ 0 & \deg(v_2) & 0 & \dots & 0 & 0 \\ 0 & 0 & \deg(v_3) & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & \deg(v_{n-1}) & 0 \\ 0 & 0 & 0 & \dots & 0 & \deg(v_n) \end{pmatrix}. \tag{5}$$

A partition of the vertices V of a graph $G = (V, E)$ into two nonempty sets S and $V\backslash S$ is called a *cut* and is denoted by $(S, V\backslash S)$. A cut is uniquely identified by defining a set S; hence any subset of V can be called a cut. As the sets S and $V\backslash S$ define the same cut, it is often preferred to denote by S the *smaller* set, hence requiring $|S| \leq \lfloor \frac{n}{2} \rfloor$.

The *cut size* is the number of edges that connect vertices in S to vertices in $V\backslash S$:

$$c(S, V\backslash S) = |\{\{v, u\} \in E \mid u \in S, v \in V\backslash S\}|. \tag{6}$$

We denote by

$$\deg(S) = \sum_{v \in S} \deg(v) \tag{7}$$

the sum of degrees in a cut S. Note that in the presence of edge weights, the cut size is generally redefined as the *sum of the weights* of the edges crossing the cut instead of using simply the *number* of edges that cross it.

A *path* from v to u in a graph $G = (V, E)$ is a sequence of edges in E starting at vertex $v_0 = v$ and ending at vertex $v_{k+1} = u$;

$$\{v, v_1\}, \{v_1, v_2\}, \dots, \{v_{k-1}, v_k\}, \{v_k, u\}. \tag{8}$$

If such a path exists, v and u are *connected*. The path is *simple* if no vertex is repeated, that is, for all $i \in [0, k+1]$ and $j \in [0, k+1]$, $v_i \neq v_j$ unless $i = j$.

The *length* of a path is the number of edges on it, and the *distance* between v and u is the length of the *shortest* path connecting them in G. The distance from a vertex to itself is zero: the path from a vertex to itself is an empty edge sequence. A graph is *connected* if there exist paths between *all* pairs of vertices. If there are vertices that cannot be reached from others, the graph is *disconnected*. The minimum number of edges that would need to be removed from G in order to make it disconnected is the *edge-connectivity* of the graph. A *cycle* is a simple path that begins and ends at the same vertex. A graph that contains no cycle is *acyclic* and is also called a *forest*. A connected forest is called a *tree*.

A *subgraph* $G^S = (S, E_S)$ of $G = (V, E)$ is composed of a set of vertices $S \subseteq V$ and a set of edges $E_S \subseteq E$ such that $\{v, u\} \in E_S$ implies $v, u \in S$; the graph G is a *supergraph* of G^S. A connected acyclic subgraph that includes *all* vertices is called a *spanning tree* of the graph. A spanning tree has necessarily exactly $n-1$ edges. If the edges are assigned weights, the spanning tree with smallest total weight is called the *minimum spanning tree*. Note that there may exist several minimum spanning trees that may even be edge-disjoint.

An *induced subgraph* of a graph $G = (V, E)$ is the graph with the vertex set $S \subseteq V$ with an edge set $E(S)$ that includes *all* such edges $\{v, u\}$ in E with both of the vertices v and u included in the set S:

$$E(S) = \{\{v, u\} \mid v \in S, u \in S, \{v, u\} \in E\}. \tag{9}$$

We denote the subgraph induced by the vertex subset S by $G(S)$ (or by G^S where it is clear that the subgraph is an induced subgraph). An induced subgraph that is a complete graph is called a *clique*. An induced subgraph with an empty edge set is called an *independent set*. We define the *local density* of an induced subgraph in $G = (V, E)$ to be simply

$$\delta(G(S)) = \frac{|E(S)|}{\binom{|S|}{2}} \tag{10}$$

in accordance to Eq. (3). An alternative definition of density in the literature is the ratio of the edge count to the vertex count,

$$\delta'(G(S)) = \frac{|E(S)|}{|S|}, \tag{11}$$

which yields possible alternative definitions of global graph density for G, such as the average density

$$\delta'(G) = \frac{m}{n}, \tag{12}$$

or the maximum density

$$\delta'_{\max}(G) = \max_{S \subset V} \frac{|E(S)|}{|S|}. \tag{13}$$

Two graphs $G_i = (V_i, E_i)$ and $G_j = (V_j, E_j)$ are *isomorphic* if there exists a bijective (one-to-one) mapping $f : V_i \to V_j$ (called an *isomorphism*) such that $\{v, w\} \in E_i$ if and only if $\{f(v), f(w)\} \in E_j$.

The *spectrum* of a graph $G = (V, E)$ is defined as the list of *eigenvalues* (together with their multiplicities) of its adjacency matrix $\mathbf{A}_G$. Spectral properties can be computed for both undirected and directed graphs, as well as unweighted and weighted, but by far the easiest case are undirected, unweighted simple graphs, which will be the focus of the brief treatment of spectral graph theory in this section.

It is often more convenient to study the eigenvalues of the *Laplacian* matrix $\mathbf{L} = \mathbf{I} - \mathbf{A}_G$ than those of $\mathbf{A}_G$ itself [51]. The *normalized* Laplacian is defined as

$$\mathcal{L} = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}_G\mathbf{D}^{-\frac{1}{2}}, \tag{14}$$

where $\mathbf{I}$ is an $n \times n$ identity matrix (with ones on the diagonal, other elements being zero). An element-wise definition for the normalized Laplacian, which is easier to understand intuitively than the matrix version, is the following:

$$\mathcal{L}_{uv} = \begin{cases} 1, & \text{if } u = v \text{ and } \deg(v) > 0, \\ -\dfrac{1}{\sqrt{\deg(u) \cdot \deg(v)}}, & \text{if } u \in \Gamma(v), \\ 0, & \text{otherwise.} \end{cases} \tag{15}$$

As these matrices are symmetrical, their eigenvalues are real and nonnegative. Using the normalized Laplacian is convenient as the eigenvalues of $\mathcal{L}$ all fall within the interval $[0, 2]$. The smallest eigenvalue is *always* zero, as the matrix is singular, and the corresponding eigenvector is simply a vector with each element being the square-root of the degree of the corresponding vertex.

A pleasant consequence of having the spectra limited to the interval $[0, 2]$ is that it makes comparing the spectra of two graphs easier [229,51]. However, even nonisomorphic graphs can share the same spectrum [229]. Graphs that have the same spectrum are called *cospectral* (also *isospectral*) [63]. When the equality of the sets of *pairwise distinct eigenvalues* holds, but the multiplicities do not coincide, the graphs are *weakly* cospectral [215]. A survey on to which extent the spectrum determines a graph is given by van Dam and Haemers [223]. The spectra of real-world graphs is studied by Farkas et al. [86].

For comparing spectra of two graphs, it has been found better to compare the spectra of their *line graphs* [63]. A line graph $G_L = (V_L, E_L)$ of a given graph $G = (V, E)$ is a graph where the vertex set V_L corresponds to the set of edges E of the original graph G and two vertices $v_{ij} \in V_L$ (corresponding to the edge $\{v_i, v_j\} \in E$) and $v_{k\ell} \in V_L$ (corresponding to the edge $\{v_k, v_\ell\} \in E$) are connected by an edge $\{v_{ij}, v_{k\ell}\} \in E_L$ if and only if the vertex subsets $\{v_i, v_j\}$ and $\{v_k, v_\ell\}$ have a nonempty intersection,

$$\{v_i, v_j\} \cap \{v_k, v_\ell\} \neq \emptyset, \tag{16}$$

meaning that the two original edges represented by the vertices v_{ij} and $v_{k\ell}$ share one or both of their endpoints.

The (degree-adjusted) *Rayleigh quotient* [51] is the ratio

$$\frac{\sum_{v \in V} g(v)\mathcal{L}g(v)}{\sum_{v \in V} g(v)^2}, \tag{17}$$

where $g(v) : V \to \mathbb{R}$ is viewed as a column vector assigning arbitrary real values to the vertices. Simplifying with an assignment $g(x) = \mathbf{D}^{\frac{1}{2}}f(v)$ and Eq. (14) we get

$$\begin{aligned} \frac{\sum_{v \in V} g(v)\mathcal{L}g(v)}{\sum_{v \in V} g(v)^2} &= \frac{\sum_{v \in V} f(v)(\mathbf{L}f(v))}{\sum_{v \in V} (\mathbf{D}^{\frac{1}{2}}f(v))^2} = \frac{\sum_{v \in V} f(v) \sum_{w \in V} \mathbf{L}_{v,w} f(w)}{\sum_{v \in V} f(v)^2 \deg(v)} \\ &= \frac{\sum_{v \in V} f(v) \sum_{\{v,u\} \in E} \big(f(v) - f(w)\big)}{\sum_{v \in V} f(v)^2 \deg(v)} \\ &= \frac{\sum_{\{v,u\} \in E} \big(f(v) - f(u)\big)^2}{\sum_{v \in V} f(v)^2 \deg(v)}. \end{aligned} \tag{18}$$

It is easy to see that the minimum value of this ratio is zero, obtained by any function $f(v)$ that assigns the same value to the vertices in each connected component of the graph. The Rayleigh quotient in its basic form is written for any real and symmetrical[1] matrix $\mathbf{B}$ and a vector $\mathbf{x}$ as

$$\frac{\mathbf{x}^{\mathrm{T}}\mathbf{B}\mathbf{x}}{\mathbf{x}^{\mathrm{T}}\mathbf{x}} \tag{19}$$

and is widely used as an approximation to the extreme eigenvalues of $\mathbf{B}$: the ratio is minimized when $\mathbf{x}$ is the eigenvector corresponding to the smallest eigenvalue of $\mathbf{B}$

[1] In general, the definition of the Rayleigh quotient works for complex Hermitian matrices.

and this minimum value of the Rayleigh quotient is the eigenvalue itself. Similarly, the maximum gives the largest eigenvalue. These properties hold for the degree-adjusted version as well, and hence zero is the smallest of the real, positive eigenvalue of the normalized Laplacian. The accuracy of iterative optimization of the Rayleigh quotient is studied by Li [164].

The right eigenvector associated to the second-smallest eigenvalue of the Laplacian matrix is called a *Fiedler vector*, named after Fiedler [92,93], for his contributions to algebraic graph theory. If we instead use the normalized Laplacian, the corresponding vector is a *normalized* Fiedler vector.

2.4. *Markov chains*

A *Markov chain* is a stochastic process in which future states only depend on the current state, not the past, taking values from some countable state space. The probabilities for moving to another state from current state form the *transition matrix* of the Markov chain.

In general, each Markov chain, independent of how the transition probabilities are defined, can be represented by a weighted directed graph where each state corresponds to a vertex, each edge corresponds to a transition that has nonzero probability and the edge weight is the probability in question. For an unweighted graph, when one moves from one vertex to another choosing a neighbouring vertex uniformly at random, the transition matrix that results is the *normalized* adjacency matrix $\mathbf{D}^{-1}\mathbf{A}_G$ of the graph G. This means that the probability for moving from vertex v to w is simply

$$p_{v,w} = \begin{cases} \dfrac{1}{\deg(v)}, & \text{if } w \in \Gamma(v), \\ 0, & \text{otherwise.} \end{cases} \tag{20}$$

Such a walk is called *random*, *blind*, *regular* or *simple*, as it is but one of many possible definitions of walks in graphs. One may impose different kinds of weight functions on the neighbouring vertices, which generalized to the definition of a *Markov chain* with the state set being the vertex set of the graph. For more insight to the mathematics and interpretations of random walks on graphs, see for example the survey by Lovász [167] or the textbook in preparation by Aldous and Fill [6]. The latter emphasizes the Markov chain connection.

The *first passage time* from state j to state i is the time step when the chain first visits state i when started at state j. The *absorption time* from state j to state i is the first passage time in a *modified* chain, where state i is made into an absorbing state by removing all of its outbound transitions.

The spectrum of the transition matrix can be used to evaluate the *mixing time* of the chain, which is the time it takes for the chain to reach its *stationary distribution*. The stationary distribution is a distribution that no longer changes over time as more and more transitions are being performed. It defines for each state the probability that the walk is at that state if a single observation is made after the walk has been run for a sufficiently long time. The stationary distribution can be obtained by computing the *left* eigenvector corresponding to the *largest* eigenvalue of the transition matrix. The primary eigenvalue λ_1 of any transition matrix is one, as is the case for any stochastic matrix. The Perron–Frobenius theorem [113] states that for the non-principal eigenvalues, $|\lambda_i| \leq \lambda_1 = 1$. If there are more eigenvalues with the value one, the chain has more stationary distributions.

The eigenvectors form a basis for a vector space. As any vector, including the initial distribution, can be represented as an eigenvalue decomposition in the vector space determined by the eigenvectors, and all λ_i other than those corresponding to stationary distributions have absolute value smaller than one, the corresponding components get smaller and smaller as the chain is ran. This implies that the smaller the eigenvalues λ_i are, the faster the chain converges to the stationary distribution [139]. For estimating the mixing time, the *second* eigenvalue of the transition matrix is already quite useful [211]. For more information on mixing times, we recommend Behrends' book [23].

3. Graph clustering

In this section, we begin the difficult work of defining what constitutes a cluster in a graph and what a clustering should be like; we also discuss some special classes of graphs. In some of the clustering literature, a cluster in a graph is called a *community* [186,107].

Formally, given a data set, the goal of clustering is to divide the data set into clusters such that the elements assigned to a particular cluster are similar or connected in some predefined sense. However, not all graphs have a structure with natural clusters. Nonetheless, a clustering algorithm outputs a clustering for any input graph. If the structure of the graph is completely uniform, with the edges evenly distributed over the set of vertices, the clustering computed by any algorithm will be rather arbitrary. Quality measures – and if feasible, visualizations – will help to determine whether there are significant clusters present in the graph and whether a given clustering reveals them or not.

In order to give a more concrete idea of what clusters are, we present here a small example. On the left in Fig. 1 we have an adjacency matrix of a graph with $n = 210$ vertices and $m = 1505$ edges: the $2m$ black dots (two for each edge) represent the ones of the matrix, whereas white areas correspond to zero elements. When the vertices are ordered randomly, there is no apparent structure in the adjacency matrix and one can not trivially interpret the presence, number, or quality of clusters inherent in the graph. However, once we run a graph clustering algorithm (in this example, an algorithm of Schaeffer [205]) and re-order the vertices according to their respective clusters, we obtain a diagonalized version of the adjacency matrix, shown on the right in Fig. 1. Now the cluster structure is evident: there are 17 dense clusters of varying orders and some sparse connections between the clusters.

Matrix diagonalization in itself is an important application of clustering algorithms, as there are efficient computational methods available for processing diagonalized matrices, for example, to solve linear systems. Such computations in turn enable efficient algorithms for *graph partitioning* [214], as the graph partitioning problem can be written in the form of a set of linear equations. The goal in graph partitioning is to

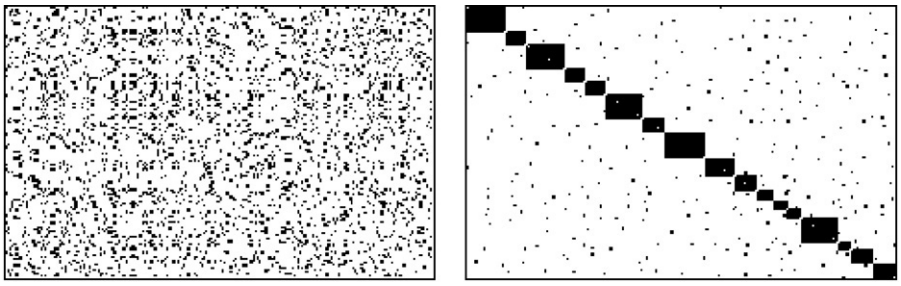


Fig. 1 – The adjacency matrix of a 210-vertex graph with 1505 edges composed of 17 dense clusters. On the left, the vertices are ordered randomly and the graph structure can hardly be observed. On the right, the vertex ordering is by cluster and the 17-cluster structure is evident. Each black dot corresponds to an element of the adjacency matrix that has the value one, the white areas correspond to elements with the value zero.

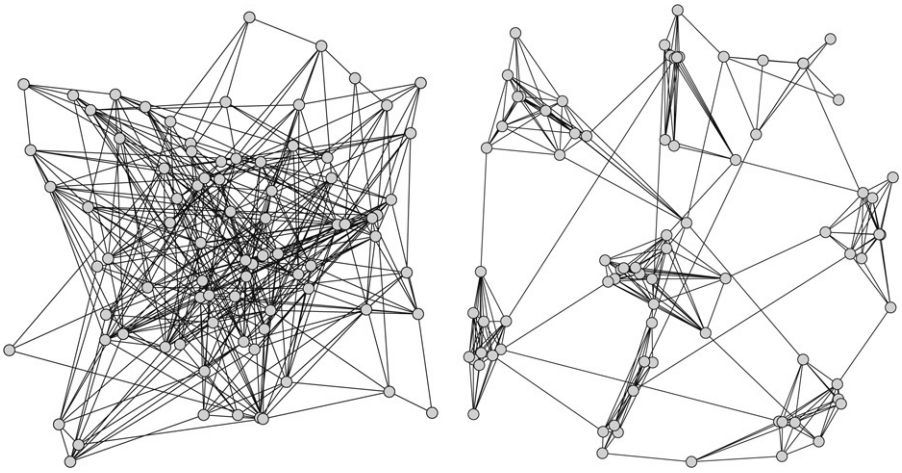


Fig. 2 – Two graphs both of which have 84 vertices and 358 edges. The graph on the left is a uniform random graph of the $\mathcal{G}_{n,m}$ model [84,85] and the graph on the right has a relaxed caveman structure [228]. Both graphs were drawn with spring-force visualization [203].

minimize the number of edges that cross from one subgroup of vertices to another, usually posing limits on the number of groups as well as to the relative size of the groups.

3.1. *Generation models for clustered graphs*

Gilbert [106] presented in 1959 a process for generating *uniform random graphs* with n vertices: each of the $\binom{n}{2}$ possible edges is included in the graph with probability p, considering each pair of vertices independently. In such uniform random graphs, the degree distribution is Poissonian. Also, the presence of dense clusters is unlikely as the edges are distributed by construction uniformly, and hence no dense clusters can be expected.

A generalization of the Gilbert model, especially designed to produce clusters, is the *planted ℓ-partition model* [59]: a graph is generated with $n = \ell \cdot k$ vertices that are partitioned into ℓ groups each with k vertices. Two probability parameters p and $q < p$ are used to construct the edge set: each pair of vertices that are in the same group share an edge with the higher probability p, whereas each pair of vertices in different groups shares an edge with the lower probability r. The goal of the *planted partition problem* is to recover such a planted partition into ℓ clusters of k vertices each, instead of optimizing some measure on the partition. McSherry [173] discusses also planted versions of other problems such as k-clique and graph colouring.

Fig. 2 shows two graphs of the same order and size, one of is a uniform random graph and the other has a clearly clustered structure. The graph on the right is a *relaxed caveman graph*. Caveman graphs were an early attempt in social sciences to capture the clustering properties of social networks, produced by linking together a ring of small complete graphs called "caves" by moving one of the edges in each cave to point to another cave [231]. Also the graph represented in Fig. 1 was generated with the caveman model. These graphs are also especially constructed to contain a clear cluster structure, possibly with a hierarchical structure where clusters can be further divided into natural subclusters.

The procedure to create a relaxed caveman graph is the following [228]: a connection probability $p \in (0, 1]$ of the top level of the hierarchy is given as a parameter, together with a scaling coefficient s that adjusts the density of the lower-level caves. The minimum $n_{\min}$ and maximum $n_{\max}$ for the numbers of subcomponents (subcaves at higher levels, vertices at the bottom level) are given as parameters. The

generation procedure is recursive. The graph on the right in Fig. 2 is a first-level construction.

Planted partition models and relaxed caveman graphs provide good and controllable artificial input data for evaluating and benchmarking clustering algorithms. The underlying cluster structure is determined by the construction, but by defining lower values for the intercluster densities, the structure can be blurred, which increases the difficulty of clustering. The planted partition model is theoretically easier to handle. However, the relaxed caveman model incorporates the generation of a hierarchy and the possibility to generate unequal-sized clusters — both often present in natural data.

3.2. *Desirable cluster properties*

Unfortunately, no single definition of a cluster in graphs is universally accepted, and the variants used the literature are numerous [82]. In the setting of graphs, each cluster should intuitively be *connected*: there should be at least one, preferably several paths connecting each pair of vertices within a cluster. If a vertex u cannot be reached from a vertex v, they should not be grouped in the same cluster. Furthermore, the paths should be *internal* to the cluster: in addition to the vertex set $\mathcal{C}$ being connected in G, the subgraph induced by $\mathcal{C}$ should be connected in itself, meaning that it is not sufficient for two vertices v and u in $\mathcal{C}$ to be connected by a path that passes through vertices in $V\backslash\mathcal{C}$, but they also need to be connected by a path that *only* visits vertices included in $\mathcal{C}$.

As a consequence, when clustering a disconnected graph with known components, the clustering should usually be conducted on each component separately, unless some global restriction on the resulting clusters is imposed. In some applications one may wish to obtain clusters of similar order and/or density, in which case the clusters computed in one component also influence the clusterings of other components.

We classify the edges incident on $v \in \mathcal{C}$ into two groups: *internal* edges that connect v to other vertices also in $\mathcal{C}$, and *external* edges that connect v to vertices that are *not* included in the cluster $\mathcal{C}$:

$$\begin{aligned} \deg_{\text{int}}(v, \mathcal{C}) &= |\Gamma(v) \cap \mathcal{C}| \\ \deg_{\text{ext}}(v, \mathcal{C}) &= |\Gamma(v) \cap (V\backslash\mathcal{C})| \\ \deg(v) &= \deg_{\text{int}}(v, \mathcal{C}) + \deg_{\text{ext}}(v, \mathcal{C}). \end{aligned} \tag{21}$$

Clearly, $\deg_{\text{ext}}(v) = 0$ implies that $\mathcal{C}$ containing v could be a good cluster, as v has no connections outside of it. Similarly, if $\deg_{\text{int}}(v) = 0$, v should not be included in $\mathcal{C}$ as it is not connected to any of the other vertices included.

It is generally agreed upon that a subset of vertices forms a good cluster if the induced subgraph is dense, but there are relatively few connections from the included vertices to vertices in the rest of the graph [24,107,142,151,185]. For examples, see Fig. 3.

One measure that helps to evaluate the sparsity of connections from the cluster to the rest of the graph is the cut size $c(\mathcal{C}, V\backslash\mathcal{C})$. The smaller the cut size, the better "isolated" the cluster. Determining when a cluster is dense is naturally achieved by computing the graph density. We refer to the density of the subgraph induced by the cluster as the *internal* or *intra-cluster density*:

$$\delta_{\text{int}}(\mathcal{C}) = \frac{|\{\{v, u\} \mid v \in \mathcal{C}, u \in \mathcal{C}\}|}{|\mathcal{C}|\,(|\mathcal{C}| - 1)}. \tag{22}$$

The intercluster density of a given clustering of a graph G into k clusters $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k$ is the *average* of the intercluster densities of the included clusters:

$$\delta_{\text{int}}(G \mid \mathcal{C}_1, \ldots, \mathcal{C}_k) = \frac{1}{k} \sum_{i=1}^{k} \delta_{\text{int}}(\mathcal{C}_i). \tag{23}$$

The *external* or *inter-cluster density* is of a clustering is defined as the ratio of intercluster edges to the maximum number of intercluster edges possible, which is effectively the sum of the cut sizes of all the clusters, normalized to the range [0, 1]:

$$\delta_{\text{ext}}(G \mid \mathcal{C}_1, \ldots, \mathcal{C}_k) = \frac{\left|\left\{\{v, u\} \mid v \in \mathcal{C}_i, u \in \mathcal{C}_j, i \neq j\right\}\right|}{n(n-1) - \sum_{\ell=1}^{k} (|\mathcal{C}_\ell|\,(|\mathcal{C}_\ell| - 1))}. \tag{24}$$

Globally speaking, the internal density of a good clustering should be notably higher than the density of the graph $\delta(G)$ (Eq. (3)) and the intercluster density of the clustering should be lower than the graph density [184].

Considering the above requirements of connectivity and density, the loosest possible definition of a graph cluster is that of a *connected component*, and the strictest definition is that each cluster should be a *maximal clique* (i.e. a subgraph into which no vertex could be added without losing the clique property). In most occasions, the semantically useful clusters lie somewhere in between these two extremes. Connected components are easily computed in $\mathcal{O}(n + m)$-time with a breadth-first search, whereas clique detection is **NP**-complete [143,28]. Typically, interesting clustering measures tend to correspond to **NP**-hard decision problems [210,1]. When the input graph is very large, such as the Internet at router level or the Web graph (see Section 3.3.2), it is highly infeasible to rely on algorithms with exponential running time, as even linear-time computation gets tedious.

