Helsinki University of Technology Laboratory for Theoretical Computer Science
Research Reports 102
Teknillisen korkeakoulun tietojenkäsittelyteorian laboratorion tutkimusraportti 102
Espoo 2006 HUT-TCS-A102

ALGORITHMS FOR NONUNIFORM NETWORKS

Satu Elisa Schaeffer

TEKNILLINEN KORKEAKOULU
TEKNISKA HÖGSKOLAN
HELSINKI UNIVERSITY OF TECHNOLOGY
TECHNISCHE UNIVERSITÄT HELSINKI
UNIVERSITE DE TECHNOLOGIE D'HELSINKI

Helsinki University of Technology Laboratory for Theoretical Computer Science
Research Reports 102
Teknillisen korkeakoulun tietojenkäsittelyteorian laboratorion tutkimusraportti 102
Espoo 2006 HUT-TCS-A102

ALGORITHMS FOR NONUNIFORM NETWORKS

Satu Elisa Schaeffer

Dissertation for the degree of Doctor of Science in Technology to be presented with due permission of the Department of Computer Science and Engineering, for public examination and debate in Auditorium T1 at Helsinki University of Technology (Espoo, Finland) on the 28 of April, 2006, at 12 o'clock noon.

Helsinki University of Technology
Department of Computer Science and Engineering
Laboratory for Theoretical Computer Science

Teknillinen korkeakoulu
Tietotekniikan osasto
Tietojenkäsittelyteorian laboratorio

Distribution:
Helsinki University of Technology
Laboratory for Theoretical Computer Science
P.O.Box 5400
FI-02015 TKK, FINLAND
Tel. +358 9 451 1
Fax. +358 9 451 3369
E-mail: lab@tcs.tkk.fi
URL: http://www.tcs.tkk.fi/

ISBN 951-22-8118-X
ISSN 1457-7615

Painotalo Casper
Espoo 2006

Abstract: In this thesis, observations on structural properties of natural networks are taken as a starting point for developing efficient algorithms for natural instances of different graph problems. The key areas discussed are sampling, clustering, routing, and pattern mining for large, nonuniform graphs. The results include observations on structural effects together with algorithms that aim to reveal structural properties or exploit their presence in solving an interesting graph problem.

Traditionally networks were modeled with uniform random graphs, assuming that each vertex was equally important and each edge equally likely to be present. Within the last decade, the approach has drastically changed due to the numerous observations on structural complexity in natural networks, many of which proved the uniform model to be inadequate for some contexts.

This quickly lead to various models and measures that aim to characterize topological properties of different kinds of real-world networks also beyond the uniform networks. The goal of this thesis is to utilize such observations in algorithm design, in addition to empowering the process of network analysis. Knowing that a graph exhibits certain characteristics allows for more efficient storage, processing, analysis, and feature extraction.

Our emphasis in on local methods that avoid resorting to information of the graph structure that is not relevant to the answer sought. For example, when seeking for the cluster of a single vertex, we compute it without using any global knowledge of the graph, iteratively examining the vicinity of the seed vertex. Similarly we propose methods for sampling and spanning-tree construction according to certain criteria on the outcome without requiring knowledge of the graph as a whole.

Our motivation for concentrating on local methods is two-fold: one driving factor is the ever-increasing size of real-world problems, but an equally important fact is the nonuniformity present in many natural graph instances; properties that hold for the entire graph are often lost when only a small subgraph is examined.

Keywords: community structure, graph algorithm, graph clustering, graph data mining, graph similarity, local search, minimum spanning tree, network model, nonuniform network, random graph, random sampling, routing, scale-free network, shortest path algorithm, small-world network

Tiivistelmä: Työ tarkastelee tehokkaiden algoritmien kehitystä erilaisille verkko-ongelmille perustuen havaintoihin luonnollisten verkkojen rakenteellisista ominaisuuksista. Painopistealueita ovat otanta, ryvästys, reititys sekä hahmonlouhinta suurissa epätasaisissa verkoissa. Tulokset koostuvat niin havainnoista verkkojen rakenteellisten ominaisuuksien vaikutuksista kuin algoritmeista rakenteen selvittämiseen tai hyväksikäyttöön mielenkiintoisten verkko-ongelmien tehokkaaseen ratkaisuun.

Perinteisesti verkkomallinnus on perustunut oletukseen solmujen tasa-arvoisuudesta sekä kaarien tasajakautuneisuudesta solmujen välille. Kuluneen vuosikymmenen aikana ajattelutapa on muuttunut rajusti erilaisten verkkojen rakennetta koskevien havaintojen seurauksena. Tästä muutoksesta ovat saaneet alkunsa erilaiset mallit ja mittarit, joilla pyritään tunnistamaan ja luokittelemaan käytännön sovelluksien todellisia verkkoja ja niiden rakenteellisia ominaisuuksia tilanteissa, joissa uniformit satunnaisverkot eivät riitä.

Väitöskirjan tavoitteena on hyödyntää luonnollisten verkkojen rakenteesta tehtyjä havaintoja algoritmisuunnittelussa ja lisäksi tehostaa verkkojen rakenteen analysointia. Mikäli verkolla tiedetään olevan tiettyjä ominaisuuksia, voidaan sen tallennusta, käsittelyä, analysointia ja ominaisuuksien tunnistamista tehostaa.

Työssä painotetaan paikallisia menetelmiä, jotka välttävät sellaisen verkon rakennetta koskevan tiedon käyttöä, joka ei suoranaisesti vaikuta haettuun ratkaisuun. Esimerkiksi etsittäessä tietyn solmun rypästä, se lasketaan tutkimalla vaiheittain alkusolmun läheisiä solmuja käyttämättä tietoa muista verkon osista. Sama lähestymistapa tuottaa menetelmiä satunnaisotantaan ja tiettyjä ehtoja täyttävän virityspuun rakentamiseen ilman koko verkon samanaikaista käsittelyä.

Esitetty otantamenettely perustuu kahden Markovin ketjun yhdistämiseen siten, että toisella ketjuista on toivottu rajajakauma siinä missä toinen on nopeasti sekoittuva; kaikki siirtymät voidaan laskea paikallisesti. Työssä esitetään kaksi paikallista ryvästysmenetelmää, jotka soveltuvat parhaiten yhden tietyn alkusolmun rypään laskemiseen. Lähestymistapa hakuongelmaan on solmukohtaisen ja rajoitetun ”tähystyspuskurin” käyttö. Reitityspuiden laadinnassa yhdistetään perinteinen kaaripainojen summan minimointi tuloksena syntyvän puun astelukujakauman hallintaan, tavoitteena tuottaa puita, joissa solmujen väliset reitit ovat paitsi kevyitä, myös kaarien lukumäärä on pieni. Verkkojen louhinnan osalta painopiste on verkkojen samankaltaisuuden tehokkaassa likimääräisessä arvioinnissa niin suuntaamattomille ja painottamattomille kuin nimikoiduille ja suunnattuille verkoille.

Perusteluita paikallisiin menetelmiin keskittymiseen on paljon. Kaksi keskeisintä valintakriteeriä tämän työn näkökulmasta ovat sovellusaluiden verkkotehtävien jatkuvasti kasvava koko sekä useiden luonnollisten verkkojen rakenteellinen epätasaisuus: ominaisuudet jotka pätevät verkolle kokonaisuutena, katoavat usein kun tarkastelu rajoitetaan pieneen aliverkkoon.

Avainsanat: epäuniformi verkko, lyhimmän polun algoritmi, paikallinen haku, pieni maailma -verkko, reititys, ryvästys, samankaltaisuus, satunnaisotanta, satunnaisverkko, skaalaton verkko, tiedon louhinta, verkkoalgoritmi, verkkomalli, virityspuu

Contents

List of Figures

List of Tables

NOTATIONS

GENERAL MATHEMATICAL NOTATIONS

$A,\ B, \ldots$	*sets*
$a,\ b, \ldots$	*elements*
$\emptyset$	*empty* set
$\bar{A}$	*complement* of set A
$A \setminus B$	set A “minus” set B; $A \setminus B = \{a \in A \mid a \notin B\}$
$\mathbb{Z}$	integers
$\mathbb{R}$	real numbers
$[a, b]$	*closed* interval from a to b
(a, b)	*open* interval from a to b
$[a, b)$	half-open interval containing a but *not* b
$(a, b]$	half-open interval containing b but *not* a
$f(x) \sim g(x)$	*similar* functions; $\lim\limits_{x \to \infty} \dfrac{f(x)}{g(x)} = 1$
$x \propto y$	x is *proportional* to y
$\lvert A \rvert$	*order* (or *cardinality*) of set A, number of elements in set A
$[n]$	index set $\{1,\ 2, \ldots, n\}$
$a \wedge b$	logical *and* (*conjunction*); true if *both* a and b hold
$a \vee b$	logical *or* (*disjunction*); true if *either one* of a and b or both hold
$\mathbf{I}$	*identity* matrix $\mathbf{I} = [i]_{j,k}$, where $i_{j,k} = 1$ if and only if $j = k$, otherwise 0
$\mathcal{O}\left(f(x)\right)$	if function $g(x)$ *grows no faster than* $f(x)$, it has complexity $\mathcal{O}\left(f(x)\right)$; formally, $\exists x_0, k > 0$ s.t. $f(x) < k \cdot g(x)\ \forall x > x_0$

PROBABILITY-THEORETICAL NOTATIONS

$X, Y, \ldots$	random variables (r.v.)
$\Pr[X = x]$	*probability* of r.v. X having value x
$\mathrm{E}[X]$	*expected value* of X
$\mathrm{Var}[X]$	*variance* of X
$\mathrm{Cov}[X, Y]$	*covariance* of X and Y
ρ	correlation
$X \sim f$	X has *distribution* f
$\mathrm{Uniform}(a, b)$	*uniform* distribution over interval $[a,\ b]$
$\mathrm{Binom}(n, p)$	*binomial* distribution with n repetitions and success probability p
$\mathrm{Geom}(p)$	*geometric* distribution with success probability p
$\mathrm{Poisson}(\lambda)$	*Poisson* distribution with rate of change λ
$\mathrm{Exp}(\lambda)$	*exponential* distribution with rate of change λ
$\mathbf{M}$	*Markov chain*
$\mathbf{P}$	*transition matrix* $\mathbf{P} = [p]_{i,j}$ of a Markov chain
$\mathcal{S}$	*state space* of a Markov chain, $\mathcal{S} = \{1, 2, \ldots, N\}$
N	*order* of the state space; $\lvert\mathcal{S}\rvert = N$
$i, j, \ldots$	*states* in the state space, $i, j \in \mathcal{S}$
$p_{i,j}$	*transition probability* from state i to state j

GRAPH-THEORETICAL NOTATIONS

G	*graph* $G = (V, E)$
V	set of *vertices* in a graph
E	set of *edges* in a graph
n	number of vertices $\lvert V\rvert$ (*order* of a graph)
m	number of edges $\lvert E\rvert$ (*size* of a graph)
K_k	*complete* graph of order k
u, v, w	*vertices* in V
$\{v, w\}$	*edge* connecting vertices v and w
$\Gamma(v)$	*neighborhood* of a vertex v
$\deg(v)$	*degree* of a vertex v
$\Delta(G)$	*maximum* degree over the vertices of G
$\delta(G)$	*density* of a (sub-)graph G
$\mathrm{diam}(G)$	*diameter* of G (largest distance)
$g(G)$	*girth* of G (length of shortest cycle)
$\mathbf{A}$	*adjacency* matrix of a graph

ABBREVIATIONS AND ACRONYMS

a.s.	almost surely
BA	Barabási-Albert scale-free graph generation method
BFS	Breadth-first search
CWG	Chilean Web Graph
DBLP	Universität Trier's Computer Science Bibliography Database
DFS	Depth-first search
DGM	generation model of Dorogovtsev, Goltsev, and Mendes
ER	Erdős-Rényi graph generation model
FIFO	First-in first-out queue order
IMDB	Internet Movie Database
MC	Markov chain
MCMC	Markov Chain Monte Carlo method
MHT	Minimum-hop tree
MST	Minimum spanning tree
`ns-2`	Network Simulator
P2P	Peer-to-peer network
PTAS	polynomial-time approximation scheme
RDF	Resource Description Framework
RW	random walk
SCC	strongly connected component
s.t.	such that
TVD	total variation distance
URI	Universal Resource Identifier
URL	Universal Resource Locator
WS	Watts-Strogatz small-world graph generation model
WWW	World Wide Web

PREFACE

The research that lead to this thesis was supported by the Academy of Finland under grants 81120 (STADYCS, 2002–2003) and 206235 (ANNE, 2004–2006), as well as the Helsinki Graduate School in Computer Science and Engineering (2001–2002), the Nokia Foundation (2004), and the Rotary Foundation (2005).

The thesis would hardly exist without the encouragement and support of my supervisor, professor Pekka Orponen. Also Pekka Nikander has provided useful insight on ad hoc networks [307]. For their support and helpful comments, I thank Helger Lipmaa and Harri Haanpää; both were there for me either online or in person whenever I needed to clear my thoughts on some calculation or construction.

For not only for exchange of ideas, but also for "material" support, I thank Kosti Rytkönen for the assisted use of his graph visualization tool, Stefano Marinoni and Mikko Särelä for the implementation of an `ns-2` simulator for ad hoc networks, Barbara Poblete for the data set used in the Chilean web study [305], and Marco Gaertler for computing global clusterings to which we compared our local method [277].

Most of the year 2005 I spent in Santiago, Chile, working on this thesis at the University of Chile, where I was instructed by Ricardo Baeza-Yates and Carlos Hurtado. For helpful discussions and exchange of ideas during my stay in Chile, I also thank Javier Bustos, Álvaro Graves, and Claudio Gutiérrez.

I am in gratitude to the Laboratory for Theoretical Computer Science in Helsinki University of Technology, where the majority of this work was done, as well as to the Computer Science Department (Departamento de Ciencias de Computación) at University of Chile. I am especially obliged to thank the numerous system administrators whom I have constantly bothered both at TKK and in Chile while writing this thesis. For their unconditional support, I thank my dear friends and fellow students at the Department of Computer Science and Engineering. In addition, I thank Riku Saikkonen for a last-minute fix on an "ä".

Finally, I thank the two professors who pre-examined this thesis, Prof. Erkki Mäkinen (University of Tampere in Finland) and Prof. Sergey Dorogovtsev (University of Aveiro in Portugal and the Russian Academy of Sciences in St. Petersburg) for their valuable comments and observations.

This document has been typeset on `emacs` using LaTeX, most figures were either drawn with `xfig` (integrating LaTeX-commands into the figures) or generated directly into `Fig3.2`-format (occasionally just by hand). A tool called `fig2dev` was most helpful in image conversion. The data plots were prepared with `gnuplot`, using various scripts for the statistics — programming was certainly one of the most entertaining parts of this work. Should anyone be interested to use some software or scripts used in this work, please contact the author.