It is not always clear whether each vertex should be assigned fully to a cluster or could it instead have different

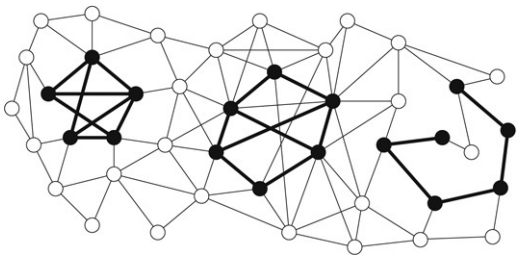


Fig. 3 – An example graph with three clusters chosen; all cluster members are drawn in black and their internal edges are drawn thicker than other edges of the graph. The cluster on the left is of good quality, dense and introvert. The one in the middle has the same number of internal edges, but many more to outside vertices, making it a worse cluster. The cluster on the right has very few connections outside, but lacks internal density and hence is not a good cluster.

"levels of membership" in several clusters? In document clustering, such a situation is easily imaginable: a document can be mainly about fishing, for example, but also address sailing-related issues, and hence could be clustered into "fishing" with 0.9 membership, for example, and to "sailing" with a level of 0.3. Another solution would be creating a supercluster to include all documents related to fishing and sailing, but the downside is that there can be documents on fishing that have no relation to sailing whatsoever.

For general clustering tasks, fuzzy clustering algorithms have been proposed [129,105], as well as validity measures [239]. Within graph clustering, not much work can be found on fuzzy clustering, and in general, the past decade has been quiet on the area of fuzzy clustering. Yan and Hsiao [241] present a fuzzy graph-clustering algorithm and apply it to circuit partitioning. A study on general clustering methods using fuzzy set theory is presented by Dave and Krishnapuram [68].

A *fuzzy graph* $G_R = (V, R)$ is composed of a set of vertices and a fuzzy edge-relation R that is reflexive and symmetrical together with a membership function μ_R assigns to each fuzzy edge a level of "presence" in the graph [75]. Different nonfuzzy graphs can be obtained by thresholding μ_R such that only those edges $\{v, u\}$ for which $\mu_r(v, u) \geq \tau$ are included as edges in G_τ. The graph G_τ is called a *cut graph* of G_R.

Dong et al. [75] present a clustering method based on a connectivity property of fuzzy graphs assuming that the vertices represent a set of objects that is being clustered based on a distance measure. Their algorithm first preclusters the data into subclusters based on the distance measure, after which a fuzzy graph if constructed for each subcluster and a cut graph of the resulting graph is used to define what constitutes a cluster. Dong et al. also discuss the modifications needed in the current clustering upon updates in the database that contains the objects to be clustered.

Fuzzy clustering has not been established as a widely accepted approach for graph clustering, but it offers a more relaxed alternative for applications where assigning each vertex to just one cluster seems restricting while the vertex does relate more strongly to some of the candidate clusters than to others.

3.3. *Representations of clusters for different classes of graphs*

It is common that in applications, the graphs are not just simple, unweighted and undirected. If more than one edge is allowed between two vertices, instead of a binary adjacency matrix it is customary to use a matrix that determines for each pair of vertices *how many edges* they share. Graphs with such edge multiplicities are called *multigraphs*.

Also, should the graph be weighted, cutting an important edge (with a large weight) when separating a cluster is to be punished more heavily than cutting a few unimportant edges (with very small weights). Edge multiplicities can in essence be treated as edge weights, but the situation naturally gets more complicated if the multiple edges themselves have weights.

Luckily, many measures extend rather fluently to incorporate weights or multiplicities. It is especially easy when the possible values are confined to a known range, as this range can be transformed into the interval $[0, 1]$ where one corresponds to a "full" edge, intermediate values to "partial" edges, and zero to there being no edge between two vertices. With such a transformation, we may compute density not by counting edges but *summing over* the edge weights in the unit line: the internal density of a cluster $\mathcal{C}$ (Eq. (22)) on Section 3.2 is rewritten as

$$\delta_{\text{int}}(\mathcal{C}) = \frac{1}{|\mathcal{C}|\,(|\mathcal{C}|-1)} \sum_{\substack{\{v,u\}\in E \\ v\in\mathcal{C}, u\in\mathcal{C}}} \omega(v,u) \tag{25}$$

to account for the degree of "presence" of the edges. Now a cluster of high density has either many edges or important edges, and a low-density cluster has either few or unimportant edges. It may be desirable to do a nonlinear transformation from the original weight set to the unit line to adjust the distribution of edge importance if the clustering results obtained by the linear transformation appear noisy or otherwise unsatisfactory.

3.3.1. *Bipartite graphs*

A *bipartite* graph is a graph where the vertex set V can be split in two sets A and B such that all edges lie between those two sets: if $\{v, w\} \in E$, either $v \in A$ and $w \in B$ or $v \in B$ and $w \in A$. Such graphs are natural for many application areas where the vertices represent two distinct classes of objects, such as customers and products; an edge could signify for example that a certain customer has bought a certain product. Possible clustering tasks could be grouping the customers by the types of products they purchase or grouping products purchased by the same people — the motivation could be targeted marketing, for instance. Carrasco et al. [41] study a graph of advertisers and keywords used in advertisements to identify submarkets by clustering.

A bipartite graph $G = (A \cup B, E)$ with edges crossing only between A and B and not within can be transformed into two graphs G_A and G_B. Consider two vertices v and w in A. As the graph is bipartite, $\Gamma(v) \subseteq B$ as well as $\Gamma(w) \subseteq B$ and these two neighbourhoods may overlap. The more neighbours the two vertices in A share, the more "similar" they are. Hence, we create a graph $G_A = (A, E_A)$ such that

$$\{v, w\} \in E_A \quad \text{if and only if} \quad (\Gamma(v) \cap \Gamma(w)) \neq \emptyset. \tag{26}$$

Similarly a graph G_B results from connecting vertices whose neighbourhoods in A overlap. Weighted versions of G_A and G_B can be obtained by setting

$$\omega(v, w) = |\Gamma(v) \cap \Gamma(w)|, \tag{27}$$

possibly normalizing with the maximum degree of the graph.

Clusterings can be computed either for the original bipartite graph G or for the derived graphs G_A and G_B [41]. An intuition on how this works can be developed by thinking of the set A as books in a bookstore and the set B the customers of the store, the edges connecting a book to a customer if the customer has bought that book. Two customers have a similar taste in books if they have bought many of the same books, and two books appeal to a similar audience if several customers have purchased both. Hence the overlap of the neighbourhoods the one side of the graph reflects the similarity of the vertices of the other side and vice versa.

Bipartite graphs often arise as representations of *hypergraphs*, that is, graphs where the edges are generalized to subsets of V of arbitrary order instead of restricting to pairs of vertices. An example is *factor graphs*. Direct clustering of such hypergraphs is another problem of interest.

3.3.2. *Directed graphs*

Up to now, we have been dealing with undirected graphs. Let us turn into directed graphs, which require special attention, as the connections that are used in defining the clustering are *asymmetrical* and so is the adjacency matrix. This causes relevant changes in the structure of the Laplacian matrix and hence makes spectral analysis more complex.

Web graphs [36] are directed graphs formed by web pages as vertices and hyperlinks as edges. A clustering of a higher-level web graph formed by all Chilean domains was presented by Virtanen [227]. Clustering of web pages can help identify topics and group similar pages. This opens applications in search-engine technology; building artificial clusters is known to be a popular trick among websites of adult content to try to fool the PageRank algorithm [35] used by Google to rate the quality of websites.

The basic PageRank algorithm assigns initially to each web page the same importance value, which is then iteratively distributed uniformly among all of its neighbours. The goal of the PageRank algorithm is to reach a value distribution close to a stationary converged state using relatively few iterations. The amount of "importance" left at each vertex at the end of the iteration becomes the PageRank of that vertex, the higher the better. A large cluster formation of $N \gg 0$ vertices can be constructed to "maintain" the initial values assigned within the cluster without letting it drain out and eventually accumulating it to a single target vertex that would hence receive a value close to N times the initial value assigned for each vertex when the iterative computation of PageRank is stopped, even though the true value after global convergence would be low. Identifying such clusters helps to overcome the problem.

Another example of directed graphs are *web logs* (widely known as blogs) that are web pages in which users can make comments — identifying communities of users regularly commenting on each other is not only a task of clustering a directed graph, but also involves a *temporal* element: the blogs evolve over time and hence the graph is under constant change as new links between existing blogs appear and new blogs are created [157]. The blog graph can be viewed directly as the graph of the links between the blogs or as a bipartite graph of blogs and users. Also the content of the online shared-effort encyclopedia *Wikipedia* forms an interesting directed graph.

4. Measures for identifying clusters

There are two main approaches for identifying a good cluster: one may either compute some values for the vertices and then classify the vertices into clusters based on the values obtained, or compute a fitness measure over the set of possible clusters and then choose among the set of cluster candidates those that optimize the measure used. In this section we first overview vertex similarity measures that can be used in the former manner to identify the cluster of a specific vertex or to group all of the vertices into a set of clusters, and then present possible cluster fitness measures that serve for methods that produce the clustering by comparing different groupings and selecting one that meets or optimizes a certain criterion.

4.1. *Vertex similarity*

There are many clustering algorithms based on *similarities* between the vertices. Should the vertices represent documents, for example, one could compute content-based similarity values for all pairs of documents and use the similarity matrix as a basis for the clustering, attempting to group together vertices that are not only well connected but also similar to each other. The higher the similarity, the stronger the need to cluster the vertices together. Computing such similarities is not necessarily simple, and in some cases evaluating the similarity of two vertices may turn out to be a task even more complex than the clustering of the graph once the similarities are known.

If a similarity measure has been defined for the vertices, the cluster should contain vertices with close-by values and exclude those for which the values differ significantly from the values of the included vertices. If instead of similarity, we use a *distance* measure, the cluster boundary should be located in an area where including more of the outside vertices would drastically increase the intracluster distances (for example, the sum of squares of all-pairs distances). Hence, with distance measures, it is desirable to cluster together vertices that have small distances to each other.

4.1.1. *Distance and similarity measures*

Defining or selecting an appropriate similarity or distance function depends on the task at hand. The number of similarity measures used in the literature has been very high for various decades [122,233]. Given a data set, a distance measure $\mathrm{dist}\,(d_i d_j)$, should fulfil the following criteria:

1. The distance from a datum to itself is zero: $\mathrm{dist}\,(d_i d_i) = 0$.
2. The distances are *symmetrical*: $\mathrm{dist}\,(d_i d_j) = \mathrm{dist}\,(d_j d_i)$.
3. The *triangle inequality* holds:

$$\mathrm{dist}\,(d_i d_j) \leq \mathrm{dist}\,(d_i d_k) + \mathrm{dist}\,(d_k d_j). \tag{28}$$

For points in an n-dimensional Euclidean space, possible distance measures for two data points $d_i = (d_{i,1}, d_{i,2}, \ldots, d_{i,n})$ and $d_j = (d_{j,1}, d_{j,2}, \ldots, d_{j,n})$ include the *Euclidean distance*

$$\mathrm{dist}\,(d_i d_j) = \sum_{k=1}^{n} \sqrt{(d_{i,k} - d_{j,k})^2} \tag{29}$$

which is the L_2 norm, the *Manhattan distance*

$$\mathrm{dist}\,(d_i d_j) = \sum_{k=1}^{n} |d_{i,k} - d_{j,k}| \tag{30}$$

which is the L_1 norm, and the L_∞ norm

$$\mathrm{dist}\,(d_i d_j) = \max_{k \in [1,n]} |d_{i,k} - d_{j,k}|. \tag{31}$$

A typical example of a nonEuclidean space is that formed by vector representations of textual data: for a collection of m text documents $D_1, D_2, \ldots, D_m$, each term t_i that appears in at least one of the documents is represented by a dimension. Denote the number of terms by n — note that typically non-informative words like articles and prepositions are filtered out to reduce the dimensionality. Each document D_j is then represented by a datum d_j such that the element at position i is the *frequency* at which term t_i appears in document D_j, i.e., how many times the term t_i is included in the document. Typically the frequencies are normalized to eliminate the effect of variations in document length. Usually these frequencies are then multiplied by a factor that is inversely proportional to the number of documents in which the term appears, to give more weight to terms that appear in fewer documents. The product measure is called *term-frequency inverse-document-frequency* (tf-idf) and is commonly used in the field of data mining and also well studied [235].

Once the vectors are computed, a similarity measure can be applied. Possibilities include variations of the aforementioned three distances as well as the dot product and/or the angle between the vectors. A common measure that utilized the latter two is the *cosine similarity*, also known as the *Ochini coefficient*: for two vectors $d_i = (d_{i,1}, d_{i,2}, \ldots, d_{i,n})$ and $d_j = (d_{j,1}, d_{j,2}, \ldots, d_{j,n})$, their cosine similarity is the *angle*

$$\rho(d_i, d_j) = \arccos \frac{d_i \cdot d_j}{\sqrt{\sum_{k=1}^{n} (d_{i,k}^2)} \sqrt{\sum_{k=1}^{n} (d_{j,k}^2)}}. \tag{32}$$

As the resulting measure is an angle in $[0, \pi)$, the most dissimilar value is $\pi/2$ and zero is the best possible similarity. An example of using cosine similarity in clustering is the work of Lakroum et al. [159].

Many similarity measures are based on the *Jaccard index* [133], defined for sets A and B as

$$\rho(A, B) = \frac{|A| \cap |B|}{|A| \cup |B|}. \tag{33}$$

This is easily transformed into a distance measure: $\mathrm{dist}\,(A, B) = 1 - \rho(A, B)$. This idea generalized to n-dimensional *binary* vectors $A = (a_1, a_2, \ldots, a_n)$ and $B = (b_1, b_2, \ldots, b_n)$ as follows: denote by $C_{i,j}$ the number of positions $k \in [1, n]$ in which $a_k = i$ and $b_k = j$. The *Jaccard similarity coefficient* for such vectors A and B is

$$\rho(A, B) = \frac{C_{1,1}}{C_{0,1} + C_{1,0} + C_{1,1}} \tag{34}$$

and their *Jaccard distance* is

$$\mathrm{dist}\,(A, B) = \frac{C_{1,0} + C_{0,1}}{C_{0,1} + C_{1,0} + C_{1,1}}. \tag{35}$$

An advantage of the Jaccard index and the derived measures is that they can be applied on *categorical data*, where the data attributes are not numerical but rather represent the presence or absence of a property. An example of the application of the Jaccard coefficient is the work of Dong et al. [75].

The cosine similarity was extended to the coincide with the Jaccard similarity for n-dimensional binary vectors $A = (a_1, a_2, \ldots, a_n)$ and $B = (b_1, b_2, \ldots, b_n)$; the extension is called the *Tanimoto coefficient* [220], defined as

$$\rho(A, B) = \frac{A \cdot B}{\sqrt{\sum_{k=1}^{n} a_1} + \sqrt{\sum_{k=1}^{n} b_1} - A \cdot B}. \tag{36}$$

There exists a distance measure based on the Tanimoto coefficient that fulfils the triangle equality (Eq. 28) [166].

For string data, other typical distance measures include the *edit distance* (also known as the Levenshtein distance [118]), which is the number of character insertions and/or deletions that need to be made in order to transform d_i into d_j, possibly with different costs for different operations. Cohen, Ravikumar, and Fienberg [57] give a survey on string-similarity metrics, and a comprehensive survey to clustering feature vectors (i.e. points in high-dimensional space) is given by Jain et al. [79,134,135].

4.1.2. *Adjacency-based measures*

In some applications, the vertices lack additional properties and there is nothing in the vertices themselves that would allow the computation of a similarity matrix. The situation is not desperate, however, as the edges incident to the vertices can be used to *derive* similarity measures for the vertices either using the adjacency information directly or through some more sophisticated computation. In this section we review vertex-similarity measures based on the structural properties of the graph instead of some application-specific properties imposed on the vertices.

Possibly the most straightforward manner of determining whether two vertices are similar using only the adjacency information is to study the *overlap* of their neighbourhoods in $G = (V, E)$: a straightforward way is to compute the intersection and the union of the two sets,

$$\omega(v, w) = \frac{|\Gamma(v) \cap \Gamma(w)|}{|\Gamma(v) \cup \Gamma(w)|}, \tag{37}$$

arriving at the Jaccard similarity of Eq. (34) on Section 4.1.1. The measure takes values in $[0, 1]$: zero when there are no common neighbours, one when the neighbourhoods are identical.

Another measure is the so-called (Pearson) correlation of the columns (or rows) in a modified adjacency matrix $\mathbf{C} = \mathbf{A}_G + \mathbf{I}$ (the modification simply forces all reflexive edges to be present). The Pearson correlation is defined for two vertices v_i and v_j corresponding to the columns i and j of $\mathbf{C}$ as

$$\frac{n\left(\sum_{k=1}^{n} (c_{i,k} c_{j,k})\right) - \deg(v_i)\deg(v_j)}{\sqrt{\deg(v_i)\deg(v_j)\left(n - \deg(v_i)\right)\left(n - \deg(v_j)\right)}}. \tag{38}$$

This value can then be used as an edge weight $\omega(v_i, v_j)$ to construct a symmetrical similarity matrix. It reaches the value one if and only if the two vertices have the same neighbourhood, and for neighbourhoods with no overlap the value is a negative number no less than -1, depending on the degrees of the vertices. Correlations can also be applied to other measures than the plain adjacency data to determine cluster structure [40].

4.1.3. Connectivity measures

Clusters in graphs can also be defined through *connectivity* by calculating the number of (edge-distinct) paths that exist between each pair of vertices. For some vertices to belong to the same cluster, they should be highly connected to each other [120].

Edachery et al. [82] proposed that in a good cluster, it is not absolutely necessary that two included vertices v and u are connected by a direct edge, if they are at least connected by a short path. Hence a matrix containing the distance for each vertex pair could serve as a similarity matrix, treating two vertices as similar if they have a *low distance*. In their work, Edachery et al. *threshold the path length*: they require that all vertices in a cluster must be at distance at most k from each other. They call such a subgraph a *k-clique* and present a clustering algorithm based on finding *induced subgraphs* that are k-cliques. Note that the requirement that the induced subgraph is to be a k-clique implies that the k-paths (or shortest paths) connecting the cluster members must *only* resort to intracluster edges. The vertex set of a k-clique in a graph does not necessarily form a k-clique induced subgraph — counterexamples are easy to construct.

In order to be able to set the threshold k, one should know (at least an estimate of) the *diameter* of the input graph, which is the maximum distance over all pairs of nodes. Not knowing the diameter, one can hardly set a good value for the threshold, as values close to the diameter may already result in clusters containing large portions of the input graph, and small values of the threshold (with respect to the diameter) may split some natural clusters into two or more k-clique subgraphs.

4.2. Cluster fitness measures

In this section we survey possibilities of *cluster fitness measures*, which are functions that rate the quality of a given cluster or a clustering. Such functions can either be utilized in the identification of clusters (by optimizing one or more cluster fitness measures), choosing between alternative clusterings (by comparing the values of one or more cluster fitness measures), and comparing different clustering algorithms (by studying the values that the clusterings computed give for different cluster fitness measures).

Using cluster fitness functions to determine a clustering for a given input graph is an option to the methods based on classifying vertices by using vertex similarities. The idea is to try to directly identify clusters that fulfil a certain desirable property. Common criteria include several variants of density, i.e. measures that are based on the fraction or number of edges present in the induced subgraph, reaching the maximum value for cliques. For example Hu et al. [132] search for dense subgraphs in biological data by *graph data mining* [230].

4.2.1. Density measures

Several algorithms that search for maximal subgraphs that have a density higher than a preset *threshold* have been proposed [147]. Any definition of a clusters simply as a subgraph that is dense with respect to a given density measure is fundamentally a special case of the following decision problem [210]:

Density

Instance: An undirected graph $G = (V, E)$, a density measure $\delta(\cdot)$ defined over vertex subsets $S \subseteq V$, a positive integer $k \leq n$, and a rational number $\xi \in [0, 1]$.

Question: Is there a subset $S \subseteq V$ such that $|S| = k$ and the density $\delta(S) \geq \xi$?

Note that simple maximization of any density measure without fixing k would result in choosing any clique, including K_2 and K_3, which are neither appealing as clusters; any edge will produce a K_2 subgraph whereas K_3 is a simple triangle. When the density measure used in the above decision problem is that of Eq. (3), the density of the induced subgraph, the problem is **NP**-complete since for $\xi = 1$ it coincides with the **NP**-complete Clique problem [103,143].

The complexity of several variants of this problem are known. Feige et al. [90] consider the problem of computing the subgraph of order k with the *most* edges in a given input graph, which is **NP**-hard as solving it would also solve the maximum clique problem. Feige et al. present an approximation algorithm with approximation ratio $\mathcal{O}(n^d)$, $d < \frac{1}{3}$.

Asahiro et al. [11] study the general question: Given a graph $G = (V, E)$, is there a k-subgraph in G with at least $f(k)$ edges. They find that the problem remains **NP**-complete for $f(k) = \Omega(k^{1+\epsilon})$, $0 < \epsilon < 1$, and for $f(k) = mk^2n^2(1 + \mathcal{O}(n^{\epsilon-1}))$, as well as for $\Omega(n^{\epsilon_1})$-regular graphs if $f(k) = \Omega(k^{1+\epsilon_2})$ for $0 < \epsilon_1, \epsilon_2 < 1$.

Holzapfel et al. [127] pose the same question – whether there is a k-subgraph with average degree at least $f(k)$ – for any density function computable in polynomial time that satisfies $\forall k$ the inequality $f(k) \leq k - 1$. At one extreme they again have the **NP**-complete clique problem, but for $f(k) = 2$ the problem is solvable in polynomial time. They show that the problem is **NP**-complete if $f(k) = 2 + \Omega(k^{-(1-\epsilon)})$, $\epsilon > 0$ and has a polynomial-time algorithm for $f(k) = 2 + \mathcal{O}(k^{-1})$.

Another approach was proposed by Matsuda et al. [171], which consider p-quasi complete subgraphs as clusters; a graph $G = (V, E)$ is *p-quasi complete* for $p \in [0, 1]$, if for all $v \in V$,

$$\deg(v) \geq p(n - 1). \tag{39}$$

They show that it is **NP**-complete to determine whether a given graph has a $\frac{1}{2}$-quasi complete subgraph of order at least k.

In general, for large instances, approximation algorithms are a justified and feasible approach for locating dense subgraphs [171,206]. It is important to keep in mind that for all the above density measures, it is easy to *compute* the measure for a given subgraph: one simply needs the adjacency relation for the included vertices and polynomial-time operations, typically of order $\mathcal{O}(n^2)$.

4.2.2. Cut-based measures

In addition to direct density measures, also measures of connectivity with the rest of the graph are used to identify high-quality clusters. Measures of the "independence" of a subgraph of the rest of the graph have been defined based on cut sizes. Possibly the most important such measure in the context of clustering is *conductance* (see, for example [142]),

defined for any proper nonempty subset $S \subset V$ in graph $G = (V, E)$ as follows:

$$\Phi(S) = \frac{c(S, V\backslash\mathcal{C})}{\min\{\deg(S), \deg(V\backslash S)\}}. \quad (40)$$

Finding a cut with minimum conductance is **NP**-hard [210]. Variants of conductance include *normalized cut* [121,209] and *expansion* [95,142], as well as the *cut ratio*. The problem of optimizing the cut ratio is called the *sparsest cut* problem and it is known to be **NP**-hard [172].

In order to arrive at more independence measures, we define the *internal degree* of a cluster $\mathcal{C}$ to be the number of edges connecting vertices in $\mathcal{C}$ to each other:

$$\deg_{\text{int}}(\mathcal{C}) = |\{\{v, u\} \in E \mid v, u \in \mathcal{C}\}| \quad (41)$$

and the *external degree* of a cluster to be the number of edges that connect it to the rest of the graph:

$$\deg_{\text{ext}}(\mathcal{C}) = |\{\{v, u\} \in E \mid v \in \mathcal{C}, u \in V\backslash\mathcal{C}\}|. \quad (42)$$

Note that the external degree is in fact the size of the cut $(\mathcal{C}, V\backslash\mathcal{C})$. Using these definitions, we arrive at another "independence" measure used in clustering: the *relative density* $\rho(\mathcal{C})$ [178]. Relative density is the ratio of the internal degree to the number of incident edges:

$$\rho(\mathcal{C}) = \frac{\deg_{\text{int}}(\mathcal{C})}{\deg_{\text{int}}(\mathcal{C}) + \deg_{\text{ext}}(\mathcal{C})} = \frac{\sum_{v\in\mathcal{C}} \deg_{\text{int}}(v, \mathcal{C})}{\sum_{v\in\mathcal{C}} \deg_{\text{int}}(v, \mathcal{C}) + 2\deg_{\text{ext}}(v, \mathcal{C})}. \quad (43)$$

For cluster candidates with only one vertex (and any other candidate that is an independent set), we set $\rho(\mathcal{C}) = 0$. Thresholding $\rho(\mathcal{C})$ is **NP**-complete [210].

The computational challenge lies in *identifying* subgraphs within the input graph that reach a certain value of a measure, whether of density or independence, as the number of possible subgraphs is exponential. Consequently, finding the subgraph that *optimizes* the measure (i.e. a subgraph of a given order k that reaches the maximum value of a measure in the graph) is computationally hard. However, as the computation of the measure for a *known* subgraph is polynomial, we may use these measures to *evaluate* whether or not a given subgraph is a good cluster. We will return to this property of easy computation of the these quality measures in Section 6.3.

5. Global methods for graph clustering

This section addresses methods that are designed to obtain a *global* clustering for a given graph. The existing global approaches are capable of dealing with up to a few millions of vertices on sparse graphs [128,185,187]. In a global clustering, each vertex of the input graph is assigned a cluster in the output of the method, whereas in a *local* clustering, the cluster assignments are only done for a certain subset of vertices, commonly only one vertex. Local clustering will be the topic of Section 6. A brief survey on some global clustering methods is given by Newman [184].

When applying graph clustering to a specific application, one needs to choose or design the clustering algorithm to be used within a certain "framework" of defining what is a global clustering of a graph. One decision is whether the clusters $\mathcal{C}_1, \ldots, \mathcal{C}_k$ should form a *partition* such that

$$\mathcal{C}_i \cap \mathcal{C}_j = \emptyset \quad \text{if } i \neq j, \quad (44)$$

or alternatively a *cover* of the data set — the latter option allows for each datum to belong to more than one cluster, but each datum needs to be assigned to *at least one* cluster. In applications, both cases are, however, possible; we will return to the application-specific nature of clustering in Section 8.

5.1. Complexity of global clustering

In this section we discuss some related problems where a dataset – which can be represented as a (weighted) complete graph – is divided into clusters that optimize a certain criteria. Understanding of the approximability and the algorithms for these problems helps to understand how good global clustering algorithms can be.

The *minimum k-clustering problem* is the combinatorial optimization problem where a finite data set D is given together with a distance function $d : D \times D \to \mathbb{N}$, where d satisfies the triangle inequality (Eq. (28)). The task is to partition D into k clusters $C_1, C_2, \ldots, C_k$, where $C_i \cap C_j = \emptyset$ for $i \neq j$, such that the maximum intercluster distance is minimized (i.e. the maximum distance between two points assigned to the same cluster). This problem is approximable within a factor of two, but not approximable within $(2-\epsilon)$ for any $\epsilon > 0$ [112,125].

A related problem is the *minimum k-centre problem*, where a complete graph is given with a distance function $d : V \times V \to \mathbb{N}$ and the goal is to construct a set of *centres* $C \subseteq V$ of fixed order $|C| = k$ such that the maximum distance from a vertex to the nearest centre is minimized. Essentially, this is not a graph problem as the data set is simply a set of datums and their distances: the edges play no role.

If the distance function satisfies the triangle inequality, the minimum k-centre problem can be approximated within a factor of two [125], but it is not approximable within $(2-\epsilon)$ for any $\epsilon > 0$ [131]. Without the triangle inequality, the problem is not in **APX** [126]. A capacitated version, where the triangle inequality does hold but the number of vertices "served" by a single centre vertex is bounded from above by a constant, is approximable within a factor of five [146]: a centre serves a vertex if it is the closest centre to that vertex. Another capacitated version where the maximum distance is bounded by a constant and the task is to choose a minimum-order set of centres [19] is approximable within a factor $\log c + 1$, where c is the capacity of each centre [21]. Agarwal and Procopiuc [4] present an exact and an approximation algorithm for the k-centre problem with extensions to various distance metrics as well as both exact and approximate algorithms for the capacitated version of the problem.

A weighted version of the k-centre problem, where the distance of a vertex to a centre is multiplied by the weight of the vertex and the maximum of this product is to be minimized, is approximable within a factor of two [195], but it can not be approximated within $(2-\epsilon)$ for any $\epsilon > 0$. If it is not the maximum distance that is of interest, but the *sum* of the distances to the nearest centre is minimized instead while

keeping the order of the centre set fixed, the problem is called the *minimum k-median problem*. Feder and Greene [88] show that the problems of minimizing the maximum intracluster distance in general and that of minimizing the distance from each point to its cluster centre cannot be approximated within a factor close to two for points in $\mathbb{R}^d$ if $d \geq 2$, unless $\mathbf{P} = \mathbf{NP}$.

A popular algorithm for clustering vector data with respect to a distance function is the *k-means algorithm* [119]. The basic idea of the k-means method is to cluster a set of points in some metric space into k clusters by iteratively improving k cluster centres and grouping each point to the cluster with the closest centre; the centres are chosen to minimize the *sum-of-squares* of the intracluster distances. Unfortunately, k-means is **NP**-hard even for $k = 2$ [1].

For more variants of such problems and the related complexity results, we refer the reader to the excellent online resource of Crescenzi and Kann [62] with a brief definition, summary of the results and bibliographical references for numerous problems in combinatorial optimization.

5.2. *Iterative or online computation of global clusterings*

Clustering can either be performed to all of the data elements at once, or iteratively, assigning one element at a time to an appropriate cluster. Approaches that require the entire graph to be accessible simultaneously do not scale well for large data sets. In *iterative clustering*, the cluster assignments made to elements upon their first processing may either be considered immutable or may be changed later on to optimize some property of the clustering being computed.

If a clustering algorithm operates *one datum at a time*, having only the knowledge of previously encountered data, it is said to operate *online*. Also methods that process a group of elements at a time are possible. Such *online algorithms* for clustering provide a partial clustering for the data already seen from an unknown data stream to be clustered. They can be designed to dynamically determine the number of clusters to use, often relying on some threshold value to determine when a newly arriving datum needs to be assigned a new cluster instead of merging it to an existing cluster.

In order to successfully cluster a large database, an online clustering method should scan the database at most once. It should also be able to provide some solution, at least a crude approximate, at any time, while incrementally incorporating newly added data into the existing clustering [32]. Such an approach of *incremental* clustering is useful for clustering data sets that undergo frequent modification, such as addition, removal or editing of the data elements [43]. Incremental clustering has been suggested for web page classification by Wong and Fu [236].

Toussaint [222] studied iterative online clustering for points in space by identifying the *nearest neighbour* of the point being clustered among the set of already clustered points: the new arrival is assigned to the same cluster than the neighbour. For graph clustering, the distance measure used should preferably incorporate some structural information on connectivity among the vertices further than the immediate neighbourhood.

It is noteworthy that in online clustering, the order in which the data are processed may significantly affect the resulting clusters: should the algorithm be attempting to construct k clusters, and the data is presented one cluster at a time, there is a risk that the algorithm initially divides the first cluster into several subclusters. If the cluster assignments made by the online algorithm are *immutable*, meaning that once a datum has been assigned to a cluster, it will not be moved to another cluster at any later time, the algorithm can not recover from bad initial cluster assignments. To avoid these problems, commonly the existing partial clustering is constantly optimized with respect to some carefully selected global measure as new data are processed, reassigning also the old data as necessary.

Guha et al. [116] study clustering of points in space where the data arrive one at a time and the goal is to maintain a clustering of the data seen thus far. Additional concerns to the quality of the clustering achieved are the time and memory consumption of such a system — if a new datum may arrive while the previous one is still being clustered, the system needs to maintain a queue buffer for the incoming data and may congest. The approach of Guha et al. [116] is a constant-factor approximation for a k-means algorithm that uses $2k$ medians during the observation phase that are used compress information on the entire dataset. The algorithm has multiple phases and relies on subroutines defined in other work, such as an algorithm by Jain and Vazirani [136] for finding k medians in a data set of order $\mathcal{O}(k)$.

Zanghi, Ambroise and Miele [247] present an online clustering algorithm for graphs that clusters the graph into k clusters, although they ran the algorithm in parallel for various values of $k = 2, 3, \ldots$ and choose the clustering that maximizes the integrated classification likelihood (discussed in Section 7.2). Their approach is based on assuming a certain, relatively high probability for connections within clusters and a smaller one for intercluster connections, similar to that of the planted partition model.

5.3. *Hierarchical clustering*

A global clustering does not have to be a *single* partition or cover, but it may also be defined as a hierarchical structure, where each top-level cluster is composed of subclusters and so forth. This is useful in situations where the graph structure itself is hierarchical, and a single cluster can naturally be composed further to obtain a more fine-grained clustering or alternatively merged with another cluster to obtain a coarser division into clusters.

It depends on the application and the input data whether it makes sense to compute a *hierarchy* of clusterings or a *flat* clustering. In a flat clustering, each cluster is defined by vertex subset $\mathcal{C} \subseteq V$ as the subgraph induced by $\mathcal{C}$, although often the subset itself is called a cluster. In a hierarchical clustering, each *level* of the clustering hierarchy defines a different subset, and usually the clusters defined by the higher levels contain the clusters of the lower levels as subgraphs.

For example, if the number of clusters in which to group the data is known a priori, there is little use in knowing an entire hierarchy and it may be better to resort to flat clustering. However, in many contexts, the hierarchical

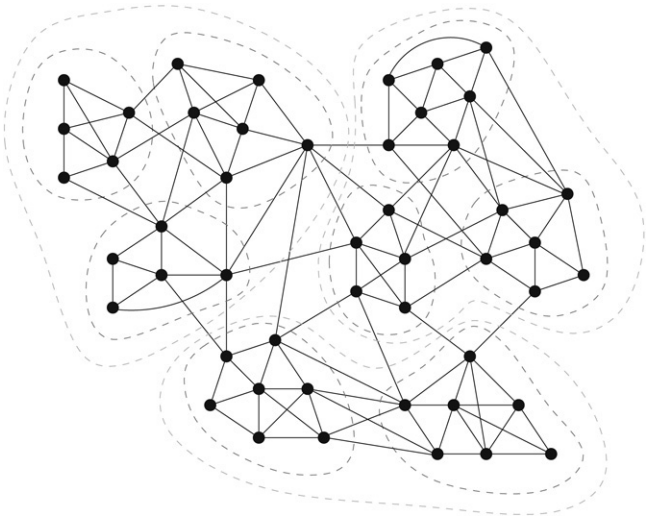


Fig. 4 – A company is divided into three departments, each of which is formed by two or three 5–7 person teams. Each person is represented by a vertex, and an edge is placed between two people if they interact on work-related matters on a daily basis. The teams and the departments are encompassed by dotted lines.

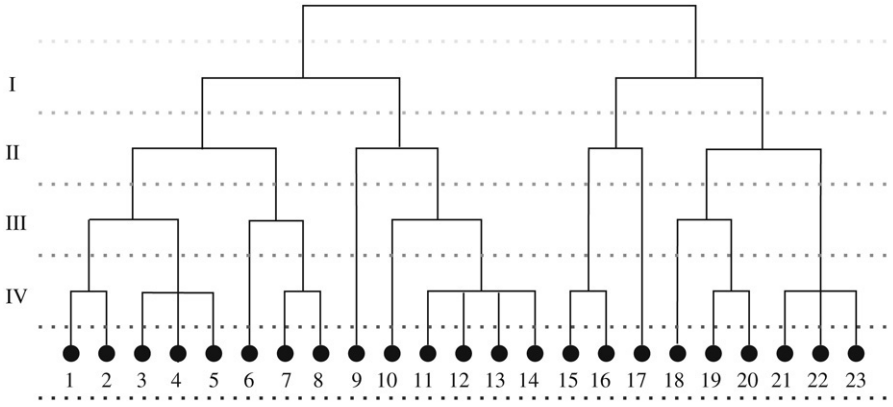


Fig. 5 – An example dendrogram that groups 23 elements into clusters at four intermediate levels, the root cluster containing the entire dataset and the leaf clusters each containing one data point. Any level of the dendrogram, indicated by dotted lines in the picture, can be interpreted as a clustering, grouping together as a cluster those elements that remain in the same branch of the dendrogram tree above the line. In the hierarchy, the cluster [1, 2] is a subcluster of [1, 5] which in turn is a subcluster of [1, 8], and so forth.

structure itself can be of interest. This is the case in the field of social networks: within a city, a workplace will form a cluster and so will a school, but within workplaces and schools, a new level of clusters appears from work teams, classes, etc. See Fig. 4 for a toy example with this kind of a structure. Comparing the existing "official" hierarchy, such as the one defined by department boundaries and team memberships, with a hierarchical clustering of the current person-to-person contact graph may give reorganizational insight and reveal hidden self-organization among the people. One of the seminal studies on social network theory was that of Zachary [244], who predicted a split in a karate club analysing the interactions between the members.

Clustering methods that produce multi-level clusterings are called *hierarchical* clustering algorithms, as opposed to flat clusterings that comprise a single partition or cover. A hierarchical clustering is generally constructed by generating a *sequence* of partitions, where each subcluster belongs to one supercluster in its entity. The *root cluster* contains at most all of the data, and each of the leaf clusters contains at least one data element; semantically relevant clusters usually appear on intermediate levels.

Such a tree is called a *dendrogram*; an example is shown in Fig. 5. If at each iteration, each cluster is split into two, a balanced binary tree of clusters of different levels will result. If the graph has high structural variations, such as large density differences, interpretation of the tree needs some insight: at a certain level of the hierarchy, some natural clusters may already have been split into two or more parts while sets of other natural clusters are still to be identified within the remaining subgraphs.

In practice, a good algorithm for clustering will not only consider the clusterings at certain levels of the tree (drawn in the figure), but also clusterings that result from cutting

different branches of the dendrogram at different levels. Determining where to cut the dendrogram usually involves the optimization of some global quality measure for the entire clustering. Such measures are discussed in Section 7.

Hierarchical clustering algorithms can be further divided into two classes, depending on whether the partition is refined or coarsened during each iteration:

- *top-down* or *divisive* algorithms (addressed in Section 5.4) that split the dataset iteratively or recursively into smaller and smaller clusters [34,37,95,97,107,138,185,186,209], and
- *bottom-up* or *agglomerative* algorithms (addressed in Section 5.5) that start with each data element in its own *singleton cluster* or another set of small initial clusters, iteratively merging these clusters into larger ones [41,74, 78,128].

At each step, the clustering algorithm selects the clusters to merge or split by optimizing a certain criterion on the data set. A *stopping condition* may be imposed on the algorithm to select the best clustering with respect to a quality measure on the current cluster set.

Křivánek and Morávek [158] present results on the complexity of problems related to hierarchical clustering. They formulate the problem as clustering a set of objects with respect to a dissimilarity matrix into a dendrogram. The goal is to find a k-level dendrogram that minimizes a measure that combines the dissimilarity values and the grouping made at each level of the dendrogram for the objects being clustered. In their terminology, level one consists of the singleton sets of the objects and level k consists of the entire set of objects. They show that for $k \geq 3$ the problem is **NP**-hard. If the goal is to find the dendrogram of arbitrary height that minimizes the measure, the problem is **NP**-hard.

5.4. *Divisive global clustering*

Divisive clustering algorithms are a class of hierarchical methods that work top-down, recursively partitioning the graph into clusters. The split at each iteration is typically into two sets, but there is no reason why a clustering algorithm could not divide a vertex set into more than two sets for the next iteration. The various criteria for determining where to split the graph are discussed in this section.

5.4.1. *Cuts*

One intuitive approach is to look for the small cuts (as defined in Section 2.3 in the input graph. Note that the notion of a cut can be naturally defined for directed and undirected graphs as well as weighted or unweighted. The minimum cut in a given (weighted) graph can be found efficiently with a maximum-flow algorithm [60,83,96].

We wish to split the graph in two by removing a cut. Remembering that we want the clusters to be dense subsets with respect to the global density of the graph, a well-chosen cut should separate two or more clusters, instead of breaking into two the vertex set of any single cluster. There are two complications with this idea, however. Firstly, we would like to be able to make some statements regarding the *relative order* of the subgraphs separated by the cut: just cutting out single vertices does not help much in computing a clustering, as removing vertices one by one results in singleton clusters and does not reveal any higher-level structural properties.

However, posing restrictions on the orders of the resulting subgraphs makes the complexity of the problem harder. For example, requiring the two "sides" of the split in the graph to have the same order, results in an **NP**-hard problem: *Minimum bisection* is the problem of dividing a $2n$-vertex graph into two n-vertex subgraphs such that the cut size is minimized [104]. Deciding whether such a cut exists remains **NP**-complete even for regular graphs [37] and for graphs with bounded maximum degree [170], but the problem is polynomial for trees [170] and graphs with bounded treewidth [212]. Feige and Krauthgamer [89] provide approximations for the minimum bisection problem. For a survey on related problems, called *graph layout* problems, see Díaz, Petit, and Serna [71].

As the graph bisection is **NP**-hard, also ℓ-partition where the graph is to be partitioned to ℓ equal-sized groups such that the grouping minimizes the total number of edges crossing from one group to another is also **NP**-hard. Johnson et al. [138] discuss efficient strategies for solving min-cut clustering problems from an integer programming viewpoint.

Condon and Karp [59] present an ℓ-bisection algorithm that finds in linear time the optimal partition with probability $1 - \exp(-n^{\Theta(\epsilon)})$ under the planted ℓ-partition model with $p \geq r + n - \frac{1}{2} + \epsilon$ for constant ϵ. Their algorithm greedily classifies the vertices into two groups, L_1 and R_2, minimizing the total number of edges crossing the various cuts. The processing order of the vertices is based on randomly and uniformly sampling vertex pairs among the unprocessed vertices. This division is then recursively applied to the sets L_1 and R_1 to create a second-level division into four groups, and further until the desired group-size ℓ has been reached. Note that the algorithm will fail for some combinations of ℓ and n. They also present a nonrecursive version. Dubhashi, Laura and Panconesi [80] further develop the approach of Condon and Karp to cluster categorical data rather than graphs.

The second complication with cut-based methods is shared by most hierarchical divisive algorithms; one needs to know when to stop splitting the graph. Setting limits on the cluster order or the number of clusters can be feasible in the presence of a priori information on how the clustering should be like. Another approach is to optimize some cluster quality index; such issues are discussed in Section 7.

Hartuv and Shamir [120] propose a divisive clustering algorithm that uses a density-based stopping condition. Their intuition is that for some vertices to belong to the same cluster, they should be highly connected to each other, whereas there should not be many paths connecting them to vertices outside the cluster. The splitting of the graph is done by removing from the graph at each iteration the edges that cross the current minimum cut. For each connected component, they check whether the component is highly connected. If it is, it will not be divided further. If it is not highly connected, the iteration continues with the removal of the edges crossing the minimum cut. Their definition for a highly connected graph is that the edge-connectivity of the graph of order n is above $\frac{n}{2}$.

Rather than simple cut size, a popular criterion for partitioning is that of low conductance [41,209,210], believed

to be in general superior to simple minimum cuts when used in graph clustering [95,142]. One reason for such preference is that conductance also takes into account the orders of the sets that are being cut apart, yielding often in more significant separations. Although finding a cut with minimum conductance is **NP**-hard [210], several clustering algorithms have been proposed based on variants of conductance [142, 121,95,34,48,108], generally iteratively finding a cut with low conductance or normalized cut size and splitting the graph.

The general idea is that conductance and other similar measures tend to reach optimal values at cluster boundaries and not within clusters, as each cluster should be internally dense while being sparsely connected to the rest of the graph. Hence the division should not likely break a natural cluster but rather separate clusters from other clusters. Care must be taken to define the stopping condition in order to recognize when the subgraph that is being cut contains only one cluster and should not be cut further. Usually the minimum conductance of a single-cluster graph is much higher than the conductance of a multicluster one, but the magnitude of the difference depends on the graph structure. No global threshold or percentage for the relative increase can be given that would correctly stop the clustering for all graphs at the most natural cluster division.

The workarounds to solving an **NP**-complete problem are various; Johnson et al. [138], for example, choose a cut with conductance almost as low as the minimum over all cuts (implementational details by Cheng et al. [47]), whereas Carrasco et al. [41] use an exact, fast algorithm of Lang and Rao [160] for finding a cut S' that is better than S such that $S' \subset S$ (i.e. improving the cut only by removing vertices from the selected subset). Matula and Shahrokhi [172] present an efficient method for finding sparsest cuts for a broad class of graphs. Arora et al. [10] give a $\mathcal{O}(\sqrt{\log n})$-approximation algorithm for sparsest cut and conductance.

Shi and Malik [209] obtain a clustering (which in their application is actually a segmentation of an image) by computing the eigenvector associated with the second-smallest eigenvalue of a Laplacian matrix that incorporates edge weights. Using the components of this eigenvector as vertex weights, they search for the smallest normalized cut. If the value of normalized cut is below a predetermined threshold, the graph is partitioned in two using the cut set that gives the minimum normalized cut. If a partitioning is made, the above process is repeated for the two subgraphs thus created.

He et al. [121] study the normalized-cut method of Shi and Malik [209] and discuss its connections to the k-means algorithm. He et al. restrict to the case of binary edge weights in the case of the k-means method. Applying matrix algebra, they are able to show that what is being optimized in the normalized-cut method is actually the same that is optimized the iteration of the k-means method, with slight modifications on how neighbouring vertices are weighted. Another method based on matrix algebra is that of Drineas et al. [1] who present an clustering algorithm for large graphs that is based on computing the *singular value decomposition* of a suitably selected random submatrix.

5.4.2. *Maximum flow*

As the well-known connection with maximum-flow and minimum-cut problems suggests (see for example [60,83,96]), there exist clustering algorithms based on flow computations [34,41,94,160]. The algorithms for computing maximum flows in graphs (such as that of Goldberg and Tarjan [111]) are efficient and hence such operations are not too costly to be used as subroutines for clustering moderate-size instances.

Most flow-based methods take edge weights readily into account; flow computations extend even to cases where the edge capacities are single-parameter functions instead of constants [102]. Flake et al. [95] identify clusters by inserting an artificial sink and calculating flows to that sink. The minimum cuts that correspond to the maximum flows are used to build a *minimum-cut tree*, as defined by Gomory and Hu [8]. The minimum-cut tree is then used to define what is the cluster of a given vertex v with respect to the artificial sink vertex. The algorithm is designed for undirected weighted graphs, and the weight assigned to all of the edges of the artificial sink vertex is a parameter of the method. It requires some intuition to choose a good value for the parameter. The basic version of the algorithm simply treats all connected components of the min-cut tree after the artificial sink has been removed as clusters, but the authors also present a recursive version that incorporates adjusting the parameter and imposing quality criteria such as desirable cluster order and cluster count.

5.4.3. *Spectral methods*

When a graph is formed by a collection of k disjoint cliques, the normalized Laplacian (Eq. (14)) is a block-diagonal matrix that has eigenvalue zero with multiplicity k and the corresponding eigenvectors serve as indicator functions of membership in the corresponding cliques: the elements of the clique have a different value (of larger magnitude) than the other vertices. Any deviations caused by introducing edges between the cliques causes $k-1$ of the k eigenvalues that were zero to become slightly larger than zero and also the corresponding eigenvectors change. However, some of the underlying structure can still be seen in the eigenvectors of the Laplacian even when edges are added to connect the cliques and when some edges are removed from within the original cliques.

This phenomenon is the basis of *spectral clustering*, where an eigenvector or a combination of several eigenvectors is used a vertex similarity measure for computing the clusters. For example clustering and other analysis of the network of the Internet autonomous-system domains has been done with spectral methods [109,178]. A comprehensive introduction to the mathematics involved in spectral graph theory is the textbook of Chung [51]; we also recommend the textbook of Biggs [27]. The dissertation of McSherry [174] provides an overview of the area, also applying it to graph partitioning.

Also other matrices than the Laplacian can be used to compute such spectral measures; if instead of the adjacency matrix of a simple graph, the input is some kind of a similarity matrix for a complete graph, similar computations still may yield good results. The downside is that computing or even approximating eigenvalues and eigenvectors is not fast for all

graphs and hence such methods may face scalability issues when applied to massive graphs.

Spectral clustering is typically based on computing the eigenvectors corresponding to the *second-smallest* eigenvalue of the normalized Laplacian or some eigenvector of some other matrix representing the graph structure. Possible matrices include modifications of the adjacency matrix such as the transition matrix of a blind random walk on the graph. The component values of the resulting eigenvector are used as vertex-similarity values to determine the clustering. A recent survey of properties of the second-smallest eigenvalue is given by de Abreu [70].

Spectral methods are in general computationally demanding, although a distributed algorithm for decentralizing the computational load has been proposed by Kempe and McSherry [144]. The idea of the decentralization is that the computation of the eigenvector elements of each vertex for the first k eigenvectors can be computed by a separate processor, assuming that all processors are aware of the weights of the incident edges and messages may be exchanged between the processors that are in charge of neighbouring vertices. The number of messages needed is $\mathcal{O}(k^3)$ per round of computation and the number of rounds needed in order to obtain a deviation $\epsilon > 0$ from the exact eigenvectors is $\mathcal{O}(\log^2(n\epsilon^{-1})T)$, where T is the mixing time of the random walk on the input graph. The basis of the decentralized algorithm is the orthogonal iteration method, where random initial vectors are iteratively first multiplied by the adjacency matrix and then orthonormalized. The decentralization is in effect decentralizing the matrix multiplication and the orthonormalization.