1 INTRODUCTION

Deciding which street to take to get to the airport the quickest, finding the website of an appropriate hotel for an upcoming business trip, and choosing a set of articles to read on a promising new technology are all problems that deal with networks. The modern society is largely based on networks, such as telecommunication networks, social networks, highways, flight routes, power grids, and water supply systems. The design, usage, and maintenance of these networks are tasks that affect the cost and reliability of services, which in turn reflect on the lives of the people who use them without much thought to the underlying network structure.

Any system that can be characterized as a set of inter-related entities allows an abstraction into a *network*. The entities are the *nodes* of the network and the interactions, dependencies, proximities, or other types of relations between them are captured in the *connections* of the network. In general, properties such as weights, labels, or classifications may be imposed on the nodes and connections alike. The connections may also be assigned a direction, for example to represent a one-way street in a road network.

The practical tasks involving networks are numerous in science and engineering, and the networks in question are larger than ever before. Traditionally network problems have been studied under the assumption that the networks' generation, growth, and connection patterns are governed by a random process, thereby making little or no use of information on the structure of the networks.

In *natural* networks, however, not all nodes have equal connectivity patterns. Some airports offer many more connections than others, and some people have more and stronger social contacts than others. By characterizing a network as natural, we simply suggest that the network in question arose in real-world context rather than as an artificial mathematical model, hence not restricting ourselves to biological networks.

In this thesis we study networks in which the nodes and their connection patterns are complex in the sense that the presence or absence of a connection could not be merely modeled by flipping a coin and where structural properties are not constant over the entire network. We call such networks *nonuniform*; the existing literature lacks a single, descriptive, and widely accepted term for such networks.

The study of nonuniform networks has become popular ever since the seminal paper of Watts and Strogatz [314] brought forward the term "small-world network", followed by the concept of "scale-free networks", initiated largely by Barabási's research group [25, 26]. In this thesis, we study the discovery and utilization of observations about nonuniform network structure from an algorithmic perspective.

The thesis is organized as follows. First, we briefly introduce the concept of nonuniform networks in Chapter 2, reviewing models and properties of nonuniform networks and discussing the effects of network structure on algorithmic behavior. Chapter 3 addresses the issue of sampling nonuniform networks. Chapter 4 presents and discusses methods for network clustering, also known as community discovery, which allows for more efficient handling of relevance queries, for example, on large nonuniform networks. Chapter 5 concentrates on searching and routing in nonuniform networks. In Chapter 6 we discuss graph data mining, after which we conclude the thesis and discuss possible continuations of this work in Chapter 7.

All run-time sensitive experiments reported in this thesis were ran on an AMD Athlon XP 1600 MHz workstation with 1,024 MB of main memory. With some computation-intensive experiments, the work was divided to heterogeneous workstations, in which case run-time comparisons have not been made.

The author's contributions in this thesis include sampling, clustering, path-finding, spanning-tree construction, and pattern-matching algorithms, together with experiments, comparisons to other work, and applications to network problems. The thesis partially overlaps with a former monograph [306], which we suggest to readers unfamiliar with the field of complex networks as an introduction; a good alternative is Newman's review article [237]. Material from an article co-authored by Pekka Orponen [249] has been used in Chapter 3, and in Chapter 4, material from other publications [250, 277, 284, 305, 307] respectively has been included. Contributions of collaborators that are discussed in this thesis are the following:

- In Section 4.2.2, the idea of using of Fiedler vectors for clustering and the derivation of the approximation formula are by Pekka Orponen [250].
- In Section 4.2.3, author's original reduction for RELATIVE DENSITY was clarified to its present form by Jiří Šíma [284].
- In Section 4.5.1, many practical concerns of the integration of the local clustering in hierarchical ad hoc networks are due to Pekka Nikander [307]; the `ns-2` simulations of the algorithm are based on an implementation by Stefano Marinoni and Mikko Särelä [280].

The thesis includes many plots and graphics made with `gnuplot`, some of which unfortunately are not sufficiently legible in black-and-white prints. For those plots that appeared risky in this sense, an effort was made to order the legend in decreasing order of the curves on the y-axis whenever possible. For the interested reader, we do recommend resorting to an electronic version of this thesis with zoomable color illustrations; it is available at

`http://www.tcs.hut.fi/Publications/series-a.shtml`

as well as in the electronic collections of the library of the Helsinki University of Technology at `http://lib.tkk.fi/Diss/isbn9512281198`.

2 NETWORKS AS GRAPHS

A mathematical formalization for a network is a *graph* G, defined as a pair of sets $G = (V, E)$. A good and comprehensive introduction to graphs is Reinhard Diestel's textbook [96].

The set V of *vertices* represents nodes of a network. The vertices, unless otherwise stated, are labeled by integers $1, 2, \ldots, n$. The number of vertices $n = |V|$ is the *order* of the graph. The symbols u, v, and w (with subscripts when necessary) are used to refer to a single specific vertex.

The connections of a network are represented by the set E of *edges*. Unless otherwise stated, an edge is assumed to be an *unordered pair of distinct vertices*, denoted simply by $\{v, w\}$. If all edges are unordered pairs, the graph is *undirected*. An edge $\{v, v\}$ is a *reflexive* edge, also called a *self-loop*. In a *weighted graph*, a weight function is defined that assigns a weight on each edge. In a *simple graph*, only one edge may exist between a given pair of vertices (but reflexive edges are allowed).

The edge count $|E| = m$ is the *size* of the graph. The *density* of a graph $G = (V, E)$ is defined as the ratio of the number of edges present to the maximum possible,

$$\delta(G) = \frac{m}{\binom{n}{2}}. \tag{2.1}$$

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

In many cases, instead of studying the graph G itself, it is useful to study its *complement*, which is a graph with the same vertex set, but with only those edges included in its edge set that are *not* present in E (usually excluding reflexive edges). Vertices among which no edges appear form an *independent set*. Any independent set in graph $G = (V, E)$ induces a clique in the complement of G.

If $\{v, w\} \in E$, we say that v is a *neighbor* of w. The set of neighbors for a given vertex v is called the *neighborhood* of v and is denoted by $\Gamma(v)$. A vertex v is a member of its own neighborhood $\Gamma(v)$ 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 $\mathbf{A}_G = (a^G_{v,w})$ where

$$a^G_{v,w} = \begin{cases} 1, & \text{if } \{v, w\} \in E, \\ 0, & \text{otherwise.} \end{cases} \tag{2.2}$$

For a multigraph, the elements of an adjacency matrix indicate the edge multiplicities instead of being binary. Another alternative for representing a graph is to form an *adjacency list* for each vertex, listing the neighbors it has. It depends on the application at hand which representation is the most practical. For *sparse graphs*, i.e., graphs with low density, the adjacency-list representation is more compact, whereas for *dense graphs*, i.e., high-density graphs, it is often more practical to store and process the adjacency matrix.

A graph where the vertex set can be partitioned into k nonempty non-overlapping subsets such that edges only appear across subsets and not within is a

k-*partite* graph. In particular, if the partition consists of two such subsets, the graph is *bipartite*.

The number of edges incident on a given vertex v is the *degree* of v and is denoted by $\deg(v)$. In a simple graph, the degree of a vertex is equal to the number of neighbors it has,

$$\deg(v) = |\Gamma(v)|. \tag{2.3}$$

The *maximum degree* of a graph $G = (V, E)$ is

$$\Delta = \Delta(G) = \max_{v \in V}\{\deg(v)\}. \tag{2.4}$$

The *average degree* is denoted $\bar{k}$ and is simply $2m/n$, as each edge has exactly two distinct endpoints.

Listing the degrees of all vertices in G, one obtains a *degree distribution* of the graph, fluently represented visually as a histogram and characterized mathematically by the probability distribution function that gives the probability for a randomly chosen vertex to have degree k. If all the vertices of a graph have the same degree k, the graph is k-regular. A 3-regular graph is also called a *cubic graph*.

A *path* from v to w is a sequence of edges in E starting at vertex v and ending at vertex w;

$$\{v, v_1\}, \{v_1, v_2\}, \ldots, \{v_{k-1}, v_k\}, \{v_k, w\}. \tag{2.5}$$

If such a path exists, v and w are *connected*. The *length* of a path is the number of edges on it, and the *distance* between v and w is the length of the *shortest* path connecting them in G. The distance from a vertex to itself is zero, as the path from a vertex to itself is an empty edge sequence; sometimes in the presence of reflexive edges more complex definitions of reachability between vertices become useful.

A path is said to be a *cycle* if $v = w$. A *simple* path is one that contains no cycle, i.e., the vertices $v, v_1, v_2, \ldots, v_k$ and w are all distinct. The *shortest path* between any pair of vertices is always simple.

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* and each group of vertices that are all connected by paths is called a (connected) *component* of the graph. In directed graphs, a component where each pair of vertices is connected by a path is called a *strongly connected* component (SCC).

A connected graph that has no cycles is *acyclic*; such graphs are called *trees* and they always have exactly $n - 1$ edges. A tree $T = (V, F)$ is a *spanning tree* of a graph $G = (V, E)$ if $F \subseteq E$.

A disconnected graph where each component is acyclic is called a *forest*. Graphs that are not trees or forests are said to be *cyclic*. The length of the shortest cycle in a cyclic graph is called the *girth* $g(G)$ of the graph.

The *maximum* distance over all vertex pairs in V is the *diameter* of the graph $G = (V, E)$, denoted by $\operatorname{diam}(G)$. The *average* distance of a graph G is denoted by $\mathcal{L}(G)$; it is also called the *average path length* of the graph. We only consider unordered pairs of distinct vertices, which yields a somewhat higher average than including also the n reflexive pairs that all have distance zero, but in the literature both definitions are used. In a disconnected graph, distance-based measures are only meaningful for each connected component independently, as it is not clear whether a disconnected vertex pair should have infinite distance or simply be excluded of the calculations, resulting in poorly comparable measures.

When the degrees of the vertices are known, the expected average distance can be theoretically obtained for certain families of graphs [70]. There are many terms that refer to the lengths of the shortest paths in a graph, including *characteristic path length* [314] (i.e., the average path length) and *graph geodesics* [320].

A *subgraph* $G^S = (S, F)$ of $G = (V, E)$ is composed of a set of vertices $S \subseteq V$ and a set of edges $F \subseteq E$ such that $\{v, w\} \in F$ implies $v, w \in S$; graph G is then called a *supergraph* of G^S. Complete subgraphs are called *cliques*.

In a graph $G = (V, E)$, an *induced subgraph* for a vertex set $S \subseteq V$ is a graph that has S as the vertex set and the edge set includes all such edges $\{v, w\}$ in E that have both of the vertices v and w (called the *endpoints* of the edge) included in the set S. We often use the symbol for the vertex set of an induced subgraph in place of the subgraph symbol as the vertex set unambiguously defines the induced subgraph in a graph $G = (V, E)$; for example, for the density of a subgraph $G^S = (S, F)$, we may write either $\delta(S)$ or $\delta\left(G^S\right)$.

A partition of the vertices V of a graph $G = (V, E)$ into two sets S and $V \setminus S$ is called a *cut* and is denoted by $(S, V \setminus S)$. The number of edges that connect a vertex in S to a vertex in $V \setminus S$ is called the *cut size* and is denoted by $c(S, V \setminus S)$.

Two graphs $G = (V, E)$ and $P = (V_P, E_P)$ are *isomorphic* if there exists a bijective mapping $\phi : V \to V_P$ that *preserves* the adjacency relation, i.e.,

$$\{v, w\} \in E \Leftrightarrow \{\phi(v), \phi(w)\} \in E_P. \tag{2.6}$$

This means that the graphs are identical except for the vertex labels. The bijection ϕ is called a *graph isomorphism*.

A graph with edges formed by *ordered* pairs is a *directed* graph and each directed edge $\langle v, w\rangle$ has a *source* (or start) vertex v from which the edge leads to a *target* (or end) vertex w. For directed graphs, we distinguish between the *out-degree* of a vertex

$$\deg_{\text{out}}(v) = |\{\langle v, w\rangle \mid w \in V\}| \tag{2.7}$$

and the *in-degree*

$$\deg_{\text{in}}(v) = |\{\langle w, v\rangle \mid w \in V\}|. \tag{2.8}$$

The above definitions are straightforward to generalize to directed graphs; in defining density, the maximum number of edges is doubled, for paths, the edge directions must agree, and for degree distributions, the two types of degrees are often dealt with separately.

2.1 GENERATION MODELS

The mathematical study of graphs is known as *graph theory* and originates from the works of Leonhard Euler in the 18th century. When the applications of networks started to become more eminent in the late 1950s, the study of *random graphs* became popular.

In 1959, Gilbert [133] proposed the following model to generate graphs of arbitrary order: fix the graph order n and choose a probability p_e. Out of the possible $\binom{n}{2}$ unordered vertex pairs, include each one as an edge in the graph G uniformly at random with probability p_e. Hence, the expected number of edges in the graph is

$$\mathrm{E}\left[m\right] = p_e \binom{n}{2}. \tag{2.9}$$

Another approach was taken by Erdős and Rényi [109, 110], who instead fix both n and m and select the set of m edges out of all possible $\binom{n}{2}$ unordered pairs uniformly at random. The former model is known as the $\mathcal{G}_{n,p}$ model and the latter as the $\mathcal{G}_{n,m}$ model. We refer to these kinds of graphs as *uniform random graphs*, occasionally using the widely-spread abbreviation ER for Erdős-Rényi graphs, although it is also used in the literature to refer to Gilbert's $\mathcal{G}_{n,p}$ model. For more information on the properties of these models, we recommend for example Bollobás' book on random graphs [39].

These uniform random graph models were *not* intended to capture properties of real-world network problems, as emphasized by Erdős and Rényi [110] already in 1960, but rather to analyze the existence of certain substructures and the behavior of graph algorithms, such as finding shortest paths, maximal cliques, or minimal colorings of graphs. Nevertheless, the need for generation models in application areas resulted in wide adoption of uniform random graphs as models of real-world networks, often with modification such as placing the vertices on a plane and using connection probabilities proportional to Euclidean distance [315].

In 1998, Watts and Strogatz [314] threw the employment of uniform random graphs as models of real-world networks under re-evaluation, reporting observations on natural network data that were in vast disagreement with the uniform models. The disagreeing property studied by Watts and Strogatz measures the density of subgraphs induced by vertex neighborhoods:

Definition 2.1. The *clustering coefficient* of vertex v is

$$\mathcal{C}\left(v\right) = \delta\left(\Gamma\left(v\right)\right), \tag{2.10}$$

i.e., the density of the subgraph induced by the neighbors of v. The clustering coefficient of a graph $\mathcal{C}\left(G\right)$ is the *average* of the clustering coefficients over the vertex set.

Watts and Strogatz observed that the clustering coefficient was usually considerably higher for natural network data than for uniform random graphs with the same order and size.

In addition to the clustering coefficient, Watts and Strogatz [314] also studied the lengths of the shortest paths connecting two vertices. For uniform random graphs, the average path length grows slowly with the order [70],

$$\mathcal{L}(G) \sim \frac{\ln n}{\ln \bar{k}}. \tag{2.11}$$

From the construction parameters, for $\mathcal{G}_{n,p}$ it holds for the average degree that $\bar{k} = p(n-1)$ and for $\mathcal{G}_{n,m}$ clearly $\bar{k} = \frac{2m}{n}$. Watts and Strogatz observed that for natural networks, $\mathcal{L}(G)$ was almost as small as for uniform random graphs.

Based on these observations, Watts and Strogatz suggested a toy model (referred to as the WS model) for constructing networks that match the clustering coefficient and average path length of natural graphs. They fix the graph order n and place the vertices on a circle. An initial lattice is formed by connecting each vertex to the k nearest vertices along the circle on both sides. Such a lattice structure produces the high clustering coefficient. With the goal of introducing small average path length to the graph, Watts and Strogatz choose a small *rewiring* probability p_e, and for each vertex v, with probability p_e *rewire* the other endpoint of each of the edges incident on v uniformly at random over V. See Figure 2.1 for an illustration.

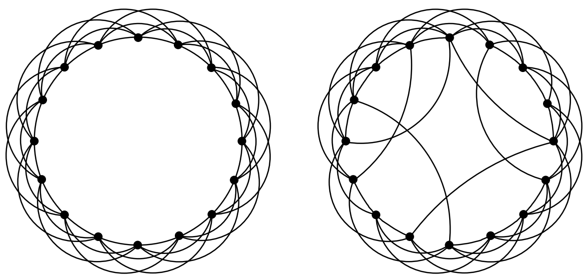


Figure 2.1: On the left, the initial lattice graph of Watts and Strogatz with $n = 16$ and $k = 3$; on the right, a rewired version with small p_e (adapted from [314]).

The model of Watts and Strogatz [314] has its weaknesses, inspired by which several modifications and alternatives have been proposed [189, 242, 243], but it serves to show that for small values of p_e, the introduction of rewired edges rapidly drops the average path length almost to the expected level of uniform random graphs of same order and size, but the highly clustered lattice structure keeps the average clustering coefficient high for a relevant regime before it drops as p_e is increased above a critical level. Graphs with these two properties — a high clustering coefficient and a low average path length — are now commonly known as *small-world networks*, a term coined by Watts and Strogatz [314].

The concept of small-world networks became quickly popular and related studies on natural network properties were initiated by many researchers. In 1999, Barabási and Albert [25, 26] came forward with another observation on natural networks that disagreed with both the uniform random graphs and the numerous variations that had sprung off the model of Watts and Strogatz.

The observation made by the research group of Barabási deals with the degrees of vertices in natural networks. For uniform random graphs, the degree of a vertex follows the binomial distribution Binom $(n-1, p_e)$, yielding a Poissonian distribution for the number of vertices with given degree k [39]. Calculating and plotting the degree distributions for the original small-world network model (or a variation) shows a form similar to the Poissonian: most vertices have a degree close to the average degree, and both tails of the distribution decay rapidly around the mean value. When plotting the degree distribution of real-world networks, Barabási's research group discovered forms strikingly distinct from Poissonian but much alike across different types of natural networks. All the observed distributions had a persistent right tail. Also, when the data was plotted on a log log-scale, the majority of the data set could be approximated with straight lines with almost the same slope.

It had turned out that the average degree of a natural network was not at all characteristic of the degrees of the included vertices, and practically all graphs based on natural data contained a few special vertices with extremely large degrees. These high-degree vertices are known as *hubs* and such distributions are called *scale-free* distributions as they could be more or less crudely be approximated by a *power law*[1] of the form

$$\Pr[k] \sim k^{-\gamma}, \tag{2.12}$$

meaning that the probability that a randomly chosen vertex has degree k is proportional to $k^{-\gamma}$, possibly with multiplicative and additive constants and some noise in the beginning and the end of the distribution. Similar observations were made on several different structural properties of the Internet [112], after which it became trendy to plot nearly any data set on a log log-scale and fit a line by linear regression, followed by suggestions of more sophisticated functions that provide better fits to the data. The key finding was nevertheless not the exact function, but the presence of such hubs in various natural networks, often forming a hierarchy of hubs and causing a small diameter and average path length. However, many of such power-law measurements made suffered from sampling bias [77], the reasons of which are discussed in Chapter 3.

Networks with such approximate degree distributions are widely known as *scale-free networks*, also the term *power-law graph* is used. Also this observation launched a wave of generation model proposals. The method suggested by Barabási and Albert uses a small seed graph to grow a graph of order n by adding one vertex at a time, assigning a fixed number of *outgoing* edges to each added vertex v and choosing the other endpoint for each of these edges with probability proportional to the *current degree* of the vertices that are already present in the graph, linking the edges with a higher probability to vertices that already have a high degree. This principle is known as *preferential attachment*; ideas similar to preferential attachment, also known as the "rich get richer" model, can be traced to the 1950s [285]. Generation models based on this idea are often called Barabási-Albert (BA) models.

Many variations of preferential attachment generation models have been proposed, for example to introduce clustering into the edge creating process [154] as it is not captured by the pure preferential attachment rule alone. The mathematics of the proposal were open to questioning, initiating more

[1] A review of power laws and related distributions is given by Mitzenmacher [229].

formal approaches [41, 83]. Also methods for matching a given degree sequence have been suggested [223, 230]. Ferrer i Cancho and Solé [116] observed that simultaneous minimization of density and average distance leads to such distributions, which may well explain why real-world networks tends to have these properties, as their natural evolution is in a sense a decentralized optimization process.

The observations of network structure in the real world did not stop there, but were followed by analysis of other properties, such as degree-degree correlations [37, 304], i.e., whether vertices of high degree tend to be connected to other high-degree vertices — known as *assortative* correlation — or rather to low-degree ones (*disassortative* correlation). It was also observed that clustering is often a consequence of a *hierarchical* topology [27]. Further analysis of clustering properties is given by Newman [236].

As many natural graphs are not static but instead evolve over time, also observations of the dynamics of structural properties are of interest. It was recently discovered by Leskovec *et al.* [204] that the density of some natural graphs tends to grow (in the sense that the average degree of the graph grows slowly over time, called *accelerated growth* [221]) and that the diameter decreases, both of which are observations that contradict previous assumptions — earlier generation models for matching natural network data were constructed to capture a constant average degree (a fixed number of edges added by preferential attachment per vertex [25]) and a slowly increasing diameter (related to the slow growth of the average path length [10, 314]). Leskovec *et al.* propose generation models to match their observations, and it can be expected that numerous variations of and alternatives to these models will be proposed in the near future by the natural network research community.

A recent survey [90] summarizes many of the most frequently used measurements of network structure emerging from the aforementioned studies and related work. Also, it has been proven useful to take advantage of structural observations, such as the presence of small separators,[2] on natural graphs [33] for applications such as compressing large graphs [34].

As a result of this sudden interest in properties and models of natural networks, there now exists a cornucopia of generation models to meet the observations of almost any imaginable natural network [102, 103, 237, 306]. More models are suggested on a monthly if not weekly basis, an up-to-date review on the models is hard to find. We use the term *nonuniform random graph* to refer to families of graphs for which the edge placement is not uniform over the vertex set, yielding interesting structural properties, such as high clustering or the presence of hubs. The term *nonuniform network* encompasses randomly generated nonuniform graphs as well as natural networks that are modeled by graphs with nontrivial structural properties.

2.2 ALGORITHMIC IMPLICATIONS OF NETWORK STRUCTURE

Algorithms can be evaluated based on the amount of computation they need to produce a solution for a given problem instance. The amount of com-

[2] A *separator* of a graph $G = (V, E)$ is a set of edges the removal of which splits the graph into components.

putation can be measured by the number of elementary operations used or by the running time of the algorithm. For many problems, the amount of computation needed depends heavily on the problem instance. An instance that requires much time on *any* algorithm for a specific problem is said to be a *hard instance* of that problem, although the concept of hardness is tricky and difficult to define in a generally satisfactory manner.

The traditional approach to the design and analysis of algorithms is to assume each problem instance equiprobable, although it is known for many problems that the hard instances seem to form only a specific subset of the instance space. This is the case, for example, for the satisfiability problem SAT, where the task is to find a *truth assignment* to a set of variables such that a given *logical formula*, composed of these variables, negations, disjunctions, and conjunctions, evaluates to true. The logical formulas are called SAT-instances. If the formula is composed of three-literal disjunctions (called *clauses*) that are joined by conjunctions, it is called a 3SAT-instance. For the decision problem 3SAT, using a specific random instance generator, there is a clear region of the clauses-to-variables ratio in which randomly generated instances are much harder to solve than for other ratios [228].

The phenomenon of a rapid transition from an easy regime to a hard regime or vice versa when adjusting the generation parameters of instances of a (combinatorial) problem is known as a (combinatorial) *phase transition* [217]. Work is undergoing to identify phase transitions for graph problems and such thresholds (with respect to density measures) have been observed for graph coloring [197] and vertex cover [29]. There are conflicting reports on whether the Hamiltonian cycle problem, where the task is to determine whether a given graph $G = (V, E)$ contains a cycle of length n that visits each vertex in V, exhibits a phase transition [124, 125, 303].

For problems involving graphs, it is known that restricting the family of graphs may yield an easier problem than the general one. Bounding the maximum degree $\Delta(G)$ from above allows for more efficient exact and/or approximation algorithms — for example the **NP**-complete clique problem where the task is to determine whether a given graph contains a complete subgraph of order k is solvable in $\mathcal{O}\left(n^{k+1}\right)$ time if it is known that $\Delta(G) \leq k$, as only subsets of vertices of order $(k+1)$ or smaller need to be examined.

Restricting to *planar graphs*, i.e., graphs that can be drawn on a plane without having any of the edges cross, allows solving for example the *graph isomorphism* problem in linear time, even though for general graphs no polynomial algorithm is known (the problem is known to be in **NP**, but completeness is not believed to hold) [165].

The problem of determining whether one graph contains the other as a subgraph is called *subgraph isomorphism* [301] and is known to be **NP**-complete [130]. The subgraph isomorphism problem, along with many other generally hard problems, is efficiently solvable for planar graphs [108]. Simplifications, special cases, other variants of isomorphism problems and algorithms (both exact and approximate) have been under much study [122, 222].

It has been acknowledged for a couple of decades that comparing the behavior of "competing" graph algorithms for a specific problem by generating uniform random instances and observing the behavior of the algorithms does *not* give informative results [50], which suggests that analyzing them for uniform random instances might not be the most fruitful approach either. Another statement towards the significance of structural properties in algorithm design and analysis is the suggestion of Garay *et al.* [129] to characterize the time complexity of a graph algorithm in terms of the diameter instead of the order of the input graph. They study the distributed construction of *minimum spanning trees* (MST) for undirected weighted graphs[3]; they also point out that the graph diameter has been found to be the governing factor of the complexity of *leader election*, which is the problem of choosing one vertex $v \in V$ to have a special role in some process that follows the leader election, such as spanning tree formation.

Garay *et al.* [129] bring forth the question of whether it is possible to determine which structural properties determine the complexities of fundamental graph problems and employ this information in designing better graph algorithms. Our work addresses essentially the same question, embedding it into the study of natural networks: given the observations that natural networks have nonuniform properties, whether and how this information could be used to design efficient algorithms for networks that are known to have a certain property, such as a scale-free degree distribution or a small-world topology? It is also of interest whether instances with certain properties fall into the easy or the hard regime for problems with observed or conjectured phase transitions. In this section we review experiments and observations on the behavior of algorithms on nonuniform graphs.

A straightforward application area is the study of *epidemic spreading* in a population of interacting individuals: the vertices represent the individual and an edge is placed between two individuals if they are in contact. The structure of these interactions, referred to as *network topology*, has been found to have significant impact on epidemic spreading. Hence making structural observations of the interaction patterns aids in preventing or limiting epidemic outbreaks. An effect of a small-world topology has been observed [231, 251] as well as implications of having a scale-free topology [255]. An efficient *vaccination scheme* known as the *acquaintance strategy* for scale-free networks [79] works as follows: choosing a uniform random sample and then a random neighbor of each included vertex results in a sample where high-degree vertices are more probable to be included. The effects of degree correlations on epidemic spreading have been studied by Boguñá *et al.* [38].

In common terms, if one wishes to vaccinate people with many social contacts, instead of simply vaccinating randomly selected people (even if one asked them for subjective evaluations on how "connected" they are, which would not be reliable), vaccinating people who were named as friends by randomly selected people gives effective results in preventing epidemics, as people with many social contacts are more probable to be "nominated" for vaccination and hence become less likely to expose their numerous friends to the epidemics.

Another property of practical interest is the *robustness* of networks, a concept which entails for example the decomposition pattern that a network exhibits when vertices or edges are removed. The *attack tolerance* of scale-free networks with redundancy is higher than that of networks with redundancy

[3] A *minimum* spanning tree is a spanning tree for which the sum of the weights of all included edges is minimum, i.e., no spanning tree exists with smaller total edge weight.

only [11], meaning that the surviving portion of a redundant scale-free network remains more functional under random vertex or edge eliminations, although a malicious adversary with knowledge of the graph topology may quickly destroy a scale-free network by eliminating only the hubs. Also theoretical results on the robustness of scale-free graphs are available [40].

Searching for a particular vertex in a given graph is a problem with numerous real-world applications. Small-world networks can be hard to search with local methods even though paths are short [132, 310], but the same topological property can also be turned into an advantage in, for example, searching the World-Wide Web [4]. This has also been done for scale-free topologies [5]. Finding the shortest path between two vertices in a graph has also been studied in the framework of nonuniform networks [182, 184]. The effect of the topological properties on the functioning of peer-to-peer (P2P) networks is a problem of interest and under investigation [253, 327]; such networks will be discussed in more detail in Chapter 5.

The behavior of *random walks*, i.e., traversals that select the next vertex uniformly at random from the neighbors of the current vertex, has many implications in algorithm design and applications. It has been studied in different network topologies; small changes to the completely blind uniform walk can have drastic effects on how many distinct vertices does such a walk visit on average [294, 325]

2.2.1 Network properties and computational load

The running time of an algorithm on a given instance is not easily predicted. In many cases, the closer the density is to 0.5, the longer the computation takes, as a large number of edges is slower to process than a small number (and with density much larger than 0.5, the complement has a density smaller than 0.5, and many problems can be solved also through the complement).