How well spectral measures work as separators in clustering is studied by Guattery and Miller [115]. They concentrate on the problem of finding a separator that divides the vertices of the graph into two sets such that the number of edges crossing the boundary is small (if not minimal) using the information contained in the Fiedler vector or possibly combining information from several eigenvectors. Guattery and Miller find that using the Fiedler vector to partition a graph into two equal-sized vertex sets work poorly for a family of bounded-degree planar graphs and in that there exists a family of graphs for which spectral methods in general work poorly.

Qiu and Hancock [198] present a spectral method for clustering graphs based on the Fiedler vector of the graph. Bach and Jordan [15] address the question of clustering a complete weighted graph with a spectral method, also discussing approximations for the eigenvectors. Ng et al. [188] complain that there are several spectral clustering algorithms that all work a little differently with respect to utilizing the eigenvectors and that commonly no proofs are presented regarding the quality of the produced clustering. Although their work addresses the field of clustering points in space, the situation is similar in spectral graph clustering — there is no one canonical way to utilize spectral methods, and even the matrix the spectrum of which is used is not always the same.

A spectral clustering method for directed weighted graphs is given by Capoccia et al. [40]. Their idea is to compute eigenvectors and use the correlations between the elements to determine the cluster structure. Kannan et al. [142] show that in general, spectral methods (or actually, one common variant) find good clusterings. Ding and He [72] propose a spectral method that directly computes k clusters in a complete weighted graph. The problem of determining what value of k to choose – a requisite for the employment of various clustering algorithms – is discussed in Section 7.2.

Goh, Kahng and Kim [110] have studied the spectrum created by the Barabási–Albert generation method for scale-free graphs with two outgoing edges per added vertex. For their studies, they computed the exact spectrum for graphs up to 5000 vertices and determined the first few of the largest eigenvalues for graphs of order as high as 400,000, which gives a hint on the scalability of spectral methods, although the techniques they used were not the most modern. Theoretical results on the spectra of graphs with defined degree distributions are also available [53]. Saerens et al. [204] discuss the relation between *principal components analysis* of graphs and spectral clustering.

A graph partition into two sets with few edges between the sets using the magnitudes (or signs) of the components of an eigenvector (or a combination of eigenvectors) is called a *spectral bisection* [115]. Spectral measures perform well on such 2-classification tasks [72]. When three clusters are present, spectral information groups two of these together in the sense that the separation between these two and the third one is clear and easily interpreted from the second (and third) eigenvector, but the other two are harder to distinguish [123].

Intuition on how the two-classification works is relatively easy to gain through the behaviour of the Rayleigh quotient (as in Eq. (18)) when the function $f(v)$ is interpreted as an indicator vector: a positive value indicates that the vertex belongs to a cluster $\mathcal{C}_A$ and a negative value that it belongs to a cluster $\mathcal{C}_B$. As the edges should be mostly internal to the clusters, almost all differences in the sum of the Rayleigh quotient are zero. Only the edges connecting vertices in $\mathcal{C}_A$ with those of $\mathcal{C}_B$ contribute to the sum.

If we normalize the vector represented by $f()$ for example to have norm n, and consider a vector *other* than the vector of all ones that minimizes the Rayleigh quotient. If such a vector has both positive and negative values, the positive ones get assigned to one of the classes and the negative ones to the other class in order to minimize the ratio. There will be only one "gap" in the values of $f()$ and that gap will show the class boundary.

If we, however, wish to perform classification into more than two classes, some number of classes get assigned negative values and the rest positive ones, and the "gaps" of the values assigned to each class will vary, making it much harder to automatically determine the division. The remedy for the multicluster problem is to perform the two-classification iteratively, using the spectra of the resulting induced subgraphs. This will yield a divisive hierarchical clustering algorithm. Spielman and Teng [213] show that such partitioning performs well on bounded-degree planar graphs and finite element meshes. Their analysis is based on a relation of the cut size with the second-smallest eigenvalue of the Laplacian. By showing that for d-dimensional well-shaped meshes, this eigenvalue is $\mathcal{O}(n^{-2d^{-1}})$, they can show that spectral methods can be applied to identifying cuts of size $\mathcal{O}(n^{(d-1)d^{-1}})$. Pothen et al. [196] present a heuristic

algorithm for minimum bisection using the eigenvectors of the Laplacian matrix.

Brandes et al. [34] propose a clustering algorithm that first computes edge weights using k distinct eigenvectors of the normalized adjacency matrix, associated with the largest eigenvalues less than one. Using these weights, a minimum spanning tree is computed. Partitioning the spanning tree by removing all edges with weight below a certain threshold and considering each connected component of the resulting forest to be a cluster defines a clustering, and for different values of the threshold, different clusterings are obtained. A dendrogram can be constructed removing the edges one by one starting from the weakest edge.

5.4.4. *Betweenness*

In order to cluster an unweighted graph $G = (V, E)$, Newman and Girvan [186] impose weights on the edges based on structural properties of the graph G. The idea is based on that of node-betweenness defined by Freeman [100] for use in sociological studies. The weight used by Newman and Girvan [186] is the *betweenness* of an edge $\{v, w\}$, which is the number of shortest paths connecting any pair of vertices that pass through the edge. Freeman in turn studied node-betweenness, which is defined for each *vertex* as the number of shortest paths in the graph that pass through that vertex. As a computational detail, one should note that there may exist multiple paths of the same length between a given pair of vertices. Hence each of these shortest paths should be accounted for in proportion to their number when computing the betweenness values of the edges that these paths use. If there are k shortest paths connecting v and u, each of them will have weight $\frac{1}{k}$ in betweenness calculations of the edges on those paths.

Girvan and Newman [107,186] assume edges with high betweenness to be links between clusters instead of internal links within a cluster: the several shortest paths passing through these edges are the shortest paths connecting the members of one cluster to those of another. Hence they split the network into clusters by removing one by one edges with high betweenness values. If more than one edge has the highest betweenness value, one of them is chosen randomly and removed. The removal is followed by recalculation of the betweenness values, as the shortest paths have possibly been altered. This gives a clustering algorithm polynomial in n and m.

Straightforward algorithms compute the betweenness on an edge operate in $\mathcal{O}(n \cdot m)$ time. Brandes [33] proposes algorithms for betweenness-centrality computations that require $\mathcal{O}(n + m)$ space. The running time for his unweighted version is $\mathcal{O}(nm)$ and the weighted version runs in $\mathcal{O}(nm + n^2 \log n)$ time. He solves for each vertex once a single-source shortest path problem, with small modifications to either Dijkstra's algorithm or the breadth-first search algorithm (see for example [60] for shortest-path algorithms). Newman proposes a method based on random walks rather than exact betweenness-value computation [181]. Comellas and Gago Álvarez [58] derive bounds on the betweenness values using the spectrum of the graph.

Fortunato, Latora and Marchiori [97] propose a hierarchical method closely related to the betweenness-method of Newman and Girvan, but instead of the betweenness values, they use *information centrality* [162] that is defined for each edge as the relative decrease in the *average efficiency* of the graph upon the removal of the edge. Latora and Marchiori [161] define efficiency of a pair of distinct vertices $v, u \in V$ as the inverse of their distance in the graph, $1/\operatorname{dist}(v, u)$, and the *average efficiency* of the graph G is defined as the average of the individual efficiencies over all $n(n-1)$ ordered pairs of distinct vertices.

The trick with these hierarchical, edge-removal-based clustering methods is that to decide when to stop the partitioning, just as was the case with cut-removal methods. Newman [185] proposes computing a quality measure called *modularity* over the entire clustering at each iteration and stopping when there is no improvement. Many formulations of essentially the same measure exist, depending on whether the graph is weighted and whether minimization or maximization is used. Modularity is in general defined for weighted graphs, where the weights represent some application-specific attributes. For unweighted graphs, one can simply set $\omega(v, w) = 1$ for all edges to obtain a working definition of modularity.

In terms of the edge weights, modularity $\mathcal{M}(\mathcal{C}_1, \ldots, \mathcal{C}_k)$ is defined over a specific clustering into k known clusters $\mathcal{C}_1, \ldots, \mathcal{C}_k$ as

$$\mathcal{M}(\mathcal{C}_1, \ldots, \mathcal{C}_k) = \sum_{i=1}^{k} \mathcal{E}_{i,i} - \sum_{\substack{i \neq j \\ i,j \in \{1,\ldots,k\}}} \mathcal{E}_{i,j}, \tag{45}$$

where

$$\mathcal{E}_{i,j} = \sum_{\substack{\{v,u\} \in E \\ v \in \mathcal{C}_i,\, u \in \mathcal{C}_j}} \omega(v, u), \tag{46}$$

with each edge $\{v, u\} \in E$ included at most once in the computation. Defining the internal and external degrees in terms of these modularity measures gives

$$\deg_{\text{int}}(\mathcal{C}_i) = \mathcal{E}_{i,i} \quad \text{and}$$

$$\deg_{\text{ext}}(\mathcal{C}_i) = -\mathcal{E}_{i,i} + \sum_{j=1}^{k} \mathcal{E}_{i,j}. \tag{47}$$

Newman [180] also presents a formulation of modularity in matrix form, using the spectrum of the $k \times k$ modularity matrix, where the elements are the values $\mathcal{E}_{i,j}$. Newman also uses the spectral information to derive centrality measures for the vertices of the input graph.

Note that as modularity directly incorporates the *number* of internal edges per cluster, the orders of the clusters tend to have an effect: small clusters in simple graphs may only contribute a few internal edges. Danon et al. [66] provide a modification of a modularity-based algorithm of Newman [185] to accommodate for clusters of varying orders without slowing down the computation.

5.4.5. *Voltage and potential*

Electrical circuits provide reasonable intuition for graph clustering: think of the graph as a circuit that has a *unit resistor* on each edge. Calculate the potentials at all of the vertices (i.e. the voltages for all the edges), and then cluster

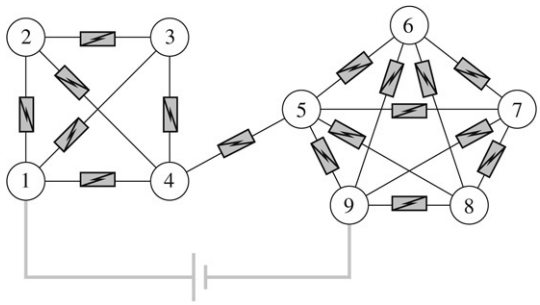


Fig. 6 – A graph transformed into an electric circuit: a unit resistor has been placed on each edge, and a battery connects the vertices labelled one and nine (battery connection drawn in grey).

the vertices based on the potential differences [186,187,238]. However, in order to have a current in the circuit, one must introduce a battery. The problem is the placement of the battery: how to choose the source and the sink of the current? In Fig. 6, there are two natural clusters, a clique of four vertices and another clique of five vertices, the two being connected by a single edge. If we connect the battery between vertex number one in the 4-clique and vertex number nine in the 5-clique, the resulting potentials are higher for the 4-clique than for the 5-clique and we can divide the graph into clusters based on the vertex potentials only.

The potential vectors for graphs can be easily computed with circuit analysis tools like SPICE [197,199]. If vertices with labels 1 and 2 share an edge, one creates a corresponding unit resistor for example by assigning a name to the resistor, such as `r12`. Each edge corresponds to such a resistor-definition line, and a voltage source (a one-volt battery) is placed between a source vertex (in Fig. 6, the vertex labelled 1) and a sink vertex (vertex labelled 9). The simulator then calculates and reports the voltage for each resistor, among other things.

The input given to SPICE corresponding to Fig. 6 and the output of the transient analysis yielding the vertex potentials are shown in Fig. 7.

However, when the cluster structure is not known a priori, we need to place the batter between random vertices, which makes the interpretation of the voltage vector much harder, as we might have chosen near-by vertices and the number of natural clusters present in the graph is not known. Repeatedly choosing random source and sink vertices and averaging over the resulting voltages would give values of at least some use, but in a highly nonuniform network, the battery placement may have a significant effect and the variances could be disturbingly large.

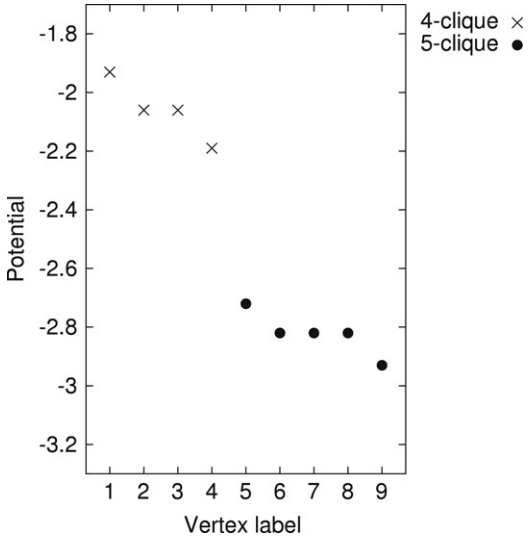
Wu and Huberman [238] streamline the procedure by fixing the source vertex at a high potential, and choose a *random sink* vertex to be at low potential. The voltages of the vertices "close" to the seed vertex will be higher than the voltages of those that are far and hence the voltage difference of a vertex pair can be used as a similarity measure. Their algorithm obtains an approximate solution to the Kirchhoff equations by iterative computation, starting with all but the seed and sink vertices at potential $\frac{1}{2}$, the seed at 1, and the sink at 0.

A divisive global clustering algorithm based on voltage computations would pick a seed vertex and a "far away" sink vertex, classifying the remaining vertices into those that have potentials closer to the sink potential and to those with potentials closer to the source potential of the seed vertex. Based on this classification, the graph could then be split. Continuing such a procedure in an iterative fashion would then yield the hierarchical clustering.

In order to *visualize* the cluster structure resulting from such computations, we use gray-scale similarity colour maps. We iterate over the entire vertex set, fixing the vertices one at a time to be the source vertex $s \in V$, also called the *seed* vertex of the computation. This way we obtain a n-element vector

```
V1 1 9 1
r12 1 2 1kOhm
r13 1 3 1kOhm
r14 1 4 1kOhm
r23 2 3 1kOhm
r24 2 4 1kOhm
r34 3 4 1kOhm
r45 4 5 1kOhm
r56 5 6 1kOhm
r57 5 7 1kOhm
r58 5 8 1kOhm
r59 5 9 1kOhm
r67 6 7 1kOhm
r68 6 8 1kOhm
r69 6 9 1kOhm
r78 7 8 1kOhm
r79 7 9 1kOhm
r89 8 9 1kOhm
```

(a) Input file.



(b) Vertex potentials computed by SPICE.

Fig. 7 – On the left, the definition of the circuit of Fig. 6 in SPICE syntax, given as input to the DMCS-SPICE Java Applet [218]. The transient analysis, tracing the voltages at each vertex, gives the figure on the right, where the two clusters are clearly separated. The voltage vector is (−1.93, −2.06, −2.06, −2.19, −2.72, −2.82, −2.82, −2.82, −2.93).

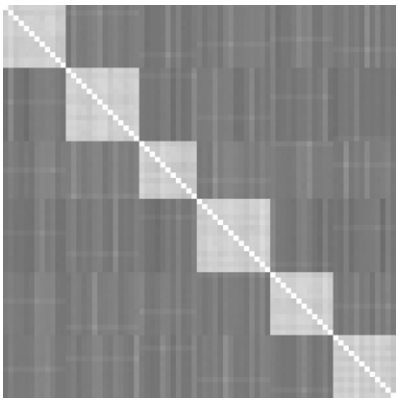


Fig. 8 – Visualization of the potential vectors used in the clustering algorithm by Wu and Huberman [238] averaged over 500 random sink–source pairs. The graph is a relaxed caveman graph with 75 vertices and 900 edges forming a six-cluster structure. In the matrix, the vertices are sorted by clusters. Large values (white being the maximum) indicate near-by vertices. The clusters are visible and easily identifiable as the light squares on the diagonal.

that reflects the cluster structure *viewed from the perspective* of the vertex s. (We return to the question of clustering with respect to a seed vertex in Section 6.)

We then compose a matrix by combining the n vectors, using each vector as a column of the matrix, thus obtaining an $n \times n$ matrix. The crucial part is the *ordering* of the vertices, both in the vectors and in the matrix, using the same order in both. The order should be based on the clustering obtained, such that first come all vertices in $\mathcal{C}_1$, followed by all vertices in $\mathcal{C}_2$, and so forth.

In the case of the Wu–Huberman algorithm, the elements of the vectors are voltages. Similar voltages should be assigned to the members of a single cluster. As viewing the numbers themselves can be tedious, we transform them into grey-scale colours. We first scale the element values to the unit line, with the maximum value at one and the minimum at zero. Then we discretize it to 256 levels, assigning the value zero to black and the value one to white. We draw a rectangle, colouring the position corresponding to the element at column i and row j with the colour that corresponds to the value that the ith vector assigns to the jth vertex.

For a successful clustering, a chain of visible blocks appears on the diagonal of the colour map, assuming that the matrix construction was ordered according to the clustering suggested by its values. The less evident these diagonal blocks are, the less obvious is the cluster structure that the similarity vectors imply. An example similarity matrix obtained by the Wu–Huberman algorithm is shown on the left in Fig. 8.

The problem of choosing a random sink can be avoided by considering diffusion in an *unbounded* medium instead of a finite electrical circuit. Orponen and Schaeffer [191] compute vertex-similarities as the solution to the *discrete Dirichlet problem* [163,154]. The discrete Dirichlet problem, in the context of graphs, is the problem of assigning values to vertices such that the difference between the value of a vertex and the average of the values of all its neighbours is zero, requiring that a set of *border vertices* hold certain fixed values. The assignment of these values corresponds to the unique harmonic function of the graph. Orponen and Schaeffer [191] fix only one boundary vertex: the seed vertex s. The value-assignment vector needed is in fact equal to the Fiedler vector of the graph with respect to the seed vertex [51].

The Fiedler vector can be computed by minimizing the degree-adjusted Rayleigh quotient (see Eq. (17) on Section 2.3) which in turn can be approximated iteratively with local computation [191]. Fig. 9 shows similarity matrices constructed by computing the Fiedler vectors for each vertex in a small example graph; the same graph was used in the examples of Fig. 8. A pleasant surprise is that by approximating the vector, mainly "noise" gets eliminated and the cluster structure is even more evident. A related method for global clustering with similar computations is proposed by Ding and He [72]. Also, a clustering method based on Fiedler vectors for document clustering of the World-Wide Web is reported by He et al. [121].

For more intuition on the relationship of electrical networks to graph structure, we recommend the book of Doyle and Snell [77] that studies the connection between electrical networks and Markov chains.

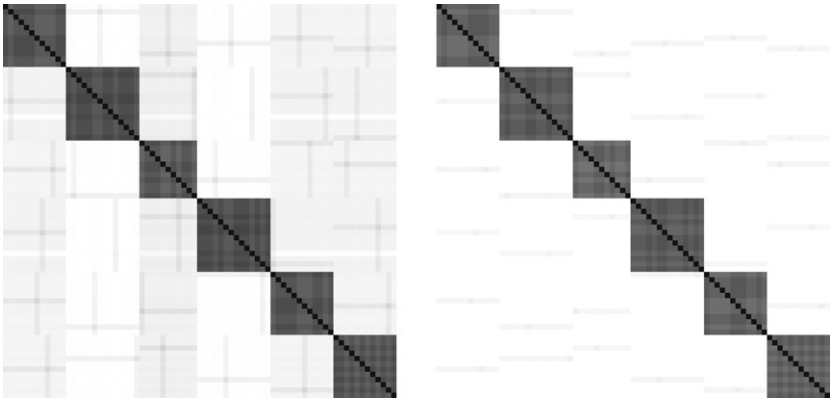


Fig. 9 – Similarity matrices for a 75-vertex graph with 900 edges, composed of six clusters constructed by computing the Fiedler vector for each vertex. On the left, exact vectors were used, and on the right, a locally computable approximation of Orponen and Schaeffer [191] for the Fiedler vector was used.

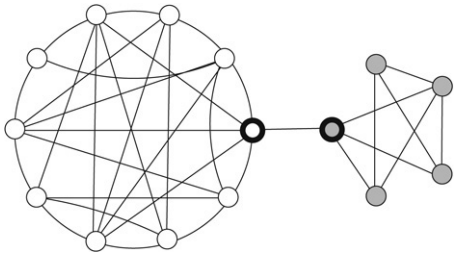


Fig. 10 – A two-cluster graph. If a blind random walk is currently in the white thick-bordered vertex, it will remain in the white cluster with probability $\frac{5}{6}$ and switch over to the grey cluster with probability $\frac{1}{6}$. Being at any other white vertex, a random walk will necessarily stay within the white cluster. Similarly, a blind random walk on the grey thick-bordered vertex has probability $\frac{1}{4}$ to switch over to the white cluster, and probability $\frac{3}{4}$ to remain in the grey cluster.

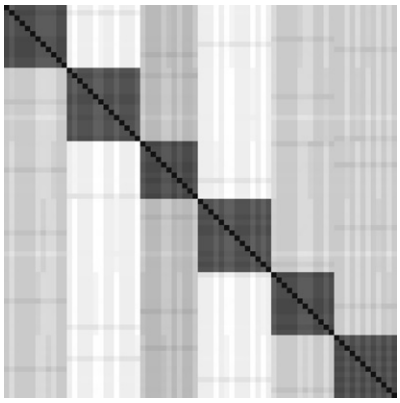


Fig. 11 – Visualizations of the absorption-time matrix for a 75-vertex graph with 900 edges and a clear six-cluster structure, vertices sorted by clusters. The transition matrix is that of a blind random walk and small values (black is zero) indicate nearby vertices. The clusters are visible and easily identifiable as the black squares on the diagonal.

5.4.6. *Markov chains and random walks*

The components of the eigenvector corresponding to the second eigenvalue of the transition matrix of a random walk on a graph (see Section 2.4) serve as "proximity" measures for how long it takes for the walk to reach each vertex. Two vertices in the same cluster should be "quickly reachable" from each other, i.e. have small absorption time from one to the other. Also, if a random walk visits a vertex of a certain cluster, it should be likely to visit several of the other cluster members before leaving the cluster [224]. For example, in Fig. 10, if a random walk is currently in one of the two clusters, it is more likely to remain in that cluster than to move over to the other cluster.

We can also obtain another similarity matrix for the vertices by computing n absorption-time vectors, one for each vertex v_i, appending the zero element at position i of the resulting vector, and then using these vectors as the columns of a matrix, ordering vertices $v_1, v_2, \ldots, v_n$ both inside the vectors and when arranging them into a matrix. Such a similarity matrix can be used to cluster the graph: Fig. 11 shows an example of a similarity matrix of absorption times for the same graph that was used in Figs. 8 and 9. Note that the notion of absorption times directly extends to directed graphs.

Again, absorption times to a seed vertex can be used to classify the vertex set of the graph to "nearby" vertices and to "far-away" vertices and the classification can then be used to split the graph. Iterating this one obtains a cluster hierarchy. An algorithm for approximate graph partitioning based on approximating the distributions of several random walks on the input graph is given by Spielman and Teng [214]. The thesis of van Dongen [224] discusses graph clustering using Markov chains. His approach is to apply a sequence algebraic matrix operations on the Markov chain corresponding to the input graph such that performing the operations will eliminate intercluster interactions and only leave the intracluster parts. The "filtered" information can then be used to identify the clusters by permuting the matrix to diagonal form to reveal the structure.

Meilă and Shi [175,176] show that the normalized cut of a graph, one of the variants of graph conductance, can be expressed in terms of the transition probabilities and the stationary distribution of a random walk in the graph, thus linking the mathematics of random walks to those of cut-based clustering. Orponen and Schaeffer [190] in turn express the absorption times of a random walk in a graph in terms of the eigenvectors of the graph's Laplacian and use their locally computable approximation of the Fiedler vector [191] to obtain an approximation of the absorption times. This links the random walks to spectral clustering, which relates to cut-based methods.

5.4.7. *Other divisive methods*

Auber et al. [12] provide a rather mathematical approach for graph clustering. They assume that some similarity metric has been defined over the vertex set of the graph. They define a discrete density function for this metric to "group" the similarity values into a discrete set. The histogram of the density function is then filtered by convolution with a carefully chosen kernel function. The output of the filtering is used to split the graph in two. This is then continued iteratively to obtain a cluster hierarchy. There are some parameters the selection of which affects the resulting cluster hierarchy. The mathematics involved are not quite recommendable for novice practitioners. Auber et al., however, offer a publicly available implementation of the method in a visualization tool.

A graph-clustering algorithm based on random walks, specifically k-step transition probabilities between a sink vertex and other vertices, is presented by Yang and Liu [242]. They sort the adjacency matrix with respect to the ordered k-step transition probabilities and then find a cut position in the matrix such that the resulting division yields a good diagonal matrix. This cut is used to split the graph into two clusters, and the procedure is then repeated for the diagonal submatrices of the adjacency matrix.