In this section demonstrate the complex relationships between graph structure, different measures such as the clustering coefficient and average path length, and the running time of an algorithm. We use the COLOR/02/03/04 data set [168] for *graph coloring* combined with known feasible color counts and the running times obtained on these graphs with certain heuristic algorithms [63, 64]. In graph coloring, the task is to assign a *color* to each vertex such that no two neighbors share a color, minimizing the number of colors used. The minimum number of colors needed for a given graph G is called the *chromatic number* $\gamma(G)$ of the graph. For graph coloring, small-world graphs appear even harder instances than uniform random graphs [310]. We made similar observations [306] for the CLIQUE problem.

It is acknowledged that although many properties of networks may be measured, it is not clear which set of measurements would be the most informative for a specific application, as many of the possible measures may be correlated or otherwise dependent of each other [90]. For demonstrative purposes, we performed a small-scale statistical analysis on a set of 86 connected and cyclic graphs from the aforementioned data set, some properties of which are shown in Table 2.1. We only included those instances that had girth and diameter defined for ease of comparison on the statistics (having infi-

Table 2.1: Some descriptive statistics of 86 connected cyclic graphs of the [168] data set for which upper bounds for the chromatic number are known, $\gamma_{\text{ub}}(G) \geq \gamma(G)$, and running times (in seconds) $\mathcal{T}(G)$ of a graph coloring algorithm.

Measure	$\mathcal{C}(G)$	$\gamma_{\text{ub}}(G)$	diam (G)	m	$g(G)$	$\Delta(G)$	n	$\mathcal{L}(G)$	$\mathcal{T}(G)$
Mean	0.31	20.98	3.98	35664	3.03	143.0	624.24	2.45	145.25
Std. dev.	0.27	29.77	2.79	80402	0.5830	167.8	997.93	1.19	426.94
Minimum	0	3	2	20	2	4	11	1.03	0
Median	0.34	9	3	5724	3	94	250	2.12	12.5
Maximum	0.97	224	15	449449	4	924	5231	6.94	2693

nite diameter or girth included would be impractical) and aiming to obtain meaningful correlations.

Possible linear dependencies between two random variables X and Y can be studied through their *correlation*,

$$\rho(X, Y) = \frac{\text{Cov}[X, Y]}{\sqrt{\text{Var}[X]\,\text{Var}[Y]}} \in [0,\ 1]. \tag{2.13}$$

Correlation is zero for *independent* random variables and one for *linearly* dependent variables, i.e., $X = aY + c$ for $a, b, c \in \mathbb{R}$. Rank correlations deal with the ranks of the values when ordered by magnitude instead of using the values directly. All correlations aim to measure the extent to which one random variable may be explained by observing the value of another when assuming linear dependency. Table 2.2 shows the Pearson correlations (assuming Gaussian distribution for both variables) as well as the Spearman rank correlations (a non-parametric method, hence better suited for this task) [214] over the data set.

The edge count and maximum degree are correlated with the color count of the graph as is to be expected — introducing more edges to a graph usually implies that a higher number of colors is needed to color the modified version. Also the presence of high-degree vertices, especially those with high clustering coefficients, tends to cause high color counts. Similarly the running time is correlated with edge count (as the input must be read in any case, regardless of the network topology) and maximum degree. The other factors do not significantly correlate with clustering, although a higher clustering coefficient is related to a larger color count. Diameter naturally correlates with average path length, being its upper bound.

As stated above, the size of the graph (i.e., the edge count) governs the time needed for an algorithm to read the input and can therefore be expected to explain the runtime well. More complex attempts to explain the runtime in terms of the other properties by means of linear regression are unsuccessful, as the edge count alone gives a relatively high coefficient of determination, $R^2 = \rho^2 = 0.8471$, and adding more independent variables causes no statistically significant improvement in the model. Using only the edge count to explain the runtime works well for running times up to 500 seconds, but fails for instances with high edge counts, as shown in Figure 2.2. This effect could be caused by the experimental setup rather than some properties

Table 2.2: The Pearson (top) and the Spearman rank (bottom) correlations of some measures of graph structure together with the running time $\mathcal{T}(G)$ and the lowest known color count $\gamma_{\text{ub}}(G)$ of the graph. The value is in boldface if $|\rho| \geq 0.5$ for improved legibility.

Measure	$\mathcal{C}(G)$	$\gamma_{\text{ub}}(G)$	$\text{diam}(G)$	m	$g(G)$	$\Delta(G)$	n	$\mathcal{L}(G)$
$\gamma_{\text{ub}}(G)$	**0.6286**							
	0.7209							
$\text{diam}(G)$	0.0050	-0.2083						
	-0.1465	-0.2693						
m	0.4426	**0.8210**	-0.1621					
	0.4678	**0.8301**	-0.1747					
$g(G)$	-0.4490	-0.1267	0.0149	-0.0354				
	-0.5614	-0.3986	0.2143	-0.2549				
$\Delta(G)$	0.4303	**0.7670**	-0.2252	**0.8015**	-0.0392			
	0.4058	**0.8094**	-0.1593	**0.9258**	-0.1361			
n	0.1297	0.2737	-0.0545	**0.6494**	0.0132	**0.5496**		
	0.1720	**0.5803**	0.1405	**0.8824**	-0.0034	**0.8266**		
$\mathcal{L}(G)$	-0.2295	-0.3198	**0.8390**	-0.2217	0.2209	-0.2663	-0.0289	
	-0.4548	**-0.5016**	**0.8833**	-0.3013	0.3424	-0.2729	0.0896	
$\mathcal{T}(G)$	0.3498	**0.7540**	-0.1171	**0.9213**	-0.0230	**0.7156**	**0.5536**	-0.1702
	0.2171	**0.5261**	-0.1860	**0.7870**	-0.1553	**0.7174**	**0.7526**	-0.2166

of the instances themselves. For example, the use of internal versus external memory may change when the instance grows above a certain size.

Hence we conclude that the factors that determine the running time of an algorithm for a given graph are not easily interpreted by looking at a set of global measures of the instance other than the size (which naturally affects the run time as most algorithms read the input before processing the instance), but rather a consequence of the graph structure as a whole.

In algorithm design, assuming that each edge is equally likely to occur in a typical instance is not feasible in light of the evidence on the structure of real-world problems being far from uniform. Instead one could take advantage of assumptions regarding the structural properties of the data set whenever the source of the natural data is known. It is an advantage to know whether the instances dealt with in an application area typically have a scale-free degree distribution or certain degree-degree correlations. For example, it has been observed that for many communication networks, power laws govern several different properties [112] and significant correlation patterns can be found [254, 304].

If no information on the source of the data is available or no educated guesses on the structural properties can be made, a quick approximate computation of some central properties can be used, for example, using a sample of the entire network to speed up the property-discovery phase. The goal of this thesis is to motivate taking structural properties of the application data into account in algorithm design, demonstrate with some examples the benefits of such an approach.

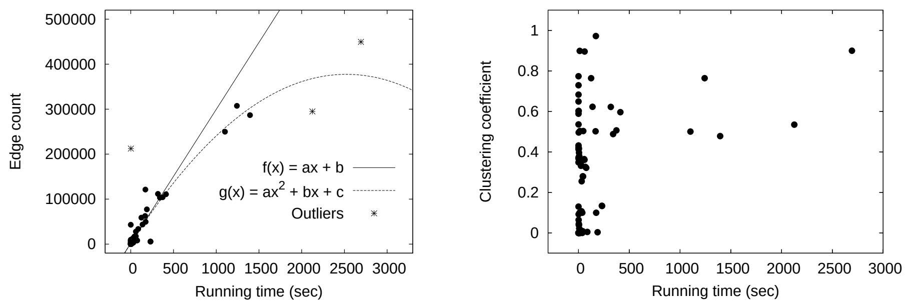


Figure 2.2: Scatter plots of running time versus edge count (left) and clustering (right). The edge count predicts the running time well for most small instances with fewer than 150,000 edges, but fails for most of the larger ones, as shown in the leftmost scatter plot. The clustering coefficient is a poor predictor alone, as shown in the rightmost scatter plot, but combined with edge count in linear regression helps somewhat in predicting the runtime.

2.2.2 Nonuniformity of natural networks

As an example of a nonuniform network, we discuss the WordNet [224] graph. The data set we used was extracted from an RDF schema [309] of the WordNet, forming a graph with 468,951 vertices representing words of the English language and 1,084,317 edges representing different types of semantical similarities among the words, such as two words being synonyms or of the opposite meaning. It is noteworthy that this graph, as most other natural networks, is sparse, with $\delta(G) \approx 9.86 \cdot 10^{-6}$.

In order to demonstrate the structural variability in a natural graph, we studied some properties of the WordNet graph. The degree distribution is shown in Figure 2.3. It shows a clear scale-free structure in the middle-region of the plot that can be quite well approximated with a line. The average degree of the graph is 4.6, but this is not a descriptive quantity, as 84.4 percent of the vertices have a degree below the average, placing the average value well above the third quartile. The median degree is 3. Simply judging by the average and/or median degree it is difficult to predict the presence of the high-degree hubs: the three largest degrees are 79,693 (17 % of the vertices are immediate neighbors of this vertex), 144,896 (with 31 % coverage), and 203,157 (43 %). Fitting a line to the logarithms of the data points gives $\gamma \approx 3.3$ for the middle region of the distribution that obeys a power law.

The difficulty of estimating the degree distribution of a massive graph can be demonstrated by a simple experiment: five times we chose a set of distinct 1,000 vertices in the WordNet graph uniformly at random by sampling simply the vertex labels and calculated the degree distribution for those sets. We also performed *random walks*[4] on the graph, starting at a fixed vertex and moving uniformly at random to a neighbor of the present vertex for a total of 100,000 steps, using five different start vertices and repeating the walk 10,000 times for each start vertex and taking the first 1,000 distinct end points of the walks into the sample set. The words corresponding to the five start vertices (vertex

[4]Random walks as well as sampling will be discussed in detail in Chapter 3.

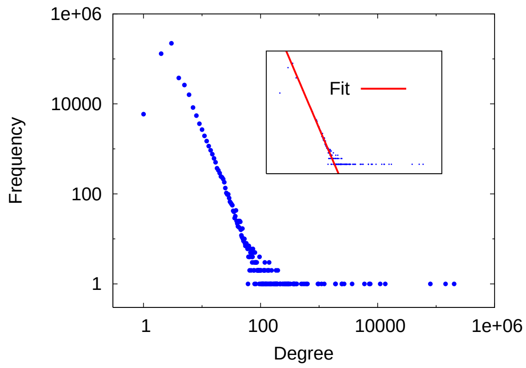


Figure 2.3: The degree distribution of the WordNet graph. The inset shows a line fitted to the "linear part" of the point set using $\deg(v) \in [8, 100]$; the line is $f(x) = -3.35524x + 6.78824$.

degrees shown after each word) are: `green` 2, `wacky` 3, `zip` 5, `heat` 12, and `bear` 16.

The results are plotted in Figure 2.4 with the five uniform sets on the top row and the five random-walk based samples on the bottom row. In each plot, a line with the same slope that the one fitted to the original degree distribution is drawn on each plot, after adjusting the additive constant for each plot in order to overlay the line with the sample data sets. The uniform samples fail to capture the presence of the biggest hubs as the distributions only carry to degrees of a few hundreds and the slope of the sample set is a bit steeper than that of the degree plot of the graph shown in Figure 2.3. The samples obtained by repeating random walks include some of the hubs, although none have picked the largest ones; the slopes are somewhat gentler that that of the graph's degree distribution. The real form of the degree distribution lies in some sense between the two observed distributions, and in addition the presence of large hubs remains unknown if only such samples are available. It can be deduced that a scale-free distribution applies for the graph, but the exponent of the power law can only be crudely approximated from the samples.

Neither of the approaches attempted for sampling the WordNet graph gives a good view on the global structure of the network, simply because the structure of the graph is not uniform or easily predictable: the observations one may make vary depending on which part of the graph is viewed. What we do discover from the plots in Figure 2.4 is that the random walks and the uniformly chosen sets produce different results, which is an informative observation in itself — the details of this are related to the behavior of random-walk based sampling and will be discussed in Chapter 3.

The presence of hubs in a network can be detected by studying the number of vertices reachable from a single vertex v in k "hops", i.e., following paths of length k that start in v. If this quantity, which we denote by $h_v(k)$ and call *hop-span*, grows fast on average over V as k is increased, then it is likely that the network contains high-degree vertices that are reached by a short path from almost any part of the graph and that have several neighbors. A network with a small-world structure shows the same behavior, regardless of whether it has hubs: in just a few hops, practically any vertex can be reached.

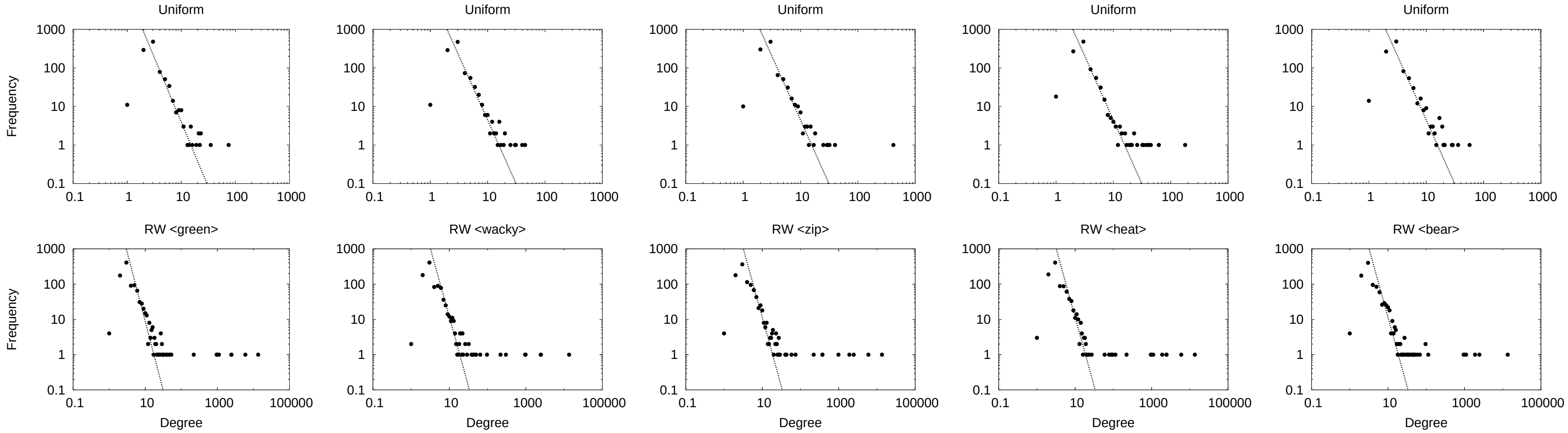


Figure 2.4: The degree distributions over the five sets of vertices selected uniformly at random on the top row, and the degree distributions obtained by the random walks using five different starting vertices on the bottom row. The lines shown in the plot have the same slope than the fit in Figure 2.3, but the additive constant has been adjusted to overlay the line with the dataset.