5.5. *Agglomerative global clustering*

Naturally, in addition to dividing the graph top-down into clusters, one may also work bottom-up merging singleton sets of vertices iteratively into clusters. Such methods are called *agglomerative* clustering algorithms. Typically a similarity measure is used to the vertices to be merged into a cluster. The similarity measure could be based, for example, on the relative overlap of the vertex neighbourhoods or some semantic-based value. Possible similarity measures are addressed in Section 4.1. Sometimes merges are limited to assigning a singleton vertex into a cluster, sometimes also larger clusters may be merged — the former case aims to a single flat clustering, the latter to a hierarchy of clusterings. As with divisive methods, iterative merging typically continues until some threshold or a desired number of clusters is reached.

An approach that roots in the clustering of point sets is to begin grouping the vertices into clusters by forming a two-vertex cluster from the two most similar vertices. This is very similar to what is often done in the case of clustering points in space [135]. The intuition is that at least the two closest points should be placed in the same cluster with each other. Such merging then continues until only a desired number of clusters remains or another stopping condition is met. At each iteration, one picks the two clusters (singletons or larger) that have the highest similarity value to be merged. Note that a function must be defined for determining the similarity of vertex subsets of different orders.

This approach is generally known as the *pairwise nearest neighbours* method and the merging criterion in general is based on greedy optimization of an objective function. The method is studied in the dissertation of Virmajoki [99].

Agglomerative clustering algorithms include that of Carrasco et al. [41] for bipartite graphs and that of Hopcroft et al. [128] for general graphs. Du [78] first clusters the graph into initial clusters using information on vertex degrees and then agglomeratively combines the initial clusters until an agreeable clustering is achieved. Clauset et al. [56] present a betweenness-based method that runs in $\mathcal{O}(n \log^2 n)$ time in practice for sparse natural graphs. The method performs greedy optimization of modularity (Eq. (45)) similarly to the method of Newman [185]: Clauset et al. greedily maximize the ratio of internal edges for cluster members that have nonzero external degree.

Another modularity-optimizing approach is presented by Donetti and Muñoz [74] who perform agglomerative clustering using spectral properties to construct the full cluster hierarchy and then select a clustering from the resulting tree maximizing modularity. The idea is to first let the hierarchical clustering algorithm create the entire dendrogram for the data and then optimize modularity over all possible sets of dendrogram "nodes". As nodes chosen represent vertex sets, candidate sets of chosen nodes must form a cover of the vertex set. The set of nodes that gives the highest modularity is chosen to be the final, flat clustering.

6. Local clustering in graphs

For large graphs, global clustering becomes computationally demanding. For massive data sets, the running time of a clustering algorithm should not grow faster than $\mathcal{O}(n)$ in order to be scalable; sublinearity is strongly preferable.

What comes to memory consumption, storing the complete edge set that for dense graphs has size $\mathcal{O}(n^2)$ is also often infeasible. There are applications where the input cannot possibly be read into the main memory at once and the computational cost of swapping the memory contents may prove critical. For large enough graphs, even sparsity does not help much: for example, the World Wide Web has billions of vertices and many more edges, setting it out of reach of the global algorithms.

However, if the graph is stored in a format that allows access to connected subgraphs or adjacency lists of nearby vertices, ideas similar to agglomerative clustering can be applied: clusters can be computed one at a time based on only partial views of the graph topology. This is called *local clustering*. An example of a data structure allowing local access to adjacency information is a search tree of adjacency lists with vertex identifiers as seeds.

Additional motivation for *local* clustering methods come from large networks that are not explicitly available, but rather require on-demand generation or exploration with a *crawler*, such as the programs that are used to index the World Wide Web for search-engine construction [50,221].

Computing the desired answer by a clustering algorithm for many applications only requires a small subset of vertices to be clustered instead of the whole graph. Such tasks include locating documents or genes closely related to a given "seed" data set. The scalability problem of global clustering is avoided, as the graph as a whole does not need to be processed unless a single cluster contains nearly the entire graph. Also, clusters for different seeds may be simultaneously obtained by parallel computation.

In this section we study local approaches for finding a good cluster containing a specified *seed* vertex or a set of vertices by examining only a limited number of vertices at a time, proceeding in the "vicinity" of the seed vertex. We denote by $\mathcal{C}(v)$ the cluster of vertex v, that is, the resulting cluster when using v as the seed vertex.

An application-specific detail that arises in local clustering is whether the clustering should be *symmetrical*: that is, if vertex v belongs to the cluster $\mathcal{C}(u)$, should u necessarily belong to the cluster $\mathcal{C}(v)$? For example networks of social contacts tend to be asymmetrical: you may consider someone to be your acquaintance while that person does not remember having met you before. Similarly you may be considered a friend by someone you would think a mere acquaintance. Directed input graphs can be expected to allow more natural clusterings when symmetry is not required.

It is noteworthy that local clustering algorithms may be used to obtain a global clustering of the entire input graph [17,54,205]: the options include, for example, initiating the procedure n times using each vertex as the seed vertex once and applying some majority-vote rule or a quality measure to combine the local clusters into a global clustering. Also

computing a fuzzy global clustering from a set of locally determined clusters is an option.

Another approach for deriving a global clustering through a local method is to select seed vertices according to some preference rule and excluding the local clusters found from further clustering, hence limiting the number of times the local procedure is executed. The computation for clusters with different seeds may in some cases be near-trivially parallelized, as the clusters are formed independently, as in the case of the Fiedler-vector method of Orponen and Schaeffer [191].

6.1. Definition of locality in graphs

In order to distinguish local computation from global computation, we must define what information about an input graph we consider to be *locally available*. For a single seed vertex s, we assume that at least the adjacency list containing the identifiers of vertices in $\Gamma(s)$ are known and that the algorithm may "crawl" into any of the neighbouring vertices. A wider definition would also allow direct access from a vertex v to its second neighbours:

$$\bigcup_{v\in\Gamma(s)} \Gamma(v)\,. \tag{48}$$

Optionally we can allow knowledge of the degrees of the vertices in $\Gamma(s)$. However, such information could also be obtained by on-demand local crawling from s.

We also allow a local algorithm to remember any adjacency information that has already been seen, although in practice restrictions are posed on the amount of memory available. Therefore, once a local algorithm has computed a candidate cluster $\mathcal{C}$ for s, it knows the list of vertices included in $\mathcal{C}$, any edge internal to $\mathcal{C}$, as well as a list of border vertices directly adjacent to the cluster (i.e. vertices that are neighbours of at least one included vertex but are not themselves included in $\mathcal{C}$).

For edge weights, the weight of each edge that has at least one endpoint in the subgraph is considered locally available. For vertex weights, the weights of the included vertices *and* their immediate neighbours are considered locally available.

6.2. Local search

Local search methods are heuristic and/or probabilistic algorithms designed to find near-optimal solutions among large, complex sets of solution candidates. These methods aim not explore the entire solution space but rather, possibly with probabilistic decision-making, study a limited region that contains at least good if not the best solutions. The extent in which the input graph is traversed depends on the local search method applied.

Each solution candidate is represented by a state and the set of states is called the *state space*. A *neighbourhood relation* over the space of possible solutions, with the goal of examining solution candidates one by one and then moving to a neighbouring candidate. The neighbourhood relation should be such that the search may navigate from one state to another with light computation. One also needs to define a *fitness function* that measures the quality of the solution represented by a state. The computational cost of evaluating the function should be small or at least moderate, as during the course of a search it will be repeatedly evaluated for different solution candidates.

The rule for choosing the next state to proceed to may be *heuristic*, which means that a *fitness function* is evaluated for all neighbours and the outcome is used to choose to which neighbour the search will proceed. Always moving to the neighbour with best fitness is a *greedy* strategy. The selection may also involve a probabilistic element, for example proceeding to each neighbour with a probability proportional to the value of the fitness function. While proceeding through the search space, the search algorithm always remembers at least the best state visited and the associated fitness.

Typically a limit is imposed on the number of steps the search may take. The search terminates either when it encounters a solution that has the best theoretically possible fitness (ideally corresponding to having found an exact optimum) when it reaches the step-count limit. The solution candidate yielding the best fitness value is the output of the local search procedure. The search may be iterated several times in order to cover more of the state space. For deterministic heuristics, one should use random initial states to explore different parts of the search space, but for probabilistic heuristics, also fixed initial states work.

Some common local search procedures are *hill-climbing*, deterministic and probabilistic versions of *tabu search*, and *simulated annealing*. Care must be taken in guiding the search in the neighbourhood: the presence of local optima is likely and the search must be guided in such a manner that it is possible to "escape" a local optimum with reasonable effort.

For this purpose, simulated annealing allows the search to proceed to a lower-fitness neighbour in the search space with probability that decreases over time as the search proceeds. The speed at which the probability decreases is controlled by two parameters: the *initial temperature* and a *cool-down coefficient* — the aim is to mimic cooling in metals. In addition, one fixes the number of iterations to be computed and the number of steps taken per each iteration. The parameters are usually chosen by running some initial experiments and choosing a parameter set that gives promising results [155].

Graph partitioning by simulated annealing has been studied by Johnson et al. [137], comparing it to the Kernighan–Lin algorithm [145] that is a well-known method for partitioning weighted graphs with respect to the edge weights. A similar approach for points in space has been suggested by Klein and Dubes [149], who compared the clusterings achieved with simulated annealing to those of a k-means algorithm. Booth et al. [29] study data partitioning with another stochastic search method, namely the Metropolis-Hastings algorithm [6]. Felner [91] uses heuristic search to solve the graph partitioning problem using heuristics for estimating the size of the optimal partition based on graph structure.

Schaeffer [205] computes with simulated annealing the cluster $\mathcal{C}(s)$ of a single, given seed vertex s, only considering all possible clusters in which s can be assigned. This reduces the size of the search space significantly: instead of having to construct a global clustering of all the vertices in the input

graph, one only needs to determine for each vertex whether it is included in $\mathcal{C}(s)$ or not. For such a procedure to be local, the fitness function used should be locally computable in the sense that the fitness of a cluster candidate should not depend on global properties of the graph.

In initiating the local search procedure, Schaeffer [205] uses a fixed initial state where the cluster candidate is formed by the seed vertex v itself along with its neighbourhood $\Gamma(v)$. The neighbourhood relation is formed by allowing two kinds of operations: the addition of an adjacent vertex and the removal of an included vertex. Upon the removal of $u \in \mathcal{C}(v)$, $u \neq v$, she ensures connectivity by setting the connected component containing v to be the next cluster candidate. The approach generalizes to a variety of initial states, fitness functions, and heuristics.

For a good review on local search methods, we recommend the book by Michalewicz and Fogel [177] and also the book by Aarts and Lenstra [2]. Stochastic search is by no means limited to local computation. For moderate-size instances, one may define as the search space the set of all possible clusterings and locally optimize a clustering quality measure by moving in the space of all possible clusterings.

6.3. *Fitness functions*

Possible fitness functions for such local clustering are numerous. Orponen and Schaeffer [191] use local search to optimize the a weighted version of the *Cheeger ratio*. The Cheeger ratio [45] is defined in the unweighted case as

$$\frac{|\{\{v,w\} \mid v \in \mathcal{C}, w \in V\backslash\mathcal{C}\}|}{\min\left\{\sum_{v\in\mathcal{C}} \deg(v), \sum_{w\in V\backslash\mathcal{C}} \deg(w)\right\}}, \tag{49}$$

which is the ratio of the cut size of the cluster to the minimum of the sums of degrees either inside the cluster or outside it. Orponen and Schaeffer use Fiedler vectors to compute edge weights (as described in Section 5.4.5) and use weighted version of degrees and cuts, where instead of counting edges their weights are summed. Optimizing the Cheeger ratio is equivalent to a graph partitioning problem [121]. An approach for computing *local cuts* using the Cheeger ratio is presented by Chung [52]. Andersen, Chung and Lang [9] in turn utilize PageRank-like computations for local graph partitioning. Practically all of the cut-related measures presented in Section 5.4.1 yield good fitness functions for local search.

The naïve definition of a cluster as a subgraph with maximum density fails in many ways, as discussed in Section 4.1.2: one easily ends up with picking single-edge two-cliques or solving an **NP**-complete problem. Nevertheless, for many optimization problems, good fitness measures inevitably correspond to **NP**-complete or **NP**-hard decision problems. As density is a natural clustering measure, variants of edge-count related measures are useful components in defining well-behaving fitness measures for clustering.

In general, the higher the internal degree $\deg_{\text{int}}(v)$ Eq. (21), the better v fits into the given cluster. In order to compare the suitability over different cluster candidates with varying order, we need to scale this by the maximum number of neighbours that a vertex could have in $\mathcal{C}$, namely $|\mathcal{C}| - 1$, to obtain a measure in $[0, 1]$:

$$\delta(v, \mathcal{C}) = \frac{\deg_{\text{int}}(v, \mathcal{C})}{|\mathcal{C}| - 1}. \tag{50}$$

This measure indicates how densely v is connected to $\mathcal{C}$ and it should give a high value if $\mathcal{C}$ is a good cluster for v. We also want to make sure that the vertex is not densely connected to *other* parts of the graph, and hence define a measure in $[0, 1]$ for vertex *introversion*, namely the ratio of internal edges to all edges incident on v:

$$\rho(v, \mathcal{C}) = \frac{\deg_{\text{int}}(v, \mathcal{C})}{\deg(v)}. \tag{51}$$

If both of the above measures have a high value, we can assume v to be correctly classified into $\mathcal{C}$. If either one is low, it would be worthwhile to try reassigning v to some other cluster.

The quality of a given cluster can be evaluated on the basis of the suitability of the included vertices; a possible measure for cluster density would be a scaled sum of vertex densities Eq. (50):

$$\delta_{\text{s}}(\mathcal{C}) = \frac{1}{|\mathcal{C}|}\sum_{v\in\mathcal{C}} \delta(v, \mathcal{C}) = \frac{1}{|\mathcal{C}|\,(|\mathcal{C}| - 1)}\sum_{v\in\mathcal{C}} \deg_{\text{int}}(v, \mathcal{C}). \tag{52}$$

The sum of the internal degrees of vertices in $\mathcal{C}$ is twice the internal degree of the cluster, as each internal edge is counted independently by both of its endpoints. This simplifies Eq. (52) into

$$\delta_{\text{s}}(\mathcal{C}) = \frac{1}{|\mathcal{C}|\,(|\mathcal{C}| - 1)} \cdot 2\deg_{\text{int}}(\mathcal{C}) = \frac{\deg_{\text{int}}(\mathcal{C})}{\binom{|\mathcal{C}|}{2}} = \delta(\mathcal{C}) \tag{53}$$

obtaining exactly the local density of the subgraph induced by the vertex set $\mathcal{C}$ (Eq. (10)).

The natural cluster of a vertex is not necessarily a complete subgraph, but rather just a "surprisingly" dense subgraph considering the global density of the graph. Note that the local density can also be interpreted as the probability that two included vertices are connected, and the higher the probability, the more tightly connected the cluster. Optimizing $\delta(\mathcal{C}) \in [0, 1]$ alone makes small cliques superior to larger but slightly sparser subgraphs, which is often undesired. Hence it is essential to find a fitness function that avoids getting "stuck" at small cliques containing v. The cluster of v should intuitively contain *at least* the largest clique that contains v.

The *introversion* of a cluster $\mathcal{C}$ can be similarly characterized by summing the suitability measures of Eq. (51) and scaling with the cluster order to obtain a measure in $[0, 1]$:

$$\rho_{\text{s}}(\mathcal{C}) = \frac{1}{|\mathcal{C}|}\sum_{v\in\mathcal{C}} \rho(v, \mathcal{C}) = \frac{1}{|\mathcal{C}|}\sum_{v\in\mathcal{C}} \frac{\deg_{\text{int}}(v, \mathcal{C})}{\deg(v)}. \tag{54}$$

If we calculate the *capacity* of the cut $(\mathcal{C}, V\backslash\mathcal{C})$ for a blind random walk, we obtain

$$Q(\mathcal{C}, V\backslash\mathcal{C}) = \sum_{\substack{v\in\mathcal{C}, u\in V\backslash\mathcal{C} \\ \{v,u\}\in E}} \frac{1}{\deg(v)} = \sum_{v\in\mathcal{C}} \frac{\deg_{\text{ext}}(v, \mathcal{C})}{\deg(v)}. \tag{55}$$

As $\deg_{\text{ext}}(v, \mathcal{C}) = \deg(v) - \deg_{\text{int}}(v, \mathcal{C})$ by definition (Eq. (21)), there is an obvious relation between the capacity of the cut

and Eq. (54) above. A cluster is properly introvert if Eq. (54) has a high value and the capacity of the cut is low.

Relative density favours introvert clusters, i.e. subgraphs with few connections to other parts of the graph. Introversion measures are, however, optimized for any connected component in which all edges are by definition internal, yielding zero for cut capacity and one for relative density as well as the summation of Eq. (54). This imposes restrictions on their usage as fitness functions, as a local search method would prefer any connected component as a cluster selection even if it would allow intuitively pleasing divisions into smaller clusters.

Bagrow and Bollt [17] grow a cluster candidate by breadth-first search level by level (always adding all the neighbours of presently included vertices at once) and optimizing the *emerging degree* of the cluster candidate, which is effectively the external degree. A threshold is used to determine when to stop growing the cluster further, in order to avoid including the entire connected component of the start vertex. Also Clauset [54] uses a measure similar to relative density. The local clustering algorithm of Clauset greedily optimises the fraction of the internal edges of *boundary vertices* only, i.e. vertices $v \in \mathcal{C}$ such that $\deg_{\text{ext}}(v, \mathcal{C}) > 0$.

One possible interpretation of the relative density is as follows: consider a global, partitional clustering of $G = (V, E)$ into clusters $\mathcal{C}_1, \ldots, \mathcal{C}_k$. Evidently

$$\sum_{i=1}^{k} (\deg_{\text{int}}(\mathcal{C}_i) + \deg_{\text{ext}}(\mathcal{C}_i)) = m + \sum_{i=1}^{k} \deg_{\text{ext}}(\mathcal{C}_i), \tag{56}$$

as every external edge has endpoints in exactly two clusters. Now for a clustering to be of high quality in terms of introversion, as m is a constant, we are interested to minimize $\sum_{i=1}^{k} \deg_{\text{ext}}(\mathcal{C}_i)$, which means that out of all clusterings into k clusters, one clustering is better than another if any two clusters have a smaller external degree whereas the external degrees of the others remain unaltered. Note that modifying just a single cluster is not possible, as a removed vertex must be included into another cluster. The computation is even more tedious if the number of clusters is allowed to vary.

Hence, to approximate this global optimum, each cluster may locally attempt to minimize its own $\deg_{\text{ext}}(\mathcal{C}_i)$; as the cluster should also attempt to be the maximal-order cluster with the minimal external degree, it should favour higher values of $\deg_{\text{int}}(\mathcal{C}_i)$ over lower ones, meaning that it attempts to maximize $\deg_{\text{int}}(\mathcal{C}_i)$ while minimizing $\deg_{\text{ext}}(\mathcal{C}_i)$, which can be directly achieved by maximizing the ratio $\deg_{\text{int}}(\mathcal{C}_i) / \deg_{\text{ext}}(\mathcal{C}_i)$. This measure, however, can take arbitrary positive values over connected cluster candidates and may result in division by zero in the absence of external edges. In order to scale it to values in $[0, 1]$ and avoid division by zero, we add to the denominator the value of the numerator, which yields exactly Eq. (43).

The relative density is the probability that a randomly chosen edge incident on the cluster is an internal edge, whereas the local density can be interpreted as the probability that two randomly chosen cluster members are connected by an edge. In a good *global* clustering, when picking an edge uniformly at random, we would like the probability that it is internal to a cluster to be high. Also, we would like the probability that two vertices that are in the same cluster are connected to be high, interpreting strong connectivity as an indicator of vertex similarity.

The cluster fitness function used by Schaeffer [205,206] is the product of the local (Eq. (53)) and relative (Eq. (43)) densities

$$\mathcal{F}(\mathcal{C}) = \delta(\mathcal{C}) \cdot \rho(\mathcal{C}) = \frac{2 \deg_{\text{int}}(\mathcal{C})^2}{|\mathcal{C}|\,(|\mathcal{C}| - 1)(\deg_{\text{int}}(\mathcal{C}) + \deg_{\text{ext}}(\mathcal{C}))}. \tag{57}$$

It is but one of the many possible combinations of the local and relative density measures.

7. Comparison, evaluation and benchmarking

For traditional methods of clustering points in space, clusters that are of different orders or shapes often produce difficulties, as well as clusters that overlap each other [98]. Similarly in graph clustering, when the clusters are of different orders and have varying densities, global methods tend to run into difficulties in correctly classifying them.

Properties of good clusterings are discussed by Kleinberg [150]. He defines an axiomatic framework for clustering a data set $S = \{1, 2, \ldots, n\}$ of "abstract points" using the notion of a *clustering function* f that takes as a parameter a distance function $d : S \times S \to \mathbb{R}$ and returns a partition of the data set into clusters based on the distance function d. The distance function must be such that all reflexive distances are zero and all other distances are positive and symmetrical. The triangle equation is left as an option instead of requiring it. Kleinberg lists three desirable properties of a clustering function:

1. *Scale-invariance*: given any distance function d and a constant $\alpha > 0$, it should hold that multiplying all distances by α *does not* change the clustering,
2. *Richness*: the range of f is the set of all partitions of S, meaning that the function is capable of producing *any* of the possible partitions of the data set S given the appropriate distance function d,
3. *Consistency*: let $P = (\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k)$ be a partition of S given a distance function d and d' be another distance function such that $\forall i, j \in S$
   - if i and j belong to the *same* cluster $\mathcal{C}_\ell$ of the partition P, it applies that $d'(i, j) \leq d(i, j)$, and
   - if i and j belong to *different* clusters of P, it applied that $d'(i, j) \geq d(i, j)$,

   then $f(d) = P$, meaning that no modification to the distance function that never lengthens an intracluster distance and never shrinks an intercluster distance should cause the clustering to change.

The theorem of Kleinberg [150] is that for $n \geq 2$ no clustering function f exists that satisfies all the above properties 1, 2, 3. Unfortunately to the case of graph clustering, these properties do not translate directly into graph clustering in general. We may apply them all to the scenario of clustering a complete weighted graph where the weights are assigned by the distance function d and the vertex set is S, but when not all edges are present or when the graph is unweighted, it is not straightforward to fill the role of the distance function in the definitions of Kleinberg [150]. In a sense, the edge relation E would be a

candidate: richness could be interpreted as the capability of producing all clusterings while choosing an appropriate edge set E, consistency in the sense that adding intra-cluster edges and/or removing intercluster edges should not change the clustering, but scale-invariance in the absence of edge weights is more challenging without modifying the vertex set as well. For a general weighted graph, scale-invariance is the easiest property to check: the clustering should not change if all weights are multiplied by a positive constant.

In practise, a question easier than "is this a good clustering" is "which of these two clusterings is better". With flat clusterings, we may define comparison measures such as overlaps or agreements between two clustering. Such measures are straightforward to define iterating over the vertices; for example, if clustering $\mathcal{C}_i(1), \mathcal{C}_i(2), \ldots, \mathcal{C}_i(n)$ and clustering $\mathcal{C}_j(1), \mathcal{C}_j(2), \ldots, \mathcal{C}_j(n)$ have a value close to one for

$$\frac{1}{n}\sum_{v\in V}\frac{\left|\mathcal{C}_i(v)\cap\mathcal{C}_j(v)\right|}{\left|\mathcal{C}_i(v)\cup\mathcal{C}_j(v)\right|}, \tag{58}$$

in a sense the clusterings agree well. However, the measure does not behave well if the clusters of one clustering are in fact subclusters of the clusters of the other clustering. For moderate-size graphs, visualization again helps, for example, by colouring the vertices according to the clusterings. When comparing two hierarchical clusterings, more complicated schemes are needed to evaluate to which extent the two divisions agree, especially if the dendrograms are of different heights.

Two flat clusterings can also be compared or the quality of a single clustering evaluated by examining the adjacency matrix of the graph ordered by clusters — an example of an adjacency-matrix visualization was given in Fig. 1. For comparison, first order the matrix by placing the vertices in order that follows the first clustering and see if the second clustering also produces a near block-diagonal structure, and then repeat ordering by the second clustering. For evaluating the cluster quality, such visualization helps reveal the presence of dense clusters. Mathematically this could be achieved, for example, by calculating the distance of each element that has the value one to the diagonal in the adjacency matrix — the smaller the value the better the clustering. Block-diagonalization has been utilized in relation to clustering by Schaeffer [206] and Carrasco et al. [41].

Another option is to compute vertex similarity measures (such as those of Section 4.1) within and across clusters — a clustering is good if the intracluster similarities are high and the inter-cluster similarities low. Similarly one could use cluster quality measures (such as those of Section 4.2) and prefer a clustering that has higher overall quality. The vertex similarities can be also be used to construct a minimum spanning tree to the graph using the inverse of the similarity as a distance measure. A good clustering is such that each cluster corresponds to a connected subtree of the minimal spanning tree [245]. Alternative uses of this observation, other than using it to evaluate the quality of a given clustering, is to cluster the spanning tree instead of the graph as a whole, as this usually leaves a large portion of the edges out of the consideration and eases the computation.

In many cases, the graph that was clustered was based on some data set, all information of which was not completely utilized in the construction of the graph. Carrasco et al. [41] also use that additional information to evaluate the quality of clusterings obtained by different methods.

Determining whether one clustering algorithm is better than another would be simplified were there a canonical set of benchmark cases, i.e. graphs for which a "correct" clustering is known. Typically used graphs include the karate club social network of Zachary [244] and other social networks for which a semantic division into clusters is known beforehand (such as known research groups in scientific collaboration networks). However, as clustering tends to be rather application specific, comparing any two algorithms not always makes sense, as the motivation and intended application areas differ.

Problems arise in the evaluation task especially when a global clustering needs to be compared with a local clustering, as global clusterings as often partitions or at least symmetrical whereas the question posed by local clustering allows for covers and asymmetrical cluster-membership relations.

7.1. *The parameter jungle*

Typically, clustering algorithms have at least a few parameters. In comparing the output of different algorithms, one needs to choose the parameters of the two algorithms under comparison fairly. This is not always trivial, as the resulting clustering may heavily depend on the parameter values chosen. Therefore, before defining quality indices, we address the problem of parameter selection.

The purpose of the parameters is to attempt to overcome difficulties caused by structural properties inherent in the data set, such as varying densities. Determining the optimal values for the parameters is usually nontrivial or even impossible, and the methods may be highly sensitive to the choice of parameter values. A common parameter is the number of clusters to compute. When clustering data such as speech or handwritten characters, aiming to identify which sound in the speech correspond to the same phoneme or which characters of the writing correspond to the same letter, the number of clusters is determined by the number of phonemes or the size of the alphabet. In many cases, however, the user will not have any a priori information on the number of clusters. For example, when clustering a social network based on phone call data, knowing the number of callers and calls made does not give any concrete information on how many clusters the graph could be expected to form. The use of methods requiring the number of clusters as a parameter is more straightforward when the user has at least some information on the range of possible cluster counts.

A problem related to choosing the number of clusters is that many algorithms implicitly assume that the clusters should be of similar orders, even though this is not necessarily the case in real-world data. The problem may be avoided either by local clustering methods where the size of the other clusters present plays no role, or by resorting to methods that have been designed to find clusters of different

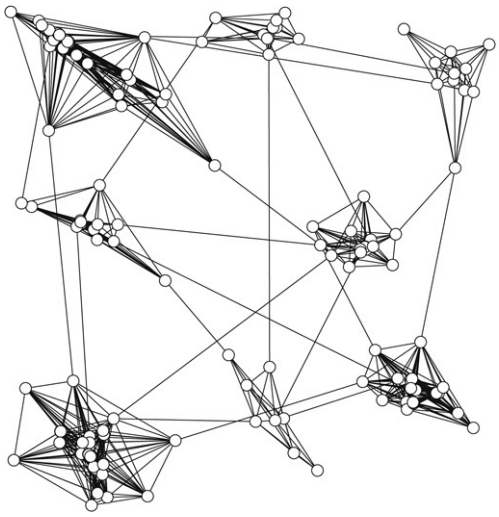


Fig. 12 – Visualization of a 160-vertex relaxed caveman graph [228] with $m = 1415$ edges computed by starting with random initial positions for each vertex and using a spring-force algorithm to iteratively move them to the final locations [203]. The graph was generated with a model that introduced a clear eight-cluster structure, but no information on the clustering was given to the spring-force algorithm. However, the natural grouping by balancing the "springs" of the edges matches the inherent cluster structure.

orders. We return to the problem of choosing the number of clusters in the next section.

With small instances, one may experiment with the parameter values and compare the resulting clusterings with respect to some quality measure. We will address the issue on how to evaluate the quality of a given clustering in Section 7. It is not computationally feasible to try out clusterings with different parameter values if the input graph is large.

In hierarchical clustering methods, one typically needs to define a threshold for the stopping condition. The measure used to determine when the algorithm should stop may either be a global measure of the current clustering or a difference of the measures of the current and previous levels in the cluster hierarchy. It is advisable to try different values of the threshold. It is useful to apply some other quality measure than the one built in the stopping condition to determine which value yields the best clustering. Again such experimentation is costly if not impossible in the case of massive instances.

When nothing is known of the structure of the graph, for small instances it may prove useful to attempt visualizations. For example a simple spring-force algorithm already suffices to reveal dense clusters in a nonuniform graph [228]. Fig. 12 shows a small graph composed of dense clusters visualized with a spring-force method. For larger graphs, sampling subgraphs and visualizing them may be helpful, although the problem of sampling massive graphs is in itself problematic [206].

Another option to explore the graph structure is to compute or estimate characteristic measures such as the density, clustering coefficient and average path length of the graph to learn more of the graph structure. Such structural information may greatly ease choosing between different clustering algorithms as well as in determining the required parameter values. Comprehensive surveys on structural properties of natural graphs include those of da F. Costa et al. [64], Newman [183], Dorogovtsev and Mendes [76], and Chakrabarti and Faloutsos [42]. Approximations and sampling may again be useful for massive instances when estimating such measures.

A practical clustering algorithm should require few if any parameters. It should also be insensitive to small changes in the parameter values and the ranges in which the parameter are to be chosen should be clearly stated. Also, any information on what the parameter actually does and how it should be chosen is of great assistance to users outside the core of the scientific community of graph clustering researchers.

7.2. *Quality indices*

In this section we review some useful measures to evaluate whether a given cluster or clustering is of high quality which generally is a hard task, or which of two or more clusters (or clusterings) is the best. A brief review on some quality indices is given by Brandes et al. [34]. Cluster fitness functions, especially ones that are not used in the clustering algorithm itself, can be used to evaluate the clusterings produced and especially to choose between two alternative clusterings, preferring those clusterings that yield high-fitness clusters. Similarly, the measures discussed as possible split-criteria for hierarchical clustering (see Section 5.4) can be used as quality indices, such as conductance and the many variants [142]. Although optimizing such measures is computationally demanding, evaluating the value of each measure for a given clustering of a given graph is a lightweight operation.

Taking a global view on what a good clustering would be like, the best case would be that the graph were a collection of vertex-disjoint cliques, preferably even disconnected cliques. It would be easy to choose each clique to be a cluster without any room for further discussion. This motivates a method of evaluating the quality of a given clustering: how many edges would need to be added in total into all of the clusters to make them cliques and also, how many edges would need to be removed to disconnect each pair of clusters. One may sum or multiply these quantities and minimize the result: the fewer modifications to the graph are needed to turn it into a collection of cliques that are not connected to each other, the better the clustering.

Shamir et al. [208] study the complexity of determining the fewest changes to the edge set of an input graph to transform it into a set of vertex-disjoint cliques such that there are no intercluster edges. When edges can only be added, the problem is polynomial. However, when edges are added and removed, which is the more interesting general case, the problem is **NP**-complete. If edges can only be deleted, it is **NP**-hard to even approximate the problem within a constant factor. They also study variants in which the number of clusters is fixed to k.

Modularity $\mathcal{M}(\mathcal{C}_1, \ldots, \mathcal{C}_k)$ of a clustering, as defined in Eq. (45), evaluates a related property for weighted graphs. The higher the modularity, the better the clustering, as for a high-modularity clustering the total weight of intracluster edges is large and the total weight of intercluster edges is small. Modularity is in essence the graph-theoretical equivalent of minimizing the sum-of-squares of distances within clusters and maximizing it between the clusters for a clustering of a set of points in space [135], closely related to the Davies–Bouldin index [69].

Danon et al. [67] compare several graph clustering methods in terms of their sensitivity to changes in the input data and the running time, using modularity as a quality measure. They conclude that the most accurate methods are computationally expensive, but that it depends on the application whether speed or accuracy is more crucial. Intuition of intercluster sparsity combined with intra-cluster density has also been used by Brandes et al. [34] both with modularity-like formulations and conductance-based notions to evaluate the performance of clustering algorithms.

Other measures for evaluating a single cluster are distance measures such the average or maximum distance (i.e. length of the shortest path) within the included vertices, which should be small for a good cluster. These measures are useful if a method returns two candidate clusters and only one is to be chosen. A clustering fitness measure used by Wu et al. [237] compares the differences in average path lengths of the original graph and a graph where each cluster is contracted into a single vertex with distances calculated by having that single vertex represent all of its member vertices. This measure is called the *distortion of the graph geodesics*. An idea applied to clustering points in space that also gives insight to graph clustering is to use the sum of cluster diameters as a quality measure and preferring clusterings that yield smaller sums [44]. The motivation behind diameter-based clustering is that cluster members should be structurally close to each other, and hence connected by short paths. Also, it would be desirable that the diameters of the individual clusters were clearly smaller than the diameter of the graph as a whole.

Boutin and Hascoet [30] discuss and compare different cluster quality indices with respect to different instances and different clustering methods. They find that many of the quality indices are difficult to interpret and compare. One problem is that one should know what measures the algorithms under comparison are directly optimizing, as using the same measures as quality indices is not too informative: if method A optimizes measure f and method B optimizes measure g, one needs some intuition on what combinations of the values of f and g are desirable for good clusters. Preferably one should also assess the clusterings with quality measures other than f and g. This effectively brings us back to posing the question "what is a good cluster" that already served as a starting point for designing or choosing a clustering algorithm!

The *integrated classification likelihood* is a measure that can be used to choose the number of cluster [25,26]. It is based on assuming the data to follow a *finite mixture model*. Mixture models are statistical models for classification that deal with the probability that a given element belongs to a certain class. *Model-based clustering* aims to "recover" the classification in the data assuming the data to follow a finite mixture model, similarly to planted partition problem [248]. Also Fraley and Raftery [98] discuss likelihood methods for determining the number of clusters.

Such definitions rely on the notion of *likelihood*, which is the hypothetical probability that the observation made would have been generated in a certain way and not any of the other possibilities within a finite set. In the context of clustering, the goal is to estimate the likelihood that a given sample belongs to a certain cluster. Several likelihood-based formulations exist for evaluating the quality of a given clustering assuming that the input data was generated under a specific probabilistic model.

For points in space, the aforementioned Davies–Bouldin index [69] is commonly used to choose the number of clusters: one repeats the same algorithm varying the parameter that determines the cluster count and chooses the clustering that optimizes the Davies–Bouldin index. For graphs in general, modularity (Eq. (45)) could be optimized. In the presence of a priori information on the generation model, the aforementioned optimization of a likelihood measure will work.

When optimizing a quality index, it would be useful to know what is the best possible value that can be reached for a given input graph, i.e. how many intercluster edges will there be at least and how many intra-cluster edges will be missing at least. Knowing this allows to determine whether a given clustering is actually globally good, whereas not knowing the optimum only justifies comparisons between two clusterings to determine if one is better than the other.

Considering the application-specific nature of clustering problems, there seems to be no answer that would satisfy all. Often the only sensible way to evaluate the quality of a clustering is to see how well it performs for the application at hand, i.e. how costly is the computation and what are the benefits of utilizing the obtained clustering.

7.3. Scalability and stability under perturbations

The amount of digitally available information grows rapidly and hence *scalability* of computational methods is becoming an increasingly critical issue. By scalability we mean that a great increase in the size of the problem instance should only cause moderate effects in the amount of computational resources needed. With global methods, often both computation time and memory requirements pose problems — in addition to designing novel scalable methods, parallel and distributed versions of known clustering algorithms help to achieve at least some scalability, as the cost of additional hardware is no longer tremendous. Another option is to resort to approximations of qualitatively good but computationally demanding methods. Another idea, applied by Milenova and Campos [179] to cluster high-dimensional feature-vector data, is to use *sampling*. Combinations of sampling techniques and local clustering algorithms could well yield easily scalable methods that produce high-quality global clusterings; one idea would be to choose a set of seed vertices with a carefully designed sampling method that gives preference to vertices that structurally make good cluster seeds and iteratively combine information of the local

clusters of the seed vertices to obtain a global clustering. For more information on sampling vertices of massive graphs, see for example the Markov-chain constructions discussed by Schaeffer [206] or the path-sampling method of Clauset and Moore [55] further discussed by Achlioptas et al. [3] and Dall'Asta et al. [65].

On the field of feature-vector data clustering, scalability issues are more thoroughly addressed. Zaïane et al. [246] present an experimental study of different clustering methods and also discuss the difficulty of cluster validation. Farnstrom et al. [87] present a highly scalable variation of the k-means algorithm that scans a large data set once and produces a clustering using a small memory buffer — a buffer that accommodates just one per cent of the input data already serves to produce good clusterings.

Another potentially critical issue is that in clustering applications, one may wish to maintain a clustering for a graph that undergoes frequent modifications. It is however application-dependent whether the changes in the clustering should be limited to the area of the modification or whether the change should propagate and alter the cluster structure in general, and if so, to what extent.

Raghavan and Yu [200] study the stability of clustering methods when the input data is perturbed. Examples of possible perturbations are the introduction or removal of a few edges and/or vertices. They measure the stability of a clustering algorithm by computing a clustering for the original data and again for the perturbed data, then calculating how many operations it would take to transform the latter set of clusters to the former. Raghavan and Yu compare different graph-theoretical clustering methods and cluster definitions. Also Hopcroft et al. [128] evaluate their agglomerative clustering algorithm with respect to perturbations.

8. Applications of graph clustering

As has been emphasized repeatedly throughout the survey, the task of clustering is highly application-specific. In this section we review some of the key application areas of graph clustering, although it is not to be forgotten that many problems allow the utilization of other representations as well and hence clustering algorithms for feature vectors or others kinds of classification systems, for example, may equally be applied. We begin by viewing how data sets composed of points in n-dimensional space can be transformed into graphs.

8.1. Data transformations

The range of interesting clustering applications is wide, as many if not practically all systems of interacting (or simply coexisting) entities can be modelled in some way as graphs. For data that are not readily in graph, several *transformations* into graph representations are possible. In this section we discuss some of the various possibilities to convert feature-vector data into graph format. Transformations vice versa exist as well [234], but as the focus of this survey are graph-theoretical clustering algorithms, we do not address those.

One option on how to convert feature-vector data into graph format is the *Delaunay graph*. The Delaunay graph of a set of points on a plane can be constructed by representing each point by a vertex and placing an edge between each pair of points that are *Voronoi neighbours* [135]. The approach naturally generalizes to higher dimensions. Two points are Voronoi neighbours if their *Voronoi cells* are adjacent [13]. A Voronoi cell of a datum is formed by those points in the data space that are closer to that data point than any other. The boundaries of the Voronoi cells are hyperplanes that partition the space in which the data lie.

More often than relying to Delaunay graphs, when transforming feature-vector data into graph format, the data elements $d \in \mathcal{D}$ are represented by the vertices, and an edge is placed between two elements depending on their *similarity* under some measure, selected according to the application. Vertex similarity measures are addressed in Section 4.1.

Any data for which a similarity measure has been defined can be transformed into a complete weighted graph using its *connectivity matrix* $[M]_{i,j}$, where the element $m_{i,j}$ contains the similarity measure $\rho(d_i, d_j)$ for data elements d_i and d_j. If the similarity values are symmetrical, i.e. $\rho(d_i, d_j) = \rho(d_j, d_i)$, an undirected graph can be formed by representing each datum d_i by $v_i \in V$ and using

$$\omega\left(v_i, v_j\right) = \rho(d_i, d_j). \tag{59}$$

For asymmetrical similarities, the resulting graph is directed and hence the number of edges is doubled. In both cases, the a number of edges is $\mathcal{O}(n^2)$; for large data sets, this is computationally infeasible. The number of edges in the graph can be controlled by setting a threshold value ξ such that

$$\{v_i, v_j\} \in E \quad \text{if and only if} \quad \rho(d_i, d_j) \geq \xi, \tag{60}$$

although choosing the value of the threshold is application-specific and not always easily justified. Such edge-elimination is referred to as *sparsification* [219].

An example of other methods simplify the connectivity graph is that of Bansal et al. [18], who reduce the possible set of edge weights into a binary set by placing an edge with weight $+1$ between all vertex pairs that are similar (e.g. above a threshold) and an edge with weight -1 between those that are dissimilar. Optimal clustering of such graphs is **NP**-hard, but polynomial-time approximation schemes exist [18].

A similarity measure defined on the data set can usually be fluently converted into a distance measure: similarity measures assign large values for similar data elements, whereas distance measures assign smaller values for similar elements — often just taking the inverse of the similarity value works as a distance measure. A simple distance-based graph construction is assigning each point to be represented by a vertex and connecting each vertex by an edge with the vertices representing the k nearest points with respect to the distance measure [232]. The resulting graph is a *k-nearest neighbour graph*. Clustering of point sets by searching for cuts in such graphs was already studied in the 1970s by Zahn [245]. As an alternative to a fixed neighbour count, one may also set a distance *range* within which the neighbours are selected [135].

Of course one may also work the other way around, and convert a graph into a set of feature vectors, then

utilizing some clustering algorithm for general data sets. The task of constructing the feature vectors for the vertices is addressed by Wilson et al. [234] who use a spectral matrix and construct symmetric, permutation invariant polynomials from the matrix elements, then using the coefficients of the polynomials as feature vectors. In practice these vectors allow for a locally linear embedding in a low-dimensional space.

8.2. Information networks and usage information

In any communication network, graph clustering serves as a tool for analysis, modelling and prediction of the function, usage and evolution of the network. Applications include business analysis, marketing, improving the infrastructure, and identifying anomalous use.

In computer networks, clustering may be used to identify relevant substructures and to analyse the connectivity for purposes of modelling or structural optimization; the canonical example is the Internet and the structure of the Autonomous Domains [178]. One example of topology design through clustering is the work of Grout and Cunningham [114]. In the World Wide Web, clustering of hypertext documents – representing each web page by a vertex and each hyperlink by an edge – helps to identify topics and other entities formed by several interconnected documents [227,236].

What comes to Internet telephony and chat services like Yahoo Messenger, Microsoft's Messenger Live, and Skype, interesting usage statistics for optimizing related software and hardware configurations can be obtained by representing each user as a vertex and placing (weighted) edges between two users as they communicate over the system. For example, in a multiserver environment, savings could be obtained by grouping a dense cluster of users on the same server as it would reduce the interserver traffic.

Similar analysis can help traditional teleoperators identify "frequent call clusters", i.e. groups of people that all mainly call each other (such as families, coworkers, or groups of teenage friends) and hence better design and target the widely-spread offers on special rates for calling to a limited number of prespecified phone numbers. Clustering the caller information can also help to identify changes in the communication pattern of a certain client: when long calls are repeatedly being made outside the cluster, the phone may have been stolen or the client may simply have decided not to pay the bill anymore. For fraud detection, call durations and a geographical embedding would be most helpful in determining what forms the cluster of "normal call destinations" for a specific client and which calls are "out of the ordinary".

Clustering algorithms are also used in the structural design and operation of *ad hoc* [130,165,194] and *sensor networks* [101]. For networks with a dynamic topology, with frequent changes in the edge structure, local clustering methods prove useful, as the network nodes can make local decisions on how to modify the clustering to better reflect the current network topology [207]. Imposing a cluster structure on a dynamic network eases the routing task [156,216].

8.3. Database systems

When storing a large set of data, a key question is how to group the data onto *pages* in physical memory. A single page is typically large enough to contain multiple elements but only a small fraction of the entire data set. Therefore, a desirable grouping would be such than when a datum is retrieved, along would come relevant data so that possible related future queries might benefit from the already retrieved page. Also traditional concerns such as the complexity of searching, inserting, deleting, and modifying the data stored must be attended in addition to the relevancy concern in the paging design.

Diwan et al. [73] propose paging by clustering for tree-like data. Wu et al. [237] address the database organization issue for graphs, providing a solution where the data storage format itself supports quick, approximate computation of shortest paths and distances for a special class of nonuniform networks called *scale-free networks* [22]. A similar idea is presented by Agrawal and Jagadish [5], implicitly assuming an underlying cluster structure in the input graph. Their method was modified to explicitly use clustering algorithms by Schaeffer [206]. Bradley et al. [32] use a k-means like iterative algorithm to determine a clustering for a large database in one scan using a limited memory buffer.

8.4. Biological and sociological networks

In the field of *bioinformatics*, graph clustering tasks typically deal with classification of gene expression data (specifically gene-activation dependencies) [240,31] and protein interactions [16,193,148,243,7,141]. Another biological application of clustering is *epidemic spreading*. Newman [182] studies SIR-type epidemic processes in a special class of graphs and find that graphs with a cluster structure have smaller epidemics, but a lower epidemic threshold, making it easier for diseases to spread. Applications of local clustering in social networks include identifying groups of individuals "exposed" to the influence of a certain individual of interest, such as identifying terrorist networks when a member is known or locating potentially infected people when an infected and contagious individual is encountered.

Cluster analysis of a social network also helps to identify mechanisms underlying, for example, the formation of trends (relevant to market studies) and voter behaviour. In the current information society, the study of social networks tends to overlap the study of information networks, as the popularity and significance of electronic messaging has become overwhelming. However, traditional studies where the daily contacts of individuals are mapped and classified do coexist with the studies of chats and web logs.

8.5. Other applications

In the business world, other than market analysis based on social or communication networks, also stock market data can be clustered: represent each stock by a vertex and place weighted edges to represent the *correlations* of the valuations of the stocks in the stock market. Such a representation allows for the identification of clusters of stocks that

either all gain or lose value together, or alternatively – varying the cluster definition – stocks that appear to behave independently of each other. Such knowledge is useful in portfolio management when one wishes to distribute and/or concentrate investments.

A clustering analysis of the global air transportation network is given by Guimerà et al. [117]. In logistics, the *hub-location problem* [20,39] and other kinds of *facility-location problems* [61] are of interest. Several clustering-based heuristic solutions have been proposed to the hub-location problem [153,189] and heuristics and approximation algorithms that rely on a graph representation and a clustering computation exist for the facility-location problem as well [49,217]. A related problem of *sales territory design* [249, 140] also has proposed solutions building on the use of clustering methods [201,225].

Clustering also serves in manufacturing, where identification of clusters of similar parts helps to smoothe the production line (called *group technology*). Chen et al. [46] cluster transaction databases in hope of profit-increasing patterns using a noise-insensitive similarity measure between items based on cooccurrence relationship of items.

9. Open problems and future directions

In the previous sections we reviewed three major open problems of graph clustering:

- *Parameter selection:* how is the user to determine the parameter values to give as input to the clustering algorithm,
- *Scalability:* how does the runtime and memory consumption of the algorithm behave for massive input graphs, and
- *Evaluation:* how to decide which of several clusterings is the best.

For a non-expert user, ideally there would be few if any parameters and the output of the method would at least not be highly sensitive to the parameter values. The scalability issue can be resolved either by resorting to approximation algorithms to the existing methods or by novel approaches. Parallelization of local methods could also offer a solution. Especially the field of data mining frequently needs to evaluate cluster structures in very large datasets.

The problematics of evaluation could be eased by the creation of a benchmark set that allows comparison between different clustering methods at design phase. For the end user, it would be helpful in evaluating the output of different algorithms (or of the same method with different parameter values) if the clustering measures and/or the quality indices were intuitively pleasant even to users that are neither mathematicians nor computer scientist — should the measure be easily explained in lay person's terms, the user could analyse whether the vertices considered to be similar were actually grouped as similar.

More extensions of the existing graph clustering algorithms to weighted graphs would be of great interest, as well as novel methods for clustering directed graphs. In application areas, there is certainly need to cluster also multigraphs and hypergraphs.

The theoretical foundations of graph clustering are not yet fully explored; we believe that there may well be several supposedly distinct graph clustering algorithms that fundamentally compute the same exact thing. However, we do not expect there to be a single universal answer to the questions what is a good cluster in a graph and how to find it, as the field is highly application-specific.

10. Concluding remarks

In this survey we have given an overview of some of the essential definitions and techniques of graph clustering. In general, it seems that many of the good measures of clustering are intertwined: cut-based methods are in a sense spectral methods that in turn are related to random walks that model the behaviour of electrical networks and also serve to do betweenness-like computations, and so forth. These theoretical connections between many of the methods gives a reason to believe we are on the right track: the field of graph clustering seems to be revolving around fundamentally similar definitions, although some of the starting points for the algorithms are quite far apart.

We reviewed both global and local approaches and discussed the delicate issues of selecting an appropriate method for the task at hand, selecting good parameter values, and evaluating the quality of the resulting clustering. The tools available are already almost as various as the applications of graph clustering, although much work still remains to be done.

Acknowledgements

This work has been supported by the Academy of Finland under grants 81120 (STADYCS, 2002–2003) and 206235 (ANNE, 2004–2006), the Helsinki Graduate School in Computer Science and Engineering (2001–2002), the Nokia Foundation (2004), and the Rotary Foundation (2005). For their valuable comments, the author thanks Pekka Orponen, the editors and the anonymous reviewer, whose comments greatly improved the structure of the presentation.