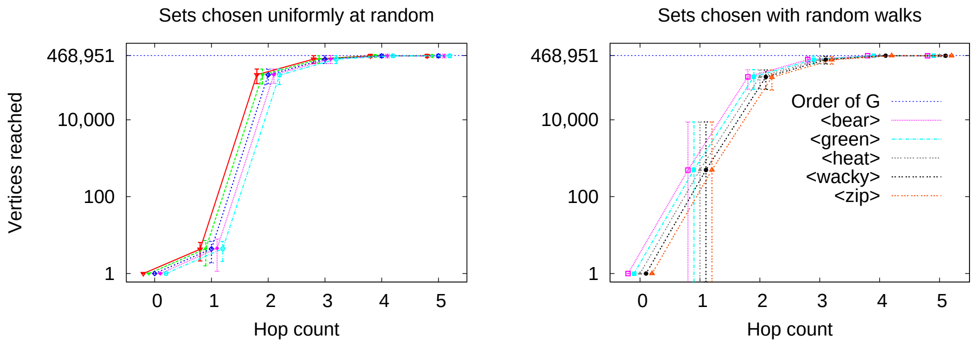


Figure 2.5: The average and standard deviation of the hop-span $h_v(k)$ for small numbers of hops (k). On the left plot, the five data sets are each averages over 1,000 vertices selected uniformly at random. On the right, the averages are over sets of 1,000 vertices that are the end-points of 100,000-step random walks, all started at specific vertices. In each plot, four of the curves were shifted to avoid overlap.

Any network with well-positioned high-degree hubs is necessarily a small-world network.

We calculated $h_v(k)$ for 1,000 randomly chosen vertices and small values of k; the result is shown in Figure 2.5 and indicates that the network is a small world: even though a vertex selected uniformly at random has fewer neighbors than an endpoint of a random walk (an effect which will be discussed in more detail in Chapter 3), all vertices in the ten sets of 1,000 samples have more or less the same number of "second neighbors", i.e., vertices that are two hops away, and furthermore, on average one only needs four hops to reach any given vertex from an arbitrary start vertex. It is also worth noticing how little variability there is in the degree (i.e., the number of one-hop neighbors) of the uniformly selected sets in comparison to that exhibited by the vertices in the random-walk based sets — a random walk is likely to end in the vicinity of a hub.

The raw data sets used to plot Figure 2.5 reveal that the diameter of the network has to be greater than five, as not all vertices are covered in five hops from all start vertices used. We can say that "effectively" the diameter is around four, and assume that it would not be much larger, but we cannot rule out the possibility of there being a "tail" of a dozen vertices forming an otherwise isolated path attached to the graph at one end.

It is not hard to imagine that estimating more complex measures such as clustering or average path-length properties of a massive natural graph is even more complex and uncertain than the observations made above, mainly on degree-based measures. The biggest hopes lie in the study of the behavior of property-discovery methods on small, computationally tractable graphs with certain natural but nonuniform structural properties, and thereby learning to interpret signs of their presence in massive, intractable instances as well.

In this thesis, we will concentrate on two issues, the first being the construction of tools to identify and analyze network nonuniformity, and the second the utilization of such observations in algorithm design.

3 SAMPLING

Natural graphs are often large, not readily available, as well as difficult to process. Hence *sampling methods* that select a subset of the vertices can be helpful when structural properties of a network need to be determined. For example, finding the average path length is computationally demanding for large sets of vertices, but computing shortest paths among a smaller set of vertices can be used to obtain an estimate of the global average. The fundamental question in sampling is how to select the subset to examine so that the estimates obtained reflect the global properties of the graph as faithfully as possible, while keeping the size of the sample needed relatively small. It is often practical to assume that if a sample reflects some commonly used structural properties such as the clustering coefficient, average path length, and degree distribution, then it can also be used to estimate other properties, although counterexamples of samples that preserve certain properties while losing others are frequent. Repeating the sampling and using more than one sampling method can be employed to improve the reliability of the observations made. The need for repetitions however makes it even more important that the sampling procedure will not consume much computation or memory.

If the graph as a whole is available or the number of vertices together with an ordering on the vertex set are known, it is trivial to obtain a uniform random sample by simply taking the list of vertices and picking numbers in $[n]$ uniformly at random to select the sample. Some graphs of interest are massive, rapidly changing, and in many aspects infeasible for obtaining global snapshots or calculating the exact vertex count. Methods to sample such graphs according to a distribution that meets some criteria of interest are useful for many applications. Such methods should preferably operate in a local manner, exploring the graph structure progressively without the need to read large portions of the graph into main memory, or preferably even without needing access to all of the graph at any time, taking advantage of lower or upper bounds or estimates of some global properties to guide the local computation. In this thesis we concentrate on sampling undirected, unweighted graphs, although methods for sampling directed graphs are also of great interest and have various applications.

3.1 MARKOV CHAINS

In order to discuss sampling in mathematical terms, basic knowledge of stochastic processes is necessary, especially regarding discrete-time processes operating in a discrete state space. We begin by briefly summarizing some of the central definitions, but for readers unfamiliar with the fundamentals of the topic, we recommend a comprehensive text book, such as that of Grimmett and Stirzaker [138].

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 $\mathcal{S}$; for a finite state space we denote $|\mathcal{S}| = N$. Formally, a *discrete* *random process* $\mathbf{M}$ is a family

$$\{M_t : t \in \{0, 1, 2, \ldots\}\}, \tag{3.1}$$

where each M_t is a discrete random variable that takes a value from $\mathcal{S} = \{s_1, s_2, \ldots, s_N\}$. A state s_i will frequently be referred to by its index i in some fixed ordering of the state space.

When *transitions* are made from one state to another, the chain *runs*. The probabilities at which a certain state is moved into from the current state characterize a Markov chain. The *Markov condition*, i.e., the dependency on only the current state, is formalized as

$$\begin{aligned}\Pr\left[M_t = s_t \mid M_0 = s_0,\, M_1 = s_1, \ldots, M_{t-1} = s_{t-1}\right] \\ = \Pr\left[M_t = s_t \mid M_{t-1} = s_{t-1}\right],\end{aligned} \tag{3.2}$$

for all $t \geq 1$ and all $s_i \in \mathcal{S}$. A Markov chain is *(time) homogeneous* if the transition probabilities satisfy

$$\Pr\left[M_{t+1} = j \mid M_t = i\right] = \Pr\left[M_1 = j \mid M_0 = i\right], \tag{3.3}$$

for all $t \geq 0$, $i, j \in \mathcal{S}$, meaning that the probability of making a *transition* between a certain pair of states does *not* depend on the time step at which the transition is made. Assuming time-invariance, the *transition probabilities*

$$p_{i,j} = \Pr\left[M_{t+1} = j \mid M_t = i\right], \tag{3.4}$$

for moving from a state $i \in \mathcal{S}$ to state $j \in \mathcal{S}$ form an $N \times N$ *stochastic matrix*[1]. Furthermore, k-step transition matrices, $k \geq 1$, are defined by the corresponding transition probabilities as $p_{i,j}^k = \Pr\left[M_{t+k} = j \mid M_t = i\right]$.

Using the Markov property, and the law of conditional probabilities[2] it is straightforward to derive that the k-step *transition matrix* is in fact $\mathbf{P}^k$, the kth power of $\mathbf{P}$. A state $i \in \mathcal{S}$ is *recurrent* (also called *persistent*), if the chain starts at state i, it will return to that state *almost surely*. States that are not recurrent are called *transient*. A state is *positive recurrent* if the expected number of steps taken to return to that state after leaving it is *finite*. For a Markov chain with a finite state set, at least one state *must* be recurrent and all recurrent states are necessarily positive recurrent. If state i has no transitions to other vertices, it is *absorbing* and $p_{i,i} = 1$.

The mean *first passage time*

$$f_{ij}^t = \Pr\left[M_1 \neq j,\, M_2 \neq j, \ldots, M_{t-1} \neq j,\, M_t = j \mid M_0 = i\right] \tag{3.5}$$

is the time step when the chain first visits state j when started at state i. Thus the probability that the chain *ever* visits state j when started at state i is

$$f_{ij} = \sum_{t=0}^{\infty} f_{ij}^t. \tag{3.6}$$

[1]Take a real-valued matrix $\mathbf{M} = [m]_{i,j}$ for which $m_{i,j} \in [0, 1]$. If the row (column) sums are all one, the matrix is row (column) *stochastic*. If both the row and column sums are one, the matrix is *doubly stochastic*.

[2]For two events A and B, the *conditional probability* of B given A is the probability of B when A is known to hold is $\Pr[B \mid A] = \frac{\Pr[A \wedge B]}{\Pr[A]}$, where $A \wedge B$ is the *joint event* where both A and B hold.

The mean *absorption time* from state j to state i is the mean first passage time f_{ij} in a *modified* chain, where state i is made into an absorbing state by removing all of its outbound transitions, that is, by setting $p_{i,k} = 0$ if $i \neq k$, and $p_{i,i} = 1$. Averaging over all states j yields the mean absorption time to state i regardless of the starting position.

For each recurrent state i, the *period* $d(i)$ of the state is the greatest common divisor of the step counts with which it is possible to return to state i:

$$d(i) = \gcd\{t : p_{i,i}^t > 0\}. \tag{3.7}$$

If $d(i) > 1$, the state is called *periodic*, otherwise *aperiodic*. If any state of a Markov chain is aperiodic, the chain itself is aperiodic; otherwise it is a periodic chain. A state that is *both* positive recurrent and aperiodic is called *ergodic*.

State i *communicates* with state j, denoted $i \rightarrow j$, if $p_{i,j}^t > 0$ for some time step $t \geq 0$. If both $i \rightarrow j$ and $j \rightarrow i$ hold, we write $i \leftrightarrow j$ and say that the states (inter-)communicate. It is easy to see that if $i \leftrightarrow j$, necessarily $d(i) = d(j)$. Also, both states i and j are transient if one of them is, and both are null recurrent if one of them is.

A set of states C is *closed* if $i \in C$ and $j \in \mathcal{S} \setminus C$ implies $p_{i,j} = 0$. A state set C is *irreducible* if all states in C communicate. The communication relation is an equivalence relation over the state space $\mathcal{S}$ that partitions it into irreducible state sets $C_1, C_2, \ldots, C_k$. A chain whose entire state space consists of a *single* irreducible state set is said to be an *irreducible* Markov chain, whereas one with several irreducible state sets is called *reducible*. A positive recurrent, aperiodic, and irreducible Markov chain is called *ergodic*.

In order to examine the distribution over the state space at a given time t, one must specify an *initial distribution*

$$\eta = (\mu_0,\ \mu_1, \ldots, \mu_{N-1}) \tag{3.8}$$

that defines for each state $i \in \mathcal{S}$ the probability μ_i that the chain starts in that state. If the chain always starts as a specified state, the initial distribution simply assigns probability one to that state and zero to others. The distribution over the state space at time t is

$$\mu_t = \eta \mathbf{P}^t. \tag{3.9}$$

An irreducible Markov chain that has only positive recurrent states necessarily has a *stationary distribution*

$$\pi = \big(\pi_0,\ \pi_1, \ldots, \pi_{N-1}\big), \tag{3.10}$$

which is a distribution to which the chain converges in time *regardless* of the initial distribution η:

$$\pi = \lim_{t\to\infty} \mu_t' = \lim_{t\to\infty} \eta \mathbf{P}^t. \tag{3.11}$$

For any stationary distribution it holds for all $j \in \{0, 1, \ldots, N-1\}$ that

$$\pi_j = \sum_{i=0}^{N-1} \pi_i p_{i,j}, \tag{3.12}$$

which means that the distribution no longer changes in time; hence the name *stationary*. Iterating the above, we obtain for all $t \geq 0$ that $\pi \mathbf{P}^t = \pi$. The stationary distribution can also be obtained by computing the left eigenvector corresponding to the largest eigenvalue of $\mathbf{P}$, namely $\lambda_1 = 1$ (the reasons for this are discussed later in this section).

To estimate how close a Markov chain is to its stationary distribution, one may use the *total variation distance* (TVD) between two distributions μ_1 and μ_2,

$$\Delta(\mu_1, \mu_2) = \frac{1}{2} \sum_{s \in \mathcal{S}} |\mu_1(s) - \mu_2(s)|, \tag{3.13}$$

using the current distribution at time t as μ_1 and setting $\mu_2 = \pi$. As an alternative the *relative point-wise distance* [286] $\Delta'_{\mathcal{S}}$ over a subset $\mathcal{S}' \subseteq \mathcal{S}$ can be used:

$$\Delta'_{\mathcal{S}} = \max_{i,j \in \mathcal{S}'} \frac{\left|p^t_{i,j} - \pi_j\right|}{\pi_j}. \tag{3.14}$$

The stationary distribution of an *irreducible* Markov chain is *unique*; it holds that

$$\pi_i = \frac{1}{\nu_i}, \tag{3.15}$$

where ν_i is the mean *recurrence time* of state i, meaning the expected number of steps needed to return to state i after the chain visits i.

For an irreducible Markov chain that has *only* positive recurrent states, there exists a *reversed* chain. Denoting the irreducible positive recurrent Markov chain by $\{M_k : k \in [t]\}$, the reversed chain is given by $\mathbf{M}^{\text{rev}}_k = M_{t-k}$. The chain $\mathbf{M}^{\text{rev}}$ has transition probabilities

$$\Pr\left[\mathbf{M}^{\text{rev}}_{t+1} = j \mid \mathbf{M}^{\text{rev}}_t = i\right] = \frac{\pi_j}{\pi_i} \cdot p_{j,i}, \tag{3.16}$$

in terms of the transition probabilities and the stationary distribution of the original chain $\mathbf{M}$.

The *detailed balance* conditions

$$\forall i,\ j \in \{0, 1, \ldots, N-1\} : \pi_i \cdot p_{i,j} = \pi_j \cdot p_{j,i}. \tag{3.17}$$

hold for the stationary distributions of *reversible* (irreducible positive recurrent) Markov chains. Also irreversible chains can have stationary distributions, although the detailed balance conditions do not hold. Reversibility of a chain $\mathbf{M}$ means that the two chains $\mathbf{M}$ and $\mathbf{M}^{\text{rev}}$ are statistically indistinguishable at equilibrium.

3.1.1 Convergence and mixing time

The time it takes until the probability that a given Markov chain started with an arbitrary initial distribution gets close to the stationary distribution is of special interest; it is called the *mixing time* $\tau_\eta(\epsilon)$ of the chain and is defined as the smallest time step after which $\Delta(\eta, \pi) \leq \epsilon$ holds for all future time steps for a small value $\epsilon > 0$. In rough terms, if the chain reaches stationary distribution in time that is polynomial in the input size (i.e., order of the state space) and also polynomial in $\frac{1}{\epsilon}$, the process is said to be *rapidly mixing*.