REFERENCES

[1] Petros Drineas, Alan Frieze, Ravi Kannan, Santosh Vempala, V. Vinay, Clustering in large graphs and matrices, Machine Learning 56 (2004) 9–33.
[2] E. Aarts, J.K. Lenstra, Local Search in Combinatorial Optimization, John Wiley & Sons, Inc., Chichester, UK, 1997.
[3] D. Achlioptas, A. Clauset, D. Kempe, C. Moore, On the bias of traceroute sampling (or: Why almost every network looks like it has a power law), in: H.N. Gabow, R. Fagin (Eds.), Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing, STOC, ACM Press, New York, NY, USA, 2005.
[4] P.K. Agarwal, C.M. Procopiuc, Exact and approximation algorithms for clustering, in: Proceedings of the Ninth Annual ACM–SIAM Symposium on Discrete Algorithms, ACM–SIAM, Philadelphia, PA, USA, 1998.

[5] R. Agrawal, H.V. Jagadish, Algorithms for searching massive graphs, IEEE Transactions on Knowledge and Data Engineering 6 (2) (1994) 225–238.
[6] D.J. Aldous, J.A. Fill, Reversibe Markov Chains and Random Walks on Graphs. http://www.stat.berkeley.edu/aldous/RWG/book.html, 2001 (in preparation).
[7] M. Altaf-Ul-Amin, Y. Shinbo, K. Mihara, K. Kurokawa, S. Kanaya, Development and implementation of an algorithm for detection of protein complexes in large interaction networks, BMC Bioinformatics 7 (2006) 207.
[8] R.G. , T. Hu, Multiterminal network flows, SIAM Journal 9 (1961) 551–570.
[9] R. Andersen, F.R.K. Chung, K. Lang, Local partitioning using PageRank vectors, in: Proceedings of the Fourty-seventh Annual Symposium on Foundations of Computer Science, FOCS, IEEE Computer Society Press, Washington, DC, USA, 2006.
[10] S. Arora, S. Rao, U. Vazirani, Expander flows, geometric embeddings and graph partitioning, in: Proceedings of the Thirty-Sixth Annual Symposium on Theory of Computing, STOC, ACM Press, New York, NY, USA, 2004.
[11] Y. Asahiro, R. Hassin, K. Iwama, Complexity of finding dense subgraphs, Discrete Applied Mathematics 121 (1) (2002) 15–26.
[12] D. Auber, M. Delest, Y. Chiricota, Strahler based graph clustering using convolution, in: Proceedings of the Eighth International Conference on Information Visualisation, IEEE Computer Society, 2004.
[13] F. Aurenhammer, Voronoi diagrams — A survey of a fundamental geometric data structure, ACM Computing Surveys 23 (3) (1991) 345–405.
[14] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti Spaccamela, M. Protasi, Complexity and Approximation: Combinatorial optimization problems and their approximability properties, Springer-Verlag GmbH, Berlin, Heidelberg, Germany, 1999.
[15] F.R. Bach, M.I. Jordan, Learning spectral clustering, Tech. Rep. UCB/CSD-03-1249, Computer Science Division, University of California, Berkeley, CA, USA, Jun. 2003.
[16] G.D. Bader, C.W.V. Hogue, An automated method for finding molecular complexes in large protein interaction networks, BMC Bioinformatics 4(2).
[17] J.P. Bagrow, E.M. Bollt, Local method for detecting communities, Physical Review E 72 (2005) 046108.
[18] N. Bansal, A. Blum, S. Chawla, Correlation clustering, Machine Learning 56 (1–3) (2004) 89–113.
[19] J. Bar-Ilan, G. Kortsarz, D. Peleg, How to allocate network centers, Journal of Algorithms 15 (3) (1993) 385–415.
[20] J. Bar-Ilan, G. Kortsarz, D. Peleg, How to allocate network centers, Journal of Algorithms 15 (3) (1993) 385–415.
[21] J. Bar-Ilan, D. Peleg, Approximation algorithms for selecting network centers, in: F.K.H.A. Dehne, J.-R. Sack, N. Santoro (Eds.), Proceedings of the Second Workshop on Algorithms and Data Structures, WADS'91, in: Lecture Notes in Computer Science, vol. 519, Springer-Verlag GmbH, Berlin, Heidelberg, Germany, 1991.
[22] A.-L. Barabási, R. Albert, Emergence of scaling in random networks, Science 286 (5439) (1999) 509–512.
[23] E. Behrends, Introduction to Markov Chains, with Special Emphasis on Rapid Mixing, Vieweg & Sohn, Braunschweig, Wiesbaden, Germany, 2000.
[24] L.M.A. Bettencourt, Tipping the balances of a small world, Tech. Rep. MIT-CTP-3361 (cond-mat/0304321 at arXiv.org), Center for Theoretical Physics, Massachusetts Institute of Technology, Cambridge, MA, USA, 2002.
[25] C. Biernacki, G. Celeux, G. Govaert, Assessing a mixture model for clustering with the integrated completed likelihood, IEEE Transactions on Pattern Analysis and Machine Intelligence 22 (7) (2000) 719–725.
[26] C. Biernacki, G. Govaert, Using the classification likelihood to choose the number of clusters, Computing Science and Statistics 29 (2) (1997) 451–457.
[27] N. Biggs, Algebraic Graph Theory, 2nd ed., Cambridge University Press, Cambridge, UK, 1994.
[28] I.M. Bomze, M. Budinich, P.M. Pardalos, M. Pelillo, The maximum clique problem, in: D.-Z. Du, P.M. Pardalos (Eds.), in: Handbook of Combinatorial Optimization, vol. Supplement Volume A, Kluwer Academic Publishers, Boston, MA, USA, 1999, pp. 1–74.
[29] J.G. Booth, G. Casella, J.P. Hobert, Clustering using objective functions and stochastic search, Journal of the Royal Statistical Society, Series B (2007) (submitted for publication).
[30] F. Boutin, M. Hascoet, Cluster validity indices for graph partitioning, in: Proceedings of the Eighth International Conference on Information Visualisation, IEEE Computer Society, 2004.
[31] F. Boyer, A. Morgat, L. Labarre, J. Pothier, A. Viari, Syntons, metabolons and interactons: An exact graph-theoretical approach for exploring neighbourhood between genomic and functional data, Bioinformatics 21 (23) (2005) 4209–4215.
[32] P.S. Bradley, U.M. Fayyad, C. Reina, Scaling clustering algorithms to large databases, in: Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining, KDD, ACM, New York, NY, USA, 1998.
[33] U. Brandes, A faster algorithm for betweenness centrality, Journal of Mathematical Sociology 25 (2) (2001) 163–177.
[34] U. Brandes, M. Gaertler, D. Wagner, Experiments on graph clustering algorithms, in: G. Di Battista, U. Zwick (Eds.), Proceedings of the Eleventh European Symposium on Algorithms, in: Lecture Notes in Computer Science, vol. 2832, Springer-Verlag GmbH, Heidelberg, Germany, 2003.
[35] S. Brin, L. Page, The anatomy of a large-scale hypertextual Web search engine, Computer Networks and ISDN Systems 30 (1–7) (1998) 107–117.
[36] A.Z. Broder, S.R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, J. Wiener, Graph structure in the Web, Computer Networks 33 (1–6) (2000) 309–320.
[37] T.N. Bui, F.T. Leighton, S. Chaudhuri, M. Sipser, Graph bisection algorithms with good average case behavior, Combinatorica 7 (2) (1987) 171–191.
[38] H. Bunke, P. Foggia, C. Guidobaldi, M. Vento, Graph clustering using the weighted minimum common supergraph, in: E.R. Hancock, M. Vento (Eds.), Proceedings of the Fourth IARP International Workshop on Graph Based Representations in Pattern Recognition, in: Lecture Notes in Computer Science, vol. 2726, Springer-Verlag GmbH, Berlin, Heidelberg, Germany, 2003.
[39] J.F. Campbell, Hub location and the p-hub median problem, Operations Research 44 (6) (1996) 923–935.
[40] A. Capoccia, V. Servedioa, G. Caldarellia, F. Colaiorib, Detecting communities in large networks, Physica A: Statistical Mechanics and its Applications 352 (2–4) (2005) 669–676.
[41] J.J.M. Carrasco, D.C. Fain, K.J. Lang, L. Zhukov, Clustering of bipartite advertiser-keyword graph, in: Proceedings of the Third IEEE International Conference on Data Mining, Workshop on Clustering Large Data Sets, 2003.
[42] D. Chakrabarti, C. Faloutsos, Graph mining: Laws, generators, and algorithms, ACM Computing Surveys 38 (1) (2006) Article No. 2.
[43] M. Charikar, C. Chekuri, T. Feder, R. Motwani, Incremental clustering and dynamic information retrieval, in: F.T. Leighton, P. Shor (Eds.), Proceedings of the Twenty-ninth Annual Symposium on Theory of Computing, STOC, ACM Press, New York, NY, USA, 1997.

[44] M. Charikar, R. Panigrahy, Clustering to minimize the sum of cluster diameters, Journal of Computer and System Sciences 68 (2) (2004) 417–441.
[45] J. Cheeger, A lower bound for the smallest eigenvalue of the laplacian, in: Problems in Analysis: Symposium in Honor of Salomon Bochner (1969), Princeton University Press, Princeton, NJ, USA, 1970.
[46] N. Chen, A. Chen, L. Zhou, L. Lu, A graph-based clustering algorithm in large transaction databases, Intelligent Data Analysis 5 (4) (2001) 327–338.
[47] D. Cheng, R. Kannan, S. Vempala, G. Wang, On a recursive spectral algorithm for clustering from pairwise similarities, Tech. Rep. MIT-LCS-TR-906, Laboratory of Computer Science, Massachusetts Institute of Technology, Boston, MA, USA, 2003.
[48] D. Cheng, S. Vempala, R. Kannan, G. Wang, A divide-and-merge methodology for clustering, in: Proceedings of the Twenty-fourth Symposium on Principles of Database Systems, ACM Press, New York, NY, USA, 2005.
[49] F.A. Chudak, D.B. Shmoys, Improved approximation algorithms for the uncapacitated facility location problem, SIAM Journal on Computing 33 (1) (2003) 1–25.
[50] T.Y. Chun, World Wide Web robots: An overview, Online Information Review 22 (3) (1999) 135–142.
[51] F.R.K. Chung, Spectral Graph Theory, American Mathematical Society, Providence, RI, USA, 1997.
[52] F.R.K. Chung, Random walks and local cuts in graphs, Linear Algebra and its Applications.
[53] F.R.K. Chung, L. Lu, V. Vu, The spectra of random graphs with given expected degrees, Internet Mathematics 1 (3) (2004) 257–275.
[54] A. Clauset, Finding local community structure in networks, Physical Review E 72 (2005) 026132.
[55] A. Clauset, C. Moore, Accuracy and scaling phenomena in Internet mapping, Physical Review Letters 94 (1) (2005) 018701.
[56] A. Clauset, M.E.J. Newman, C. Moore, Finding community structure in very large networks, Physical Review E 70 (6) (2004) 066111.
[57] W.W. Cohen, P. Ravikumar, S.E. Fienberg, A comparison of string distance metrics for name-matching tasks, in: S. Kambhampati, C.A. Knoblock (Eds.), Proceedings of IJCAI-03 Workshop on Information Integration on the Web, IIWeb-03, AAAI, 2003.
[58] F. Comellas, S. Gago Álvarez, Spectral bounds for the betweenness of a graph, Linear Algebra and its Applications 423 (1) (2007) 74–80.
[59] A. Condon, R.M. Karp, Algorithms for graph partitioning on the planted partition model, Random Structures & Algorithms 18 (2) (2001) 116–140.
[60] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, 2nd ed., MIT Press and McGraw Hill, Cambridge, MA, USA, 2001, pp. 643–700.
[61] G. Cornuéjols, G.L. Nemhauser, L.A. Wolsey, The uncapacitated facility location problem, in: P.B. Mirchandani, R.L. Francis (Eds.), Discrete Location Theory, John Wiley and Sons, Inc., New York, NY, USA, 1990, pp. 119–171.
[62] P. Crescenzi, V. Kann, A compendium of np optimization problems. http://www.csc.kth.se/viggo/wwwcompendium/wwwcompendium.html, accessed on May 18, 2007.
[63] D. Cvetković, Signless laplacians and line graphs, Bulletin, Classe des Sciences Mathématiques et Naturelles, Sciences mathématiques Académie Serbe des Sciences et des Arts CXXXI (30) (2005) 85–92.
[64] L. da F. Costa, F.A. Rodrigues, G. Travieso, P.R. Villas Boas, Characterization of complex networks: A survey of measurements, Tech. Rep. cond-mat/0505185 arXiv.org, May 2005.
[65] L. Dall'Asta, I. Alvarez-Hamelin, A. Barrat, A. Vázquez, A. Vespignani, Exploring networks with traceroute-like probes: Theory and simulations, Theoretical Computer Science 355 (1) (2006) 6–24.
[66] L. Danon, A. Díaz Guilera, A. Arenas, The effect of size heterogeneity on community identification in complex networks, Journal of Statistical Mechanics Theory and Experiment (2006) P11010.
[67] L. Danon, A. Díaz Guilera, J. Duch, A. Arenas, Comparing community structure identification, Journal of Statistical Mechanics Theory and Experiment (2005) P09008.
[68] R.N. Dave, R. Krishnapuram, Robust clustering methods: A unified view, IEEE Transactions on Fuzzy Systems 5 (2) (1997) 270–293.
[69] D.L. Davies, D.W. Bouldin, A cluster separation measure, IEEE Transactions on Pattern Analysis and Machine Intelligence 1 (4) (1979) 224–227.
[70] N.M.M. de Abreu, Old and new results on algebraic connectivity of graphs, Linear Algebra and its Applications 423 (1) (2007) 53–73.
[71] J. Díaz, J. Petit, M. Serna, A survey of graph layout problems, ACM Computing Surveys 34 (3) (2002) 313–356.
[72] C. Ding, X. He, Linearized cluster assignment via spectral ordering, in: Proceedings of the Twenty-First International Conference on Machine Learning, vol. 69, ACM Press, New York, NY, USA, 2004.
[73] A.A. Diwan, S. Rane, S. Seshadri, S. Sudarshan, Clustering techniques for minimizing external path length, in: Proceedings of the Twenty-second International Conference on Very Large Data Bases (VLDB), Morgan Kaufmann Publishers, San Francisco, CA, USA, 1996.
[74] L. Donetti, M.A. Muñoz, Detecting network communities: A new systematic and efficient algorithm, Journal of Statistical Mechanics (2004) P10012.
[75] Yihong Dong, Yueting Zhuang, Ken Chen, Xiaoying Tai, A hierarchical clustering algorithm based on fuzzy graph connectedness, Fuzzy Sets and Systems 157 (13) (2006) 1760–1774.
[76] S.N. Dorogovtsev, J.F.F. Mendes, Evolution of networks, Advances in Physics 51 (4) (2002) 1079–1187.
[77] P.G. Doyle, J.L. Snell, Random Walks and Electric Networks, Mathematical Association of America, Washington, DC, USA, 1984.
[78] H. Du, An algorithm for detecting community structure of social networks based on prior knowledge and modularity, Complexity 12 (3) (2007) 53–60.
[79] R.C. Dubes, A.K. Jain, Clustering methodologies in exploratory data analysis, Advances in Computers 19 (1980) 113–228.
[80] D.P. Dubhashi, L. Laura, A. Panconesi, Analysis and experimental evaluation of a simple algorithm for collaborative filtering in planted partition models, in: P.K. Pandya, J. Radhakrishnan (Eds.), Proceedings of the Twenty-Third Conference on the Foundations of Software Technology and Theoretical Computer Science, in: Lecture Notes in Computer Science, vol. 2914, Springer-Verlag GmbH, Berlin, Heidelberg, Germany, 2003.
[81] R.O. Duda, P.E. Hart, D.G. Stork, Pattern Classification, 2nd ed., John Wiley & Sons, Inc., New York, NY, USA, 2001.
[82] J. Edachery, A. Sen, F.J. Brandenburg, Graph clustering using distance-k cliques, in: Proceedings of the Seventh International Symposium on Graph Drawing, in: Lecture Notes in Computer Science, vol. 1731, Springer-Verlag GmbH, Berlin, Heidelberg, Germany, 1999.
[83] P. Elias, A. Feinstein, C.E. Shannon, Note on maximum flow through a network, IRE Transactions on Information Theory IT-2 (1956) 117–119.

[84] P. Erdős, A. Rényi, On random graphs I, in: Selected Papers of Alfréd Rényi, vol. 2, Akadémiai Kiadó, Budapest, Hungary, 1976, pp. 308–315. First publication in Publ. Math. Debrecen 1959.
[85] P. Erdős, A. Rényi, On the evolution of random graphs, in: Selected Papers of Alfréd Rényi, vol. 2, Akadémiai Kiadó, Budapest, Hungary, 1976, pp. 482–525. First publication in MTA Mat. Kut. Int. Közl. 1960.
[86] I.J. Farkas, I. Derényi, A.-L. Barabási, T. Vicsek, Spectra of "real-world" graphs: Beyond the semicircle law, Physical Review E 64 (2) (2001) 026704.
[87] F. Farnstrom, J. Lewis, C. Elkan, Scalability for clustering algorithms revisited, SIGKDD Explorations 2 (2) (2000) 1–7.
[88] T. Feder, D.H. Greene, Optimal algorithms for approximate clustering, in: Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, STOC, ACM Press, New York, NY, USA, 1988.
[89] U. Feige, R. Krauthgamer, A polylogarithmic approximation of the minimum bisection, SIAM Journal on Computing 31 (4) (2002) 1090–1118.
[90] U. Feige, D. Peleg, G. Kortsarz, The dense k-subgraph problem, Algoritmica 29 (3) (2001) 410–421.
[91] A. Felner, Finding optimal solutions to the graph partitioning problem with heuristic search, Annals of Mathematics and Artificial Intelligence 45 (3–4) (2005) 292–322.
[92] M. Fiedler, Algebraic connectivity of graphs, Czechoslovak Mathematical Journal 23 (1973) 298–305.
[93] M. Fiedler, A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory, Czechoslovak Mathematical Journal 25 (1975) 619–633.
[94] G.W. Flake, S. Lawrence, C.L. Giles, F.M. Coetzee, Self-organization and identification of Web communities, IEEE Computer 35 (3) (2002) 66–71.
[95] G.W. Flake, R.E. Tarjan, K. Tsioutsiouliklis, Graph clustering and minimum cut trees, Internet Mathematics 1 (1) (2004) 385–408.
[96] L.R. Ford Jr., D.R. Fulkerson, Maximum flow through a network, Canadian Journal of Mathematics 8 (1956) 399–404.
[97] S. Fortunato, V. Latora, M. Marchiori, Method to find community structures based on information centrality, Physical Review E 70 (2004) 056104.
[98] C. Fraley, A.E. Raftery, How many clusters? Which clustering method? Answers via model-based cluster analysis, The Computer Journal 41 (8) (1998) 578–588.
[99] P. Fränti, O. Virmajoki, V. Hautamäki, Fast PNN-based clustering using k-nearest neighbor graph, IEEE Transactions on Pattern Analysis and Machine Intelligence 28 (11) (2006) 1875–1881.
[100] L.C. Freeman, A set of measures of centrality based upon betweenness, Sociometry 40 (1) (1977) 35–41.
[101] T. Furuta, M. Sasaki, F. Ishizaki, A. Suzuki, H. Miyazawa, A new cluster formation method for sensor networks using facility location theory, Tech. Rep. NANZAN-TR-2006-01, Nanzan Academic Society Mathematical Sciences and Information Engineering, Nagoya, Japan, August 2006.
[102] G. Gallo, M.D. Grigoriadis, R.E. Tarjan, A fast parametric maximum flow algorithm and applications, SIAM Journal on Computing 18 (1) (1989) 30–55.
[103] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman, San Francisco, CA, USA, 1979.
[104] M.R. Garey, D.S. Johnson, L.J. Stockmeyer, Some simplified NP-complete graph problems, Theoretical Computer Science 1 (3) (1976) 237–267.
[105] I. Gath, A.B. Geva, Unsupervised optimal fuzzy clustering, IEEE Transactions on Pattern Analysis and Machine Intelligence 11 (7) (1989) 773–780.
[106] E.N. Gilbert, Random graphs, Annals of Mathematical Statistics 30 (4) (1959) 1141–1144.
[107] M. Girvan, M.E.J. Newman, Community structure in social and biological networks, Proceedings of the National Academy of Sciences, USA 99 (2002) 8271–8276.
[108] C. Gkantsidis, M. Mihail, A. Saberi, Conductance and congestion in power law graphs, in: Proceedings of the International Conference on Measurement and Modeling of Computer Systems, ACM Press, New York, NY, USA, 2003.
[109] C. Gkantsidis, M. Mihail, E. Zegura, Spectral analysis of Internet topologies, in: Proceedings of the Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies, INFOCOM, vol. 1, IEEE, New York, NY, USA, 2003.
[110] K.-I. Goh, B. Kahng, D. Kim, Spectra and eigenvectors of scale-free networks, Physical Review E 64 (5) (2001) 051903.
[111] A.V. Goldberg, R.E. Tarjan, A new approach to the maximum-flow problem, Journal of the ACM 35 (4) (1988) 921–940.
[112] T.F. Gonzalez, Clustering to minimize the maximum intercluster distance, Theoretical Computer Science 38 (1985) 293–306.
[113] G.R. Grimmett, D.R. Stirzaker, Probability and Random Processes, 3rd ed., Oxford University Press, Oxford, UK, 2001.
[114] V. Grout, S. Cunningham, A constrained version of a clustering algorithm for switch placement and interconnection in large networks, in: T. Philip (Ed.), Proceedings of the Nineteenth International Conference on Computer Applications in Industry and Engineering, CAINE, ICSA, 2006.
[115] S. Guattery, G.L. Miller, On the quality of spectral separators, SIAM Journal on Matrix Analysis and Applications 19 (3) (1998) 701–719.
[116] S. Guha, N. Mishra, R. Motwani, L. O'Callaghan, Clustering data streams, in: Proceedings of the Fourty-first Annual Symposium on Foundations of Computer Science, FOCS, IEEE Computer Society Press, Los Alamitos, CA, USA, 2000.
[117] R. Guimerà, S. Mossa, A. Turtschi, L.A. Nunes Amaral, The worldwide air transportation network: Anomalous centrality, community structure, and cities' global roles, Proceedings of the National Academy of Science of the United States of America 102 (22) (2005) 7794–7799.
[118] D. Gusfield, Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology, Cambridge University Press, Cambridge, UK, 1997.
[119] J.A. Hartigan, M.A. Wong, Algorithm AS 136: A k-means clustering algorithm, Applied Statistics 29 (1979) 100–108.
[120] E. Hartuv, R. Shamir, A clustering algorithm based on graph connectivity, Information Processing Letters 76 (4–6) (2000) 175–181.
[121] X. He, H. Zha, C.H.Q. Ding, H.D. Simon, Web document clustering using hyperlink structures, Computational Statistics & Data Analysis 41 (1) (2002) 19–45.
[122] C. Hennig, B. Hausdorf, Design of dissimilarity measures: A new dissimilarity measure between species distribution ranges, in: V. Batagelj, H.-H. Bock, A. Ferligoj, A. Ziberna (Eds.), Data Science and Classification, Studies in Classification, Data Analysis, and Knowledge Organization, Springer-Verlag GmbH, Berlin, Germany, 2006, pp. 29–38.
[123] D.J. Higham, G. Kalna, M. Kibble, Spectral clustering and its use in bioinformatics, Journal of Computational and Applied Mathematics 204 (1) (2007) 25–37.
[124] A. Hlaoui, S. Wang, Median graph computation for graph clustering, Soft Computing — A Fusion of Foundations Methodologies and Applications 10 (1) (2006) 47–53.
[125] D.D. Hochbaum, D.B. Shmoys, A unified approach to approximation algorithms for bottleneck problems, Journal of the ACM 33 (3) (1986) 533–550.

[126] D.S. Hochbaum, Various notions of approximations: Good, better, best, and more, in: D.S. Hochbaum (Ed.), Approximation Algorithms for NP-hard Problems, PWS Publishing Company, Boston, MA, USA, 1997, pp. 346–398 (Chapter 9).
[127] K. Holzapfel, S. Kosub, M.G. Maaß, H. Täubig, The complexity of detecting fixed-density clusters, in: R. Petreschi, G. Persiano, R. Silvestri (Eds.), Proceedings of the Fifth Italian Conference on Algorithms and Complexity, CIAC, in: Lecture Notes in Computer Science, vol. 2653, Springer-Verlag GmbH, Berlin, Germany, 2003.
[128] J.E. Hopcroft, O. Khan, B. Kulis, B. Selman, Natural communities in large linked networks, in: Proceedings of the Ninth International Conference on Knowledge Discovery and Data Mining, KDD, ACM, New York, NY, USA, 2003.
[129] F. Höppner, F. Klawonn, R. Kruse, T. Runkler, Fuzzy Cluster Analysis: Methods for Classification, Data Analysis and Image Recognition, John Wiley & Sons, Inc., Hoboken, NJ, USA, 1999.
[130] T.-C. Hou, T.-J. Tsai, An access-based clustering protocol for multihop wireless ad hoc networks, IEEE Journal on Selected Areas in Communications 19 (7) (2001) 1201–1210.
[131] W.-L. Hsu, G.L. Nemhauser, Easy and hard bottleneck location problems, Discrete and Applied Mathematics 1 (1979) 209–216.
[132] H. Hu, X. Yan, Y. Huang, J. Han, X.J. Zhou, Mining coherent dense subgraphs across massive biological networks for functional discovery, Bioinformatics (Suppl. 1) (2005) 213–221.
[133] P. Jaccard, Distribution de la flore alpine dans la Bassin de Dranses et dans quelques regions voisines, Bulletin del la Société Vaudoisedes Sciences Naturelles 37 (1901) 241–272. cited in [122].
[134] A.K. Jain, R.C. Dubes, Algorithms for Clustering Data, Prentice-Hall, Englewood, NJ, USA, 1988.
[135] A.K. Jain, M.N. Murty, P.J. Flynn, Data clustering: A review, ACM Computing Surveys 31 (3) (1999) 264–323.
[136] K. Jain, V.V. Vazirani, Primal-dual approximation algorithms for metric facility location and k-median problems, in: Proceedings of the Fourtieth Annual Symposium on Foundations of Computer Science, FOCS, IEEE Computer Society, Washington, DC, USA, 1999.
[137] D.S. Johnson, C.R. Aragon, L.A. McGeoch, C. Schevon, Optimization by simulated annealing: An experimental evaluation. Part I, graph partitioning, Operations Research 37 (6) (1989) 865–892.
[138] E.J.L. Johnson, A. Mehrotra, G.L. Nemhauser, Min-cut clustering, Mathematical Programming 62 (1) (1993) 133–151.
[139] N. Kahale, A semidefinite bound for mixing rates of Markov chains, Random Structures and Algorithms 11 (4) (1998) 299–313.
[140] J. Kalcsics, S. Nickel, M. Schröder, Toward a unified territorial design approach: Applications, algorithms, and GIS integration, TOP 13 (1) (2005) 1–56.
[141] N. Kannan, S. Selvaraj, M.M. Gromiha, S. Vishveshwara, Clusters in α/β barrel proteins: Implications for protein structure, function, and folding: A graph theoretical approach, Proteins 43 (2) (2001) 103–112.
[142] R. Kannan, S. Vempala, A. Vetta, On clusterings — good, bad and spectral, Journal of the ACM 51 (3) (2004) 497–515.
[143] R.M. Karp, Reducibility among combinatorial problems, in: Proceedings of a Symposium on the Complexity of Computer Computations, IBM, Plenum, NY, USA, 1972.
[144] D. Kempe, F. McSherry, A decentralized algorithm for spectral analysis, in: L. Babai (Ed.), Proceedings of the Thirty-sixth Annual Symposium on Theory of Computing, STOC, ACM Press, New York, NY, USA, 2004.
[145] B.W. Kernighan, S. Lin, An efficient heuristic procedure for partitioning graphs, Bell System Technical Journal 49 (2) (1970) 291–308.
[146] S. Khuller, Y.J. Sussmann, The capacitated k-center problem, in: J. Díaz, M.J. Serna (Eds.), Proceedings of the Fourth Annual European Symposium on Algorithms, ESA, in: Lecture Notes in Computer Science, vol. 1136, Springer-Verlag GmbH, Berlin, Heidelberg, Germany, 1996.
[147] S. Kim, Graph theoretic sequence clustering algorithms and their applications to genome comparison, in: J.T.L. Wang, C.H. Wu, P.P. Wang (Eds.), Computational Biology and Genome Informatics, World Scientific Publishing Company, 2003, pp. 81–116 (Chapter 4).
[148] A.D. King, N. Przulj, I. Jurisica, Protein complex prediction via cost-based clustering, Bioinformatics 20 (17) (2004) 3013–3020.
[149] R.W. Klein, R.C. Dubes, Experiments in projection and clustering by simulated annealing, Pattern Recognition 22 (2) (1989) 213–220.
[150] J. Kleinberg, An Impossibility Theorem for Clustering, MIT Press, Cambridge, MA, USA, 2002.
[151] J.M. Kleinberg, S. Lawrence, The structure of the Web, Science 294 (5548) (2001) 1849–1850.
[152] J.M. Kleinberg, E. Tardos, Approximation algorithms for classification problems with pairwise relationships: Metric labeling and Markov random fields, Journal of the ACM 49 (5) (2002) 14–23.
[153] J.G. Klincewicz, Heuristics for the p-hub location problem, European Journal of Operational Research 53 (1991) 25–37.
[154] M. Kozdron, The discrete dirichlet problem. http://citeseer.ist.psu.edu/293959.html, April 2000.
[155] D.L. Kreher, D.R. Stinson, Combinatorial Algorithms: Generation, Enumeration, and Search, CRC Press, Boca Raton, FL, USA, 1998.
[156] P. Krishna, N.H. Vaidya, M. Chatterjee, D.K. Pradhan, A cluster-based approach for routing in dynamic networks, ACM SIGCOMM Computer Communication Review 27 (2) (1997) 49–64.
[157] S.R. Kumar, J. Novak, P. Raghavan, A. Tomkins, On the bursty evolution of blogspace, in: Proceedings of the Twelfth International World-Wide Web Conference, WWW, ACM Press, New York, NY, USA, 2003.
[158] M. Křivánek, J. Morávek, NP-hard problems in hierarchical-tree clustering, Acta Informatica 23 (3) (1986) 311–323.
[159] S. Lakroum, V. Devlaminck, P. Terrier, P. Biela Enberg, J.-G. Postaire, Clustering of the Poincare vectors, in: IEEE International Conference on Image Processing, vol. 2, IEEE, 2005.
[160] K. Lang, S. Rao, A flow-based method for improving the expansion or conductance of graph cuts, in: G.L. Nemhauser, D. Bienstock (Eds.), Proceedings of the Tenth International Conference on Integer Programming and Combinatorial Optimization, IPCO, in: Lecture Notes in Computer Science, vol. 3064, Springer-Verlag GmbH, Berlin, Heidelberg, Germany, 2004.
[161] V. Latora, M. Marchiori, Efficient behavior of small-world networks, Physical Review Letters 87 (19) (2001) 198701.
[162] V. Latora, M. Marchiori, A measure of centrality based on the network efficiency, Tech. Rep. cond-mat/0402050, arXiv.org, February 2004.
[163] G.F. Lawler, Intersections of Random Walks, Probability and its Applications, Birkhäuser, Boston, MA, USA, 1991.
[164] R.-C. Li, Accuracy of computed eigenvectors via optimizing a rayleigh quotient, Bit Numerical Mathematics 44 (3) (2004) 585–593.
[165] C.R. Lin, M. Gerla, Adaptive clustering for mobile wireless networks, IEEE Journal on Selected Areas in Communications 15 (7) (1997) 1265–1275.

[166] A.H. Lipkus, A proof of the triangle inequality for the tanimoto distance, Journal of Mathematical Chemistry 26 (1–3) (1999) 263–265.
[167] L. Lovász, Random walks on graphs: A survey, in: Bolyai Society Mathematical Studies, 2, in: Combinatorics, Pál Erdős is Eighty, vol. 2, Bolyai Mathematical Society, 1996, pp. 353–397.
[168] B. Luo, R.C. Wilson, E.R. Hancock, Spectral feature vectors for graph clustering, in: T. Caelli, A. Amin, R.P. Duin, M. Kamel, D. de Ridder (Eds.), Proceedings of the Joint IARP International Workshops on Syntactical and Structural Pattern Recognition and Statistical Pattern Recognition, in: Lecture Notes in Computer Science, vol. 2396, Springer-Verlag GmbH, Berlin, Heidelberg, Germany, 2002.
[169] B. Luo, R.C. Wilson, E.R. Hancock, Spectral clustering of graphs, in: G. Goos, J. Hartmanis, J. van Leeuwen (Eds.), Proceedings of the Tenth International Conference on Computer Analysis of Images and Patterns, CAIP, in: Lecture Notes in Computer Science, vol. 2756, Springer-Verlag GmbH, Berlin, Heidelberg, Germany, 2003.
[170] R.M. MacGregor, On partitioning a graph: A theoretical and empirical study, Ph.D. Thesis, University of California, Berkeley, CA, USA, 1978.
[171] H. Matsuda, T. Ishihara, A. Hashimoto, Classifying molecular sequences using a linkage graph with their pairwise similarities, Theoretical Computer Science 210 (2) (1999) 305–325.
[172] D.W. Matula, F. Shahrokhi, Sparsest cuts and bottlenecks in graphs, Discrete Applied Mathematics 27 (1–2) (1990) 113–123.
[173] F. McSherry, Spectral partitioning of random graphs, in: Proceedings of the Fourty-Second IEEE Symposium on Foundations of Computer Science, FOCS, IEEE Computer Society Press, Washington, DC, USA, 2001.
[174] F. McSherry, Spectral methods for data analysis, Ph.D. Thesis, University of Washington, Seattle, WA, USA, 2004.
[175] M. Meilă, J. Shi, Learning segmentation by random walks, in: T.K. Leen, T.G. Dietterich, V. Tresp (Eds.), Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems, NIPS, MIT Press, Cambridge, MA, USA, 2000.
[176] M. Meilă, J. Shi, A random walks view of spectral segmentation, in: Proceedings of the Eighth International Workshop on Artificial Intelligence and Statistics, Morgan Kauffman, San Francisco, CA, USA, 2001.
[177] Z. Michalewicz, D.B. Fogel, How to Solve it: Modern Heuristics, 2nd ed., Springer-Verlag GmbH, Berlin, Heidelberg, Germany, 2004.
[178] M. Mihail, C. Gkantsidis, A. Saberi, E. Zegura, On the semantics of internet topologies, Tech. Rep. GIT-CC-02-07, College of Computing, Georgia Institute of Technology, Atlanta, GA, USA, 2002.
[179] B.L. Milenova, M.M. Campos, O-cluster: Scalable clustering of large high dimensional data sets, in: Proceedings of the IEEE International Conference on Data Mining, ICDM, 2002.
[180] M.E. Newman, Finding community structure in networks using the eigenvectors of matrices, Physical Review E 74 (3) (2006) 036104.
[181] M.E.J. Newman, A measure of betweenness centrality based on random walks, Tech. Rep. cond-mat/0309045, arXiv.org, September 2003.
[182] M.E.J. Newman, Properties of highly clustered networks, Physical Review E 68 (2) (2003) 026121.
[183] M.E.J. Newman, The structure and function of complex networks, SIAM Review 45 (2) (2003) 167–256.
[184] M.E.J. Newman, Detecting community structure in networks, The European Physical Journal B 38 (2) (2004) 321–330.
[185] M.E.J. Newman, Fast algorithm for detecting community structure in networks, Physical Review E 69 (6) (2004) 066133.
[186] M.E.J. Newman, M. Girvan, Mixing patterns and community structure in networks, in: R. Pastor-Satorras, M. Rubi, A. Díaz Guilera (Eds.), Statistical Mechanics of Complex Networks: Proceedings of the XVIII Sitges Conference on Statistical Mechanics, in: Lecture Notes in Physics, vol. 625, Springer-Verlag GmbH, Berlin, Germany, 2003.
[187] M.E.J. Newman, M. Girvan, Finding and evaluating community structure in networks, Physical Review E 69 (2) (2004) 026113.
[188] A.Y. Ng, M.I. Jordan, Y. Weiss, On spectral clustering: Analysis and an algorithm, in: T.G. Dietterich, S. Becker, Z. Ghahramani (Eds.), Proceedings of the Fourteenth Conference on Advances in Neural Information Processing Systems, vol. 2, The MIT Press, Cambridge, MA, USA, 2002.
[189] M. O'Kelly, A clustering approach to the planar hub location problem, Annals of Operations Research 40 (1) (1992) 339–353.
[190] P. Orponen, S.E. Schaeffer, Locally computable approximations for spectral clustering and absorption times of random walks (in preparation).
[191] P. Orponen, S.E. Schaeffer, Local clustering of large graphs by approximate Fiedler vectors, in: S. Nikoletseas (Ed.), Proceedings of the Fourth International Workshop on Efficient and Experimental Algorithms, WEA, in: Lecture Notes in Computer Science, vol. 3505, Springer-Verlag GmbH, Berlin, Heidelberg, Germany, 2005.
[192] C.H. Papadimitriou, Computational Complexity, Addison Wesley, Reading, MA, USA, 1993.
[193] J.B. Pereira-Leal, A.J. Enright, C.A. Ouzounis, Detection of functional modules from protein interaction networks, Proteins: Structure, Function, and Bioinformatics 54 (1) (2003) 49–57.
[194] C.E. Perkins (Ed.), Ad Hoc Networking, Addison Wesley, Reading, MA, USA, 2001.
[195] J. Plesník, A heuristic for the p-center problem in graphs, Discrete and Applied Mathematics 17 (1987) 263–268.
[196] A. Pothen, H.D. Simon, K.-P. Liou, Partitioning sparse matrices with eigenvectors of graphs, SIAM Journal on Matrix Analysis and Applications 11 (3) (1990) 430–452.
[197] J. Puhan, T. Tuma, I. Fajfar, Spice for Windows 95/98/NT, Elektrotehnišski vestnik 65 (5) (1998) 267–271. Electrotechnical review, Ljubljana, Slovenia.
[198] H. Qiu, E.R. Hancock, Graph matching and clustering using spectral partitions, Pattern Recognition 39 (1) (2004) 22–34.
[199] J.M. Rabaey, The spice circuit simulator, eECS Department of the University of California at Berkeley. http://bwrc.eecs.berkeley.edu/Classes/ICBook/SPICE/.
[200] V.V. Raghavan, C.T. Yu, A comparison of the stability characteristics of some graph theoretic clustering methods, IEEE Transactions on Pattern Analysis and Machine Intelligence 3 (4) (1981) 393–403.
[201] R.Z. Ríos-Mercado, E. Fernández, A reactive GRASP for a sales territory design problem with multiple balancing requirements, Tech. Rep. PISIS-2006-12, Graduate Program in Systems Engineering, Universidad Autónoma de Nuevo León, San Nicolás de los Garza, Mexico, September 2006.
[202] A. Robles-Kelly, E.R. Hancock, Graph edit distance from spectral seriation, IEEE Transactions on Pattern Analysis and Machine Intelligence 27 (3) (2005) 365–378.
[203] K.A. Rytkönen, A spring-force visualization algorithm implemented in Java (2003), unpublished.
[204] M. Saerens, F. Fouss, L. Yen, P. Dupont, The principal components analysis of a graph, and its relationships to spectral clustering, in: J.-F. Boulicaut, F. Esposito, F. Giannotti, D. Pedreschi (Eds.), Proceedings of the Fifteenth European Conference on Machine Learning, ECML,

in: Lecture Notes in Computer Science, Springer-Verlag GmbH, Berlin, Heidelberg, Germany, 2004.

[205] S.E. Schaeffer, Stochastic local clustering for massive graphs, in: T.B. Ho, D. Cheung, H. Liu (Eds.), Proceedings of the Ninth Pacific-Asia Conference on Knowledge Discovery and Data Mining, PAKDD, in: Lecture Notes in Computer Science, vol. 3518, Springer-Velgar GmbH, Berlin, Heidelberg, Germany, 2005.

[206] S.E. Schaeffer, Algorithms for nonuniform networks, Ph.D. Thesis, Helsinki University of Technology, Espoo, Finland, April 2006.

[207] S.E. Schaeffer, S. Marinoni, M. Särelä, P. Nikander, Dynamic local clustering for hierarchical ad hoc networks, in: Proceedings of the IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks, SECON'06, International Workshop on Wireless Ad-hoc and Sensor Networks, IWWAN'06, subtrack, IEEE Communications Society, New York, NY, USA, 2006.

[208] R. Shamir, R. Sharan, D. Tsur, Cluster graph modification problems, in: Proceedings of the Twenty-eighth International Workshop on Graph-Theoretic Concepts in Computer Science, in: Lecture Notes in Computer Science, vol. 2573, Springer-Verlag GmbH, Berlin, Germany, 2002.

[209] J. Shi, J. Malik, Normalized cuts and image segmentation, IEEE Transactions on Pattern Analysis and Machine Intelligence 22 (8) (2000) 888–901.

[210] J. Šíma, S.E. Schaeffer, On the NP-completeness of some graph cluster measures, in: J. Wiedermann, G. Tel, J. Pokorný, M. Bieliková, J. Štuller (Eds.), Proceedings of the Thirty-second International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM, in: Lecture Notes in Computer Science, vol. 3831, Springer-Verlag GmbH, Berlin, Heidelberg, Germany, 2006.

[211] A. Sinclair, Algorithms for Random Generation & Counting: A Markov Chain Approach, Birkhäuser, Boston, MA, USA, 1993.

[212] K. Soumyanath, J.S. Deogun, On bisection width of partial k-trees, Congressus Numerantium 74 (1990) 45–51.

[213] D.A. Spielman, S.-H. Teng, Spectral partitioning works: Planar graphs and finite element meshes, in: Proceedings of the Thirty-seventh IEEE Symposium on Foundations of Computing, FOCS, IEEE Computer Society Press, Los Alamitos, CA, USA, 1996.

[214] D.A. Spielman, S.-H. Teng, Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems, in: L. Babai (Ed.), Proceedings of the Thirty-sixth Annual Symposium on Theory of Computing, STOC, ACM Press, New York, NY, USA, 2004.

[215] S.P. Strunkov, On weakly cospectral graphs, Mathematical Notes 80 (4) (2006) 590–592. translated from Matematicheskie Zametki 80 (4) pp. 627–629.

[216] J. Sucec, I. Marsic, Clustering Overhead for Hierarchical Routing in Mobile ad hoc Networks, in: Proceedings of the Twenty-first Annual Joint Conference of the IEEE Computer and Communications Societies, vol. 3, IEEE Computer Society Press, Los Alamitos, CA, USA, 2002.

[217] C. Swamy, D.B. Shmoys, Fault-tolerant facility location, in: Proceedings of the Fourteenth Annual ACM–SIAM Symposium on Discrete Algorithms, SODA, ACM, SIAM, 2003.

[218] B. Świercz, Ł. Starzak, M. Zubert, A. Napieralski, DMCS-SPICE circuit analysis. http://lux.dmcs.p.lodz.pl/~swierczu/java_gui.php.

[219] P.-N. Tan, M. Steinbach, V. Kumar, Cluster Analysis: Additional Issues and Algorithms, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005, pp. 569–650.

[220] T. Tanimoto, IBM Internal Report, November 17 1957.

[221] M. Thelwall, A web crawler design for data mining, Journal of Information Science 27 (5) (2001) 319–325.

[222] G.T. Toussaint, Proximity graphs for nearest neighbor decision rules: Recent progress, in: Proceedings of the Thirty-Fourth Symposium on Computing and Statistics, Interface-2002, The Interface Foundation of North America, Fairfax Station, VA, USA, 2002.

[223] E.R. van Dam, W.H. Haemers, Which graphs are determined by their spectrum? Linear Algebra and its Applications 373 (2003) 241–272.

[224] S.M. van Dongen, Graph clustering by flow simulation, Ph.D. Thesis, Universiteit Utrecht, Utrecht, The Netherlands, May 2000.

[225] L. Vargas Suáarez, R.Z. Ríos-Mercado, F. López, Usando GRASP para resolver un problema de definición de territorios de atención comercial, in: M. Arenas, F. Herrera, M. Lozano, J. Merelo, G. Romero, A. Sáanchez (Eds.), Proceedings of the IV Spanish Conference on Metaheuristics, in: Evolutionary and Bioinspired Algorithms, vol. 2, Granada, Spain, 2005 (in Spanish).

[226] V.V. Vazirani, Approximation Algorithms, Springer-Verlag GmbH, Berlin, Germany, 2001.

[227] S.E. Virtanen, Clustering the Chilean web, in: Proceedings of the First Latin American Web Congress, LAWEB, IEEE Computer Society, Los Alamitos, CA, USA, 2003.

[228] S.E. Virtanen, Properties of nonuniform random graph models, Tech. Rep. HUT-TCS-A77, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Espoo, Finland, May 2003.

[229] D. Vukadinović, P. Huang, T. Erlebach, On the spectrum and structure of Internet topology graphs, in: H. Unger, T. Böhme, A.R. Mikler (Eds.), Proceedings of Second International Workshop on Innovative Internet Computing Systems, in: Lecture Notes in Computer Science, vol. 2346, Springer-Verlag GmbH, Berlin, Heidelberg, Germany, 2002.

[230] T. Washio, H. Motoda, Multi relational data mining (MRDM): State of the art of graph-based data mining, ACM SIGKDD Explorations Newsletter 5 (1) (2003) 59–68.

[231] D.J. Watts, Small Worlds, Princeton University Press, Princeton, NJ, USA, 1999.

[232] R. Weber, P. Zezula, Is similarity search useful for high dimensional spaces? in: Proceedings of the Tenth International Workshop on Database and Expert Systems Applications, 1999.

[233] W.T. Williams, M.B. Dale, P. Macnaughton-Smith, An objective method of weighting in similarity analysis, Nature 201 (426).

[234] R. Wilson, X. Bai, E.R. Hancock, Graph clustering using symmetric polynomials and local linear embedding, in: British Machine Vision Conference, 2003.

[235] S.M. Wong, Y.Y. Yao, An information-theoretic measure of term specificity, Journal of the American Society for Information Science 43 (1) (1992) 54–61.

[236] W.-C. Wong, A.W. Fu, Incremental document clustering for web page classification, in: J. Qun (Ed.), International Conference on Information Society in the 21st Century: Emerging Technologies and New Challenges, The University of Aizu, Aizu-Wakamatsu, Fukushima, Japan, 2000.

[237] A.Y. Wu, M. Garland, J. Han, Mining scale-free networks using geodesic clustering, in: W. Kim, R. Kohavi, J. Gehrke, W. DuMouchel (Eds.), Proceedings of the Tenth International Conference on Knowledge Discovery and Data Mining, KDD, ACM Press, New York, NY, USA, 2004.

[238] F. Wu, B.A. Huberman, Finding communities in linear time: A physics approach, The European Physical Journal B 38 (2) (2004) 331–338.

[239] X.L. Xie, G. Beni, A validity measure for fuzzy clustering, IEEE Transactions on Pattern Analysis and Machine Intelligence (8) (1991) 841–847.
[240] Y. Xu, V. Olman, D. Xu, Clustering gene expression data using a graph-theoretic approach: An application of minimum spanning trees, Bioinformatics 18 (4) (2002) 536–545.
[241] J.-T. Yan, P.-Y. Hsiao, A new fuzzy-clustering-based approach for two-way circuit partitioning, in: Proceedings of the Eight International Conference on VLSI Design, IEEE, New York, NY, USA, 1995.
[242] B. Yang, J. Liu, An efficient probabilistic approach to network community mining, in: J. Yao, P. Lingras, W.-Z. Wu, M. Szczuka, N. Cercone, D. Slezak (Eds.), Rough Sets and Knowledge Technology, Second International Conference, RSKT 2007, 14–16 May, Toronto, Canada, in: Lecture Notes in Computer Science, vol. 4481, 2007, pp. 267–275.
[243] Q. Yang, S. Lonardi, A parallel algorithm for clustering protein–protein interaction networks, in: Workshops and Poster Abstracts of the 2005 IEEE Computational Systems Bioinformatics Conference, 2005.
[244] W.W. Zachary, An information flow model for conflict and fission in small groups, Journal of Anthropological Research 33 (1977) 452–473.
[245] C.T. Zahn, Graph-theoretical methods for detecting and describing gestalt clusters, IEEE Transactions on Computers C-20 (1) (1971) 68–86.
[246] O.R. Zaïane, A. Foss, C.-H. Lee, W. Wang, On data clustering analysis: Scalability, constraints, and validation, in: Proceddings of the Sixth Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining, PAKDD, in: Lecture Notes in Computer Science, vol. 2336, Springer-Verlag GmbH, Berlin, Heidelberg, Germany, 2002.
[247] H. Zanghi, C. Ambroise, V. Miele, Fast online graph clustering via Erdős-Rényi mixture, Tech. Rep. 8, Jouy-en-Josas/Paris/Evry, France, April 2007 (submitted for publication).
[248] S. Zhong, J. Ghosh, A unified framework for model-based clustering, Journal of Machine Learning Research 4 (2003) 1001–1037.
[249] A.A. Zoltners, P. Sinha, Sales territory alignment: A review and model, Management Science 29 (1983) 1237–1256.