For estimating the mixing time of a reversible chain, the method of *canonical paths* [167] may prove useful. The method consists of treating the chain as an *undirected* graph G. First one chooses a set of paths $\mathcal{P}$ in G such that there is a path connecting each pair of states, denoting the length of the path by $\mathcal{L}(i,j)$ for $i,j \in \mathcal{S}$. Then one calculates for $\mathcal{P}$ the following quantity that measures how evenly the "load" of the paths is divided in G:

$$\ell(\mathcal{P}) = \max_{i,j\in\mathcal{S}} \frac{1}{\pi_i p_{i,j}} \sum \pi_i \pi_j \mathcal{L}(i,j). \tag{3.18}$$

If for all i it holds that $p_{i,i} \geq \frac{1}{2}$, then for any initial distribution η concentrated at a single state $s \in \mathcal{S}$ the mixing time has an upper bound [167]

$$\tau_\eta(\epsilon) \leq \ell(\mathcal{P}) \left(\ln \frac{1}{\pi_i} + \ln \frac{1}{\epsilon} \right). \tag{3.19}$$

For a set of states $\mathcal{S}' \subseteq \mathcal{S}$, the *capacity* over the cut $(\mathcal{S}', \mathcal{S} \setminus \mathcal{S}')$ is

$$Q(\mathcal{S}', \mathcal{S} \setminus \mathcal{S}') = \sum_{i\in\mathcal{S}',\, j\in\mathcal{S}\setminus\mathcal{S}'} p_{i,j}. \tag{3.20}$$

The *conductance* of a Markov chain $\mathbf{M}$ is the minimum capacity over all cuts of $\mathcal{S}$ that leave *at least a half* of the probability mass in $\mathcal{S}'$ at the stationary distribution:

$$\Phi(\mathbf{M}) = \min_{\substack{\mathcal{S}'\subseteq\mathcal{S} \\ 0<\sum_{i\in\mathcal{S}'}\pi_i\leq 0.5}} \frac{Q(\mathcal{S}', \mathcal{S} \setminus \mathcal{S}'))}{\sum_{j\in\mathcal{S}'} \pi_j}. \tag{3.21}$$

Mixing time of a Markov chain $\mathbf{M}$ is bounded from above by a function that is proportional to $\Phi^{-2}(\mathbf{M})$ [167, 213].

The *eigenvalue spectrum* of the transition matrix can also be used to evaluate the mixing time of the chain. The eigenvectors form a basis for a vector space. The primary eigenvalue λ_1 of a stochastic matrix is one. The Perron-Frobenius theorem [138] 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. 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 [173]. For more information on mixing times, we recommend Behrends' book [30].

Examining whether a Markov chain has converged to the stationary distribution can be done for example by measuring the total variation distance Δ of Equation 3.13. An experimental evaluation of the total variation distance for a Markov chain can be done by running several instances of the chain from the same start position and calculating an estimate based on the state distribution over the independent instances [49]. Denoting the number of instances ran by I, the number of states by N, and the number of instances that are at state i at time t by $\mathcal{F}_t(i)$, a conservative estimate that slightly overestimates the total variation distance at time step t is

$$\Delta_{\text{est}} = 1 - \sum_{\mathcal{F}_t(i) \neq 0} \min\left\{\frac{\mathcal{F}_t(i)}{I}, \frac{1}{N}\right\}. \tag{3.22}$$

The bias of the estimator can be analyzed for different stationary distributions. For example, take k instances over a chain with N states that has the uniform distribution as the stationary distribution. Assume that at time t_m the instances have mixed and hence the probability to find any single instance j in state $i \in \{0, 1, \ldots, N-1\}$ is $p = \frac{1}{N}$. The number of instances $\mathcal{F}_t(i)$ in a state i at time t is binomially distributed,

$$\mathcal{F}_t(i) \sim \text{Binom}(I, p), \tag{3.23}$$

with $p = \frac{1}{N}$. As the estimate takes the minimum of $\frac{1}{N}$ and the fraction $\frac{\mathcal{F}_t(i)}{I}$, we only need to consider states in which there are *less* than $\lceil\frac{1}{N}\rceil$ instances, as these states are the ones that introduce bias to the estimate. The "deficit" of a vertex with frequency $\mathcal{F}_t(i) < \lceil\frac{1}{N}\rceil$ is

$$\frac{1}{N} - \frac{\mathcal{F}_t(i)}{I}. \tag{3.24}$$

Combining the probabilities that there were exactly $j \in [0, \lfloor\frac{I}{N}\rfloor]$ instances at each of the N states and the corresponding deficits, the total bias is

$$\epsilon_\delta = N \sum_{j=0}^{\lfloor\frac{I}{N}\rfloor} p^j (1-p)^j \binom{I}{j} \left(\frac{1}{N} - \frac{j}{I}\right). \tag{3.25}$$

Figure 3.1 shows the bias estimate of Equation 3.25 for four different values of N, assuming a uniform stationary distribution and a fully mixed chain.

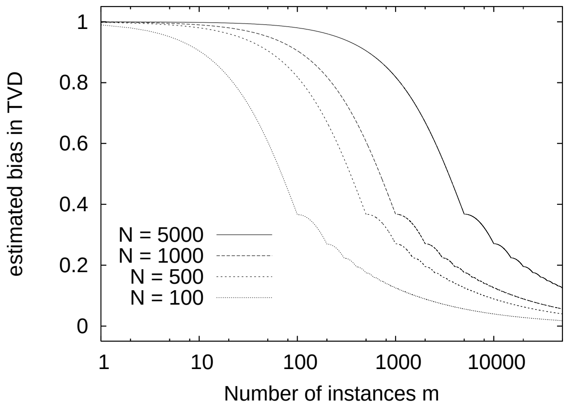


Figure 3.1: The estimated bias of Equation 3.25 to the estimator of total variation distance (Equation 3.22) for four different values of N (i.e., the order of the state space). Note that when the number of instances is a multiple of the state count, the curve displays a kneebend, as the possibility of diving the instances evenly over the state set decreases the total bias.

3.1.2 Random walks

A *(simple) random walk* is a Markov chain in which, given a *neighborhood relation* on the state space, the transition probabilities at each state are uniform over the neighboring states. In essence, a Markov chain can be thought of as a directed, weighted graph $G = (V, E)$ such that $V = \mathcal{S}$ and there exists an edge from vertex i to vertex j if and only if $p_{i,j} > 0$. For a simple random walk, considering such a representation as a directed graph, the transition probabilities are

$$p_{i,j} = \begin{cases} \dfrac{1}{\deg(v_i)}, & v_j \in \Gamma(v_i) \\ 0, & \text{otherwise.} \end{cases} \tag{3.26}$$

Simple random walks on graphs are called *regular* random walks. The *Wiener process* (also called *Brownian motion*) is a simple random walk on a discrete-step line that starts in the origin and takes a transition of one unit to the left with probability p and one step to the right with probability $1-p$. On a plane, a usual definition for the neighborhood structure is a unit grid.

When random walks are used to calculate certain properties or they themselves are the object of study, it is common to initiate several independent random walks from randomly chosen initial vertices, called *origins*. For each object of study, a *saturation time*, i.e., the length of the walk after which accurate measurements can be expected, needs to be determined. The outcomes of the independent walks are then averaged to yield the overall result of the experiment.

3.1.3 Markov Chain Monte Carlo method

The *Markov Chain Monte Carlo* (MCMC) is a popular method with many applications that involve random sampling of the state space. It is based on running a carefully designed Markov chain that has some desired distribution as the stationary distribution for long enough so that it can be assumed to have reached that distribution. The approach originates from statistical physics; for an introduction deeper than given here, we recommend the survey of Jerrum and Sinclair [167]. After the chain is assumed to have converged to the stationary distribution, there are two ways to choose a sample:

(i) one may assume that the reached distribution is close to stationary and take a sequence of visited states to be the sample,

(ii) or if more of an independent identically distributed sample is wanted, only accepting in the sample the state the chain was in at the moment it was considered to be mixed and running the chain (from a fixed initial distribution or continuing from the current state) again for that specified amount of time to get more vertices in the sample.

In order to obtain a valid sample, one must be able to estimate how long it takes for that particular Markov chain to mix, i.e., after how many steps can it be "safely" assumed to have converged to its stationary distribution. Despite the uncertainty, the method is widely used, as it is a powerful polynomial-time method for designing approximation algorithms for hard problems in combinatorial optimization or enumeration of combinatorial objects. Applications of the MCMC method include *approximate counting*, i.e., the estimation of the order of a set of certain combinatorial structures, and combinatorial optimization by defining a distribution on the space of feasible solutions that favors better solutions and then sampling according to this distribution, as well as approximating properties of large systems by sampling the configuration space of such a system (commonly used in statistical mechanics).

The main challenge is usually in obtaining a proper random sample of desired size according to a certain distribution; the random sample is then used to derive the final result in some straightforward manner. For example, if one gets a uniform random sample of webpages, by simple averaging one can derive an estimate for the average number of links per website. It is however nontrivial to get a truly uniform random sample of webpages, for example — we shall return to this topic later in this chapter.

Propp and Wilson [261, 262] sample objects from a finite set, such as nodes from a graph, using an ergodic Markov chain $\mathbf{M}$ (with a transition matrix $\mathbf{P}$) with a carefully designed stationary distribution μ. A major issue here is to determine how many steps should the Markov chain take before a sample can be taken, i.e., how long does it take for the chain to reach the stationary distribution μ from an arbitrary start distribution η (such as a fixed state).

A threshold ϵ can be set on the total variation distance to determine when the chain is sufficiently mixed, requiring that the step count T used fulfills

$$\Delta\left(\eta\mathbf{P}^T, \pi\right) < \epsilon. \tag{3.27}$$

Choosing ϵ is nontrivial. Propp and Wilson find it tedious for an experimenter to analyze the mixing time of the chain, but instead propose using the chain itself to yield an estimate for T.

Instead Propp and Wilson [261, 262] couple k "copies" of the chain $\mathbf{M}$ into a coupled Markov chain $\mathbf{M}^C$ where each state is a k-tuple and the transitions between k-tuples obey the statistics of the original Markov chain. They then run $\mathbf{M}^C$ for a predetermined number of steps T_0, check whether the k-tuple of states *coalesces*, and if not, prepend new steps until it does. The resulting state is the unbiased random sample from the state space.

Fill [119] proposes an algorithm similar to the Propp-Wilson algorithm, but bases it on *rejection sampling*. Rejection sampling, also known as *accept-reject* sampling, works by picking samples from a large population, the instances of which may or may not satisfy a criterion of interest, and then checking the sample against a predefined criterion, and rejecting if it does not comply. Evidently it is difficult to estimate how long one must sample before obtaining the desired number of non-rejected samples.

For reversible chains, the space requirement of a straightforward implementation of Fill's algorithm is much larger than that of the Propp-Wilson algorithm, but it takes fewer transitions. More elegant implementations, however, avoid this problem [119]. The assumptions of the algorithm are the ergodicity of the chain and monotonicity of the reverse transition matrix under a partial ordering that has a unique minimum element and a unique maximum element. Fill [119] argues that the running time of this approach is easier to estimate beforehand than that of the Propp-Wilson algorithm.

Other methods for or related to sampling of the state spaces of Markov chains include the *Metropolis algorithm* (we recommend Chapters 11 and 12 of a book in preparation by Aldous and Fill [12] as an introduction), the *Gibbs sampler* [58] (also known as the *heat-bath* algorithm), and Johnson's [171] coupling-based approaches. We recommend Sinclair's [286] book on the theme. Another comprehensive introduction is provided by Robert and Casella [269].

3.2 SAMPLING NONUNIFORM GRAPHS

For a regular random walk on a graph $G = (V, E)$, the transition probability from vertex v_i to vertex v_j is $p_{i,j} = \deg(v_i)^{-1}$, as observed in Equation 3.26. Such walks are often referred to as "blind" or naïve as picking any transition out of the current state is just as likely. The stationary distribution of such a chain, denoted by $\mathbf{M}^{\mathrm{rw}}$, on a connected graph G is

$$\mu^{\mathrm{rw}} = (\mu_1^{\mathrm{rw}}, \ldots, \mu_n^{\mathrm{rw}}) = \left(\frac{\deg(v_1)}{2m}, \ldots, \frac{\deg(v_n)}{2m}\right), \tag{3.28}$$

as $2m$ is the total number of edge endpoints in G. The distribution of Equation 3.28 can be shown to be a stationary distribution by studying the detailed balance conditions (Equation 3.17 on page 22) in equilibrium, as their validity implies stationarity for a distribution:

$$\begin{aligned} \mu_i^{\mathrm{rw}} \cdot p_{i,j} &= \mu_j^{\mathrm{rw}} \cdot p_{j,i} \\ \frac{\deg(v_i)}{2m} \cdot \frac{1}{\deg(v_i)} &= \frac{\deg(v_j)}{2m} \cdot \frac{1}{\deg(v_j)}. \end{aligned} \tag{3.29}$$

As the equivalence obviously holds, global equilibrium follows from the local equilibrium of the detailed balance conditions and the distribution is stationary.

As the random walk follows any edge outward from the current vertex with equal probability, the stationary distribution "favors" vertices of high degree, and hence any sampling done by a regular random walk will be skewed towards the hubs of a nonuniform network. The measurements made on the degree distribution of the Internet suffered from a similar bias, although instead of sampling single vertices, shortest paths between pairs were sampled [77]. Achlioptas *et al.* [3] show that such path-based sampling can make even a Poissonian degree distribution seem scale-free, as well as a uniform distribution (i.e., a regular graph). An analysis on what causes such a bias is discussed by Dall'Asta *et al.* [92].

The case of vertex sampling is however resolvable. There are several options on how one may enhance the blind random walk to obtain a uniform sample over vertex degrees. A relatively simple method is to apply rejection sampling, accepting a sample with a probability proportional to the inverse of the degree of the sampled vertex [299]. Possible problems include the difficulty of estimating the proportion of acceptable samples in the set of samples obtained and hence uncertainly of the running time of the method for a given number of samples needed. Also there are some mathematical constrains on when such a construction is feasible [65].

Another possibility is to add reflexive edges to each vertex to create a modified graph G' that is $\Delta(G)$-regular [95]. This means that each vertex $v \in V$ of the original graph $G = (V, E)$ is included in G', but with $\Delta(G) - \deg(v)$ (directed) self-loops included in the edge set in addition to the original edges in E. As each vertex in G' has the same degree, they all have equal probability in the distribution of Equation 3.28 for the modified graph G'. Intuitively, such a walk on G' will "stall" on an originally low-degree vertex for a long time, whereas it "passes through" high-degree vertices much quicker.

While simulating the walk, considering all the self-loops separately can be avoided due to the following observation: a self-loop will be chosen with probability

$$p = \frac{\Delta(G) - \deg(v)}{\Delta(G)}, \tag{3.30}$$

which enables us to "flip a coin" to determine whether to stay in the same state or to take an outbound transition. This is essentially a Bernoulli trial with success probability $q = 1 - p$, success interpreted as following a transition that *leaves* the current state. This observation enables us to avoid actually having to construct G' with such a large number of edges, as the expected number of Bernoulli trials before a success is *geometrically distributed* with parameter q. Hence one simply needs to draw a geometrically distributed random number $r \sim \mathrm{Geom}(q)$ to obtain the number of steps that the chain should "stall" at that state, and then continue with the transition probabilities of the regular random walk on the original graph G. We call this construction that uses the geometrically distributed random variable to sample the graph the *coin-flip* random walk.

One can also define a Markov chain that has the uniform distribution as the stationary distribution, instead of running a simple random walk. This is achieved by choosing transition probabilities $p_{i,j}$ to be *inversely proportional* to the degree of the target vertex j and checking that the detailed balance conditions hold. The detailed balance conditions imply that the probabilities should be symmetric with respect to i and j. The reason for the symmetry is that the stationary probability is $\frac{1}{n}$ for both in the uniform stationary distribution. For example, one may use

$$p_{i,j} = \begin{cases} \dfrac{1}{\deg(v_i) \cdot \deg(v_j)}, & \text{if } j \in \Gamma(v_i), \\ 1 - \displaystyle\sum_{v_j \in \Gamma(v_i)} \dfrac{1}{\deg(v_i) \cdot \deg(v_j)}, & \text{if } j = i, \\ 0, & \text{otherwise.} \end{cases} \tag{3.31}$$

For this *degree-balanced* random walk, the detailed balance conditions (Equation 3.17) also hold with the uniform stationary distribution

$$\mu^{\mathrm{id}} = \left(\mu_1^{\mathrm{id}}, \ldots, \mu_n^{\mathrm{id}}\right) = \frac{1}{n}(1, \ldots, 1). \tag{3.32}$$

The equality is easily seen as

$$\begin{aligned} \mu_i^{\text{id}} \cdot p_{i,j} &= \mu_j^{\text{id}} \cdot p_{j,i} \\ \frac{1}{n \cdot \deg(v_i) \cdot \deg(v_j)} &= \frac{1}{n \cdot \deg(v_j) \cdot \deg(v_i)}, \end{aligned} \tag{3.33}$$

which again obviously holds. Also, the self-loop probability $p_{i,i}$ is large. The effect of the self-loop is less severe if we choose the following transition probabilities [43]:

$$q_{i,j} = \begin{cases} \min\left\{\frac{1}{\deg(v_i)}, \frac{1}{\deg(v_j)}\right\}, & \text{if } v_j \in \Gamma(v_i), \\ 1 - \sum_{v_j \in \Gamma(v_i)} \min\left\{\frac{1}{\deg(v_i)}, \frac{1}{\deg(v_j)}\right\}, & \text{if } j = i, \\ 0, & \text{otherwise.} \end{cases} \tag{3.34}$$

Due to symmetry with respect to i and j, detailed balance trivially holds also for this definition with the uniform distribution as the stationary distribution:

$$\frac{1}{n} \min\left\{\frac{1}{\deg(v_i)}, \frac{1}{\deg(v_j)}\right\} = \frac{1}{n} \min\left\{\frac{1}{\deg(v_j)}, \frac{1}{\deg(v_i)}\right\}. \tag{3.35}$$

The self-loop probability $q_{i,i}$ can be rewritten in a simpler form: if $\deg(v_i) \geq \deg(v_j)$, the subtracted term is always $\deg(v_i)^{-1}$. Otherwise, the subtraction is *smaller* by $\deg(v_i)^{-1} - \deg(v_j)^{-1}$. Hence, if we sum over $v_j \in \Gamma(v_i)$, the self-loop probability is the sum of these "leftovers". Using $\deg(v) = |\Gamma(v)|$, we obtain

$$\begin{aligned} q_{i,i} &= 1 - \sum_{v_j \in \Gamma(v_i)} \min\left\{\tfrac{1}{\deg(v_i)}, \tfrac{1}{\deg(v_j)}\right\} \\ &= \sum_{v_j \in \Gamma(v_i)} \left(\tfrac{1}{\deg(v_i)} - \min\left\{\tfrac{1}{\deg(v_i)}, \tfrac{1}{\deg(v_j)}\right\}\right) \\ &= \sum_{v_j \in \Gamma(v_i)} \max\left\{\tfrac{1}{\deg(v_i)} - \tfrac{1}{\deg(v_i)}, \tfrac{1}{\deg(v_i)} - \tfrac{1}{\deg(v_j)}\right\} \\ &= \sum_{v_j \in \Gamma(v_i)} \max\left\{0, \tfrac{1}{\deg(v_i)} - \tfrac{1}{\deg(v_j)}\right\}. \end{aligned} \tag{3.36}$$

We refer to the Markov chain with the above transition probabilities by $\mathbf{M}^{\text{id}}$, and call it the *minimal-balanced chain*, as it aims to minimize the self-loop and balances out the degree-dependency present in the "blind" random walk. The chain defined by the transition probabilities of Equation 3.33 is referred to simply as *balanced*.

Intuitively, a walk that visits the hubs of a nonuniform network can quickly reach any part of the network. Continuing that line of thought, walks like the above two that *avoid* visiting hubs, take a longer time to cover the whole graph. Hence, we expect both of the above chains to mix poorly. Later in this section we discuss the eigenvalue spectrum of the regular random walk transition matrix and the balanced random walk defined by Equation 3.35.

In order to construct a rapidly mixing Markov chain for uniform sampling of $G = (V, E)$, we create a "mirror vertex" v' for each vertex $v \in V$, connect the original vertex and the mirror vertex to each other by transitions, and use different transition probabilities between the original vertices than we do with the mirrors [249]. We call such a chain a *combined random walk* and denote it by $\mathbf{M}^{cc}$. The original vertices $v \in V$ of the input graph G are called the *sampling side* of the modified graph and the mirror vertices $v' \in V'$ form the *mixing side*. The goal of the construction is to ensure that each transition probability can be computed *locally*, only knowing the adjacency list of the vertex corresponding to the current state. This is desirable as for massive graphs, no global information is available, and even making estimates on figures such as the graph order, size, or maximum degree can be hard and/or time-consuming.

We continue to denote by $\deg(v)$ the degree of v *in the original graph* G, i.e., ignoring in the degree the added edge that connects the two sides. Also $\deg(v') = \deg(v)$ in the following discussion. The transition probabilities on the sampling side are set relative to those of either of the above degree-balanced random walks, and on the mixing side, a regular random walk is mimicked with minor modifications. For transitions from a vertex v to its mirror vertex v', we set

$$p_{v,v'} = \epsilon, \tag{3.37}$$

where ϵ is a parameter of the construction. Hence we need to "set aside" probability mass on the sampling side in order to ensure that

$$\epsilon + p_{i,i} + \sum_{j \in \Gamma(i)} p_{i,j} = 1 \tag{3.38}$$

for each vertex i on the sampling side. Both of the degree-balanced walks have self-loops, but unfortunately $p_{v,v}$ may be arbitrarily close to zero, for example in the presence of a large, star-topology[3] induced subgraph. Hence we need to design such variations of the chains that we may be sure that $p_{i,i} \geq \epsilon$ on the walk on the original graph, such that the probability ϵ may be subtracted from the self-loop when constructing the combined chain, without altering the other transition probabilities between vertices on the sampling side.

We achieve this by introducing a guaranteed-weight self-loop for each vertex on the sampling side. For example, if we want to ensure that $\epsilon \geq \frac{1}{2}$, we divide each transition probability out of each vertex by two and add the $\frac{1}{2}$ thus gained (as the initial outgoing "flow" was necessarily one and was halved) to the self-loop probability. For any $\gamma \in \mathbb{Z}$, we may thus ensure that $\epsilon \geq \frac{1}{\gamma}$. For a fixed γ, the transition probabilities $r_{i,j}$ of a given Markov chain are modified as follows to allow $r_{i,i} \geq \gamma^{-1}$ and hence loosening the restrictions on ϵ for all $i \in \mathcal{S}$:

$$r'_{i,j} = \begin{cases} \dfrac{\gamma - 1}{\gamma} \cdot r_{i,j}, & \text{if } v_j \in \Gamma(v_i), \\ \dfrac{\gamma - 1}{\gamma} \cdot r_{i,j} + \dfrac{1}{\gamma}, & \text{if } i = j. \end{cases} \tag{3.39}$$

[3] In a star topology, one vertex is a hub and all other vertices are linked to the hub only.

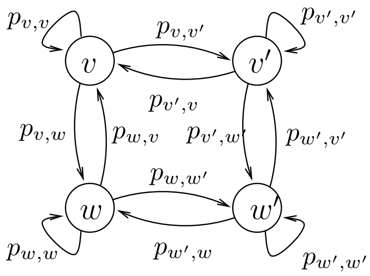


Figure 3.2: A diagram of the mirror construction for two vertices v and w on the sampling side and their mirror vertices v' and w' on the mixing side.

For the two degree-balanced chains this gives

$$p'_{i,j} = \begin{cases} \dfrac{\gamma - 1}{\gamma \cdot \deg(v_i) \cdot \deg(v_j)}, & \text{if } v_j \in \Gamma(v_i), \\ 1 - \displaystyle\sum_{v_j \in \Gamma(v_i)} \dfrac{\gamma - 1}{\gamma \cdot \deg(v_i) \cdot \deg(v_j)}, & \text{if } i = j, \\ 0, & \text{otherwise,} \end{cases} \tag{3.40}$$

(as $\frac{\gamma-1}{\gamma} + \frac{1}{\gamma} = 1$) for the chain defined by Equation 3.31 and respectively

$$q'_{i,j} = \begin{cases} \frac{\gamma-1}{\gamma} \min\left\{\frac{1}{\deg(v_i)}, \frac{1}{\deg(v_j)}\right\}, & \text{if } v_j \in \Gamma(v_i), \\ \frac{1}{\gamma} + \displaystyle\sum_{v_j \in \Gamma(v_i)} \frac{\gamma-1}{\gamma} \max\left\{0, \frac{1}{\deg(v_i)} - \frac{1}{\deg(v_j)}\right\}, & \text{if } i = j, \\ 0, & \text{otherwise,} \end{cases} \tag{3.41}$$

to that of Equation 3.34. Choosing one of these guaranteed-self-loop chains for the sampling side enables us to subtract from the self-loop the transition probability for moving to the mixing side. Similarly, we add a self-loop to each vertex $v' \in V'$ such that

$$p_{v',v'} \geq \epsilon'_v, \tag{3.42}$$

where ϵ'_v is the probability of returning to the sampling side from v',

$$p_{v',v} = \epsilon'_v. \tag{3.43}$$

See Figure 3.2 for an illustration of the connections between the sampling and the mixing sides.

The transition probability from the sampling side to the mixing side is uniform over the vertices on the sampling side, but the return probabilities from the mixing side will not be uniform over the mirror vertices, as will be shown later on. Hence we need to fix a probability

$$\delta \geq \max_{v'} \epsilon'_v \tag{3.44}$$

for the self-loop probability on the mixing side so that we can always safely subtract ϵ'_v from the self-loop of the mixing-side chain. The transition probabilities within the mixing side are therefore

$$p_{i',j'} = (1 - \delta) \frac{1}{\deg(v_i)}. \tag{3.45}$$

The stationary distribution μ^{cc} of the combined chain is a weighted combination of the distributions of the sampling side, μ^{id}, and that of the mixing side, μ^{rw}, such that an α-fraction of the time, the Markov chain is on the sampling side and an $(1-\alpha)$-fraction of the time is spent on the mixing side:

$$\begin{aligned} \mu^{cc} &= (\mu^{cc}_1, \ldots, \mu^{cc}_n, \mu^{cc}_{1'}, \ldots, \mu^{cc}_{n'}) \\ &= \left(\alpha\mu^{id}_1, \ldots, \alpha\mu^{id}_n, (1-\alpha)\mu^{rw}_1, \ldots, (1-\alpha)\mu^{rw}_n\right). \end{aligned} \tag{3.46}$$

We again examine the detailed balance conditions (Equation 3.17) for the above distribution to show that it is a stationary distribution of the combined chain. There are three cases to consider; for a self-loop, the detailed balance conditions trivially hold by definition regardless of the transition probabilities.

(i) Transitions within V on the sampling side: $v \leftrightarrow w$:

As the transition probabilities $p_{i,j}$ in V for $i \neq j$ have not changed other than the introduction of the same multiplicative constant $\frac{\gamma-1}{\gamma}$ for ensuring the self-loop to be able to cover for ϵ, we now multiply both sides of Equation 3.33 or 3.35 (depending on our choice of chain to combine) by the multiplicative constants $\frac{\gamma-1}{\gamma}$ and α, which both cancel out.

(ii) Transitions within V' on the mixing side: $v' \leftrightarrow w'$:

The transition probabilities together with the above distribution fulfill the detailed balance conditions, as we only need to add the multiplicative coefficients $(1-\alpha)$ and $(1-\delta)$ on each side of Equation 3.29, and they cancel out.

(iii) Transitions between V and V': $v \leftrightarrow v'$:

This condition can be met by setting dependencies between the parameters of the construction, the transition probability ϵ, the return probabilities ϵ'_v, and the weighing coefficient α that determines the proportion of time spent on the sampling side, as described below.

The detailed balance condition of the third type is

$$\begin{aligned} \mu^{cc}_i \cdot p_{i,i'} &= \mu^{cc}_{i'} \cdot p_{i',i} \\ \alpha \cdot \frac{1}{n} \cdot \epsilon &= (1-\alpha) \cdot \frac{\deg(v_i)}{2m} \cdot \epsilon'_i. \end{aligned} \tag{3.47}$$

From their definitions we know that $\alpha \in (0,\ 1)$, $\epsilon \in (0,\ 1)$, and for all $v_i \in V$, also $\epsilon'_i \in (0,\ 1)$. Solving the above equation for ϵ'_i gives

$$\epsilon'_i = \frac{2m\alpha\epsilon}{n(1-\alpha)\deg(v_i)} = \frac{2m}{n} \cdot \frac{\alpha}{(1-\alpha)} \cdot \epsilon \cdot \deg(v_i)^{-1}. \tag{3.48}$$

The first coefficient $\frac{2m}{n}$ is the average degree $\bar{k}$ of the original graph G. This is a global property of the graph, but we can eliminate its presence in the parameter equation by further restricting α such that

$$\frac{\alpha}{1-\alpha} \cdot \bar{k} = 1 \Rightarrow \alpha = \frac{1}{\bar{k}+1}. \tag{3.49}$$

As α does not need to be known for constructing the combined chain, but only influences the portion of time the chain spends on the sampling side, its approximate value may be estimated beforehand through estimates of $\bar{k}$. This knowledge is necessary for determining the number of steps that the chain needs to be run until it convergences. Using Equation 3.49, Equation 3.48 becomes simply

$$\epsilon_i' = \frac{\epsilon}{\deg(v_i)}, \tag{3.50}$$

which in turn gives us a safe value for δ, as ϵ_i' is maximized for the minimum degree in G, which in a connected graph is always at least one, giving $\delta \geq \epsilon$. Setting $\delta = \epsilon$ we eliminate the presence of the δ-parameter in the construction of the combined chain.

Implementing the above sampler construction does not require explicitly copying the vertex set, but only a state flag that indicates which set of transition probabilities should be used. Each of the transition probabilities needed at a vertex v, whether on the sampling side or the mixing side, is locally computable from the parameter ϵ, $\deg(v)$, and the degrees of the vertices in $\Gamma(v)$ (which are needed on the sampling side).

The above construction of combining two chains, one rapidly mixing but to an undesired distribution and the other slowly mixing but to a distribution of interest, can be generalized to scenarios other than uniform sampling. The prerequisites are the introduction of a self-loop on both sides such that the side-change transitions can be subtracted without affecting the relative stationary probabilities on each side. Also, to achieve the detailed balance conditions for the crossings from one side to another, one needs to define ϵ_i' fulfill Equation 3.47,

$$\alpha \cdot \mu_A(i) \cdot \epsilon = (1-\alpha) \cdot \mu_B(i) \cdot \epsilon_i' \tag{3.51}$$

where μ_A is the stationary distribution of the sampling side chain alone and μ_B that of the mixing side chain. This property would allow applying the construction to problems in combinatorial generation, enumeration, and counting, where a problem instance needs to be obtained according to some distribution of interest, but all simple-definition chains converging to the distribution in question mix impractically slowly.

We study the convergence behavior of the above chains through a deterministic scale-free graph construction by Dorogovtsev, Goltsev, and Mendes [101], based on [28]. In the DGM generation model, the initial graph $G_{-1} = (V_{-1}, E_{-1})$ consists of two vertices v and w and the edge $\{v, w\}$. At each discrete time step $t \geq 0$ of the process, per each $\{v, w\} \in E_{t-1}$, a new vertex u is added together with edges $\{v, u\}$ and $\{w, u\}$. Thus at time $t = 0$, G is a triangle. See Figure 3.3 for an illustration of the first five generations; Figure 3.4 shows the degree distributions of generations up to 13. At time t,

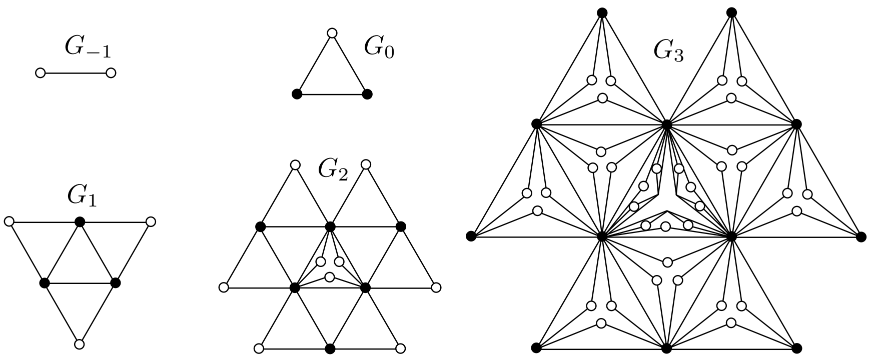


Figure 3.3: The *pseudo-fractal* graph (DGM) G_t for $t \in \{-1, 0, 1, 2, 3\}$ (adapted from [101]). Vertices added at time step t are shown white.

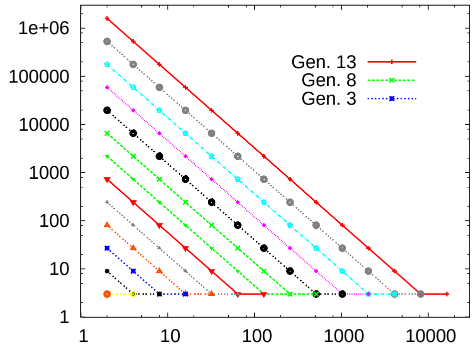


Figure 3.4: Degree distribution of G_t for $t \in [0, 13]$. The degree distributions settle in left-to-right order such that G_{13} is the rightmost plot.

the number of edges is

$$|E_t| = |E_{t-1}| + 2\,|E_{t-1}| = 3\,|E_{t-1}| = 3^{t+1}, \tag{3.52}$$

as $|E_{-1}| = 1$. Similarly, the number of vertices is

$$|V_t| = |V_{t-1}| + |E_{t-1}| = 3(3^t + 1)/2. \tag{3.53}$$

Therefore the average degree of the resulting graph G_t is

$$\bar{k}_t = \frac{2\,|E_t|}{|V_t|} = 4(1 + 3^{-t}). \tag{3.54}$$

The degrees of the vertices are well-behaved: the vector $\mathbf{k}$ of distinct degree values at time $t \geq 0$ is

$$\mathbf{k} = (k_1, k_2, \ldots, k_{t+1}) = (2, 2^2, 2^3, \ldots, 2^t, 2^{t+1}). \tag{3.55}$$

Denoting $\eta_i = |\{v \mid v \in V_t, \deg(v) = k_i\}|$, the vector η of the number of vertices with degree k_i is

$$\vartheta = (\vartheta_1, \vartheta_2, \ldots, \vartheta_{t+1}) = (3^t, 3^{t-1}, 3^{t-2}, \ldots, 3^2, 3, 3). \tag{3.56}$$

For these graphs, the transition probabilities t_i that go out from a vertex that has been alive for i generations are easy to compute. The degrees are

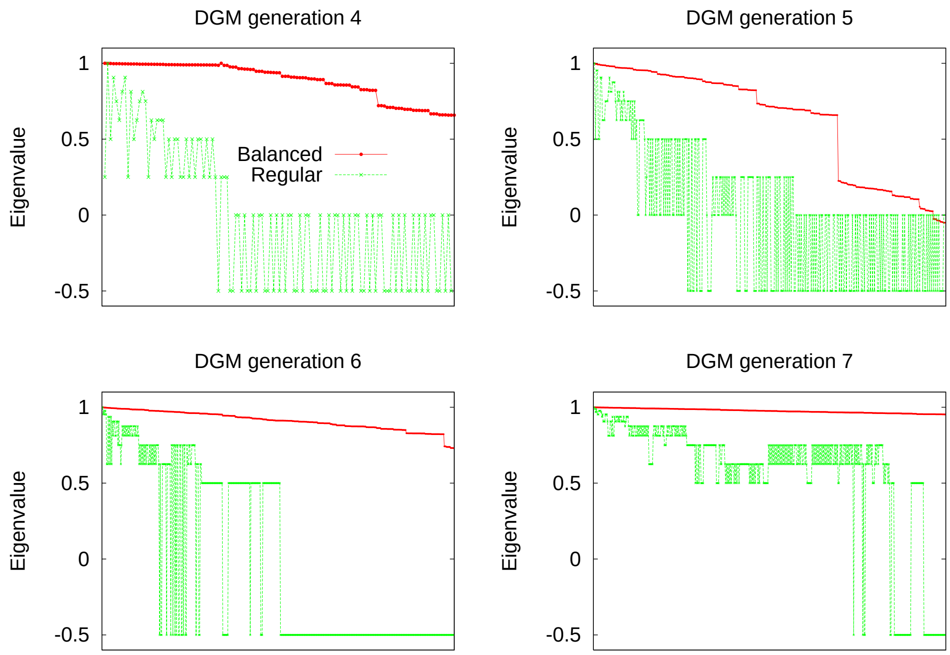


Figure 3.5: The spectra of four generations of the DGM model for two different transition matrices: one based on the regular random walk transition probabilities of Equation 3.26 on page 25 (referred to as "Regular") and another one based on the balanced random walk transition probabilities of Equation 3.35 on page 29 (referred to as "Balanced").

known from Equation 3.55, giving $t_i = 2^{-(i+1)}$. Thus for the newly created vertices, the outward transition probability is $t_0 = \frac{1}{2}$. Similarly for vertices that were created on the preceding step we have $t_1 = \frac{1}{4}$, and so forth.

We computed the eigenvalue spectra of the regular random walk transition matrix and the balanced random walk defined by Equation 3.35 for a few generations of the DGM model; the plots are shown in Figure 3.5. It is evident from the plots that the eigenvalues of the balanced chain are larger than those of the regular walk, which supports the previously discussed intuition of slow mixing, partially caused by the self-loop probabilities.

We studied the behavior of the method on the DGM model [101] and *collaboration graphs* (see Section 4.4.1 on page 63). From Figure 3.6 it can be seen that the regular random walk samples vertices preferentially to their degree, whereas the method that weights vertices inversely to their degree maintains an indifference to vertex degree, and hence will obtain a sample with a degree distribution similar to that of the graph from which the sample is taken. The tendency of the degree-balanced method to unwanted locality is evident in plots of the *coverage*, i.e., the percentage of the graph covered at each step of the walk, shown in Figure 3.7.

We studied the convergence of the sampling methods to their respective stationary distributions over the vertex set by estimating the total variation distance between the obtained and the stationary distribution with Equation 3.22. For the combination walk, only those instances that were currently in a sampling state (instead of being on the mixing side of the construction) were included in the estimate, and hence the estimates for the other two

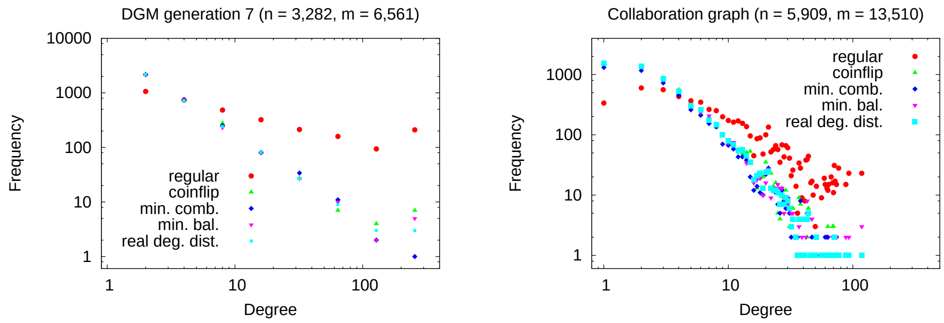


Figure 3.6: log log-plots of the sampling frequency of vertices by their degree in a set of n independent samples (fully restarted from the same randomly chosen vertex, allowing the walks to mix for $t \geq 5{,}000$) shown together with the degree distributions of the graphs themselves. The regular walk is the $\mathbf{M}^{\mathrm{rw}}$ chain, min. bal. is $\mathbf{M}^{\mathrm{id}}$, and min. comb. is $\mathbf{M}^{\mathrm{cc}}$; the coin-flip walk was discussed on page 28. We used $\epsilon = 0.25$ for the combination walk.

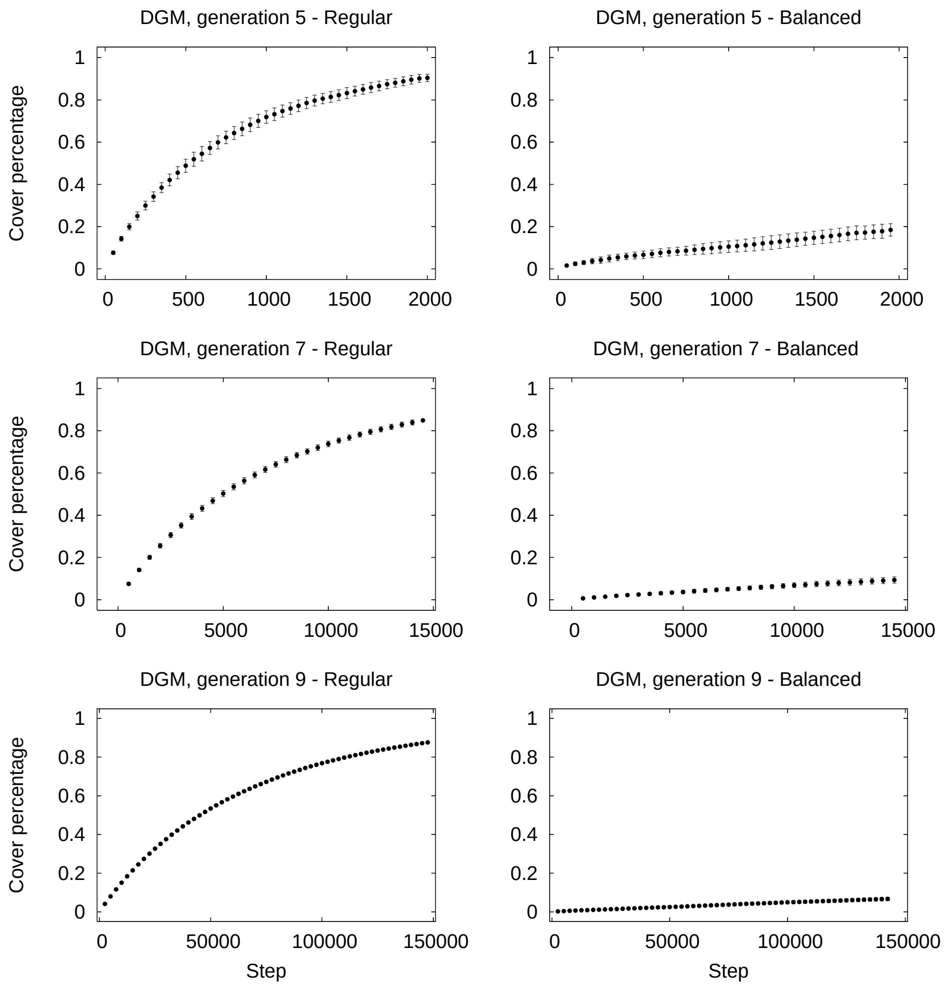


Figure 3.7: The coverage achieved by the regular (left) and the degree-balanced (right) walks at each step. In all six plots, averages and standard deviations of 50 independent walks are shown.

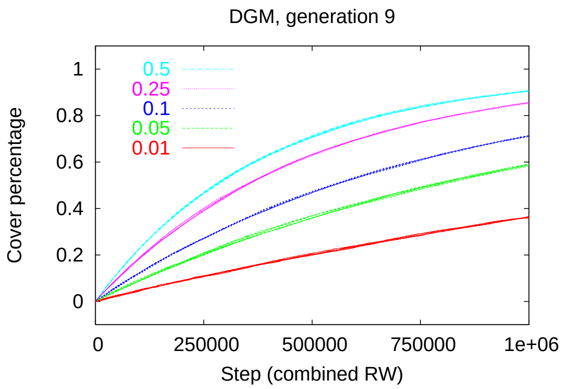


Figure 3.8: The behavior of the combined random walk on a ninth generation graph for different values of ϵ. Increase in coverage is only measured at sampling steps, i.e., the vertex visits of the mixing side of the construction are not counted in the coverage but are present in the step count.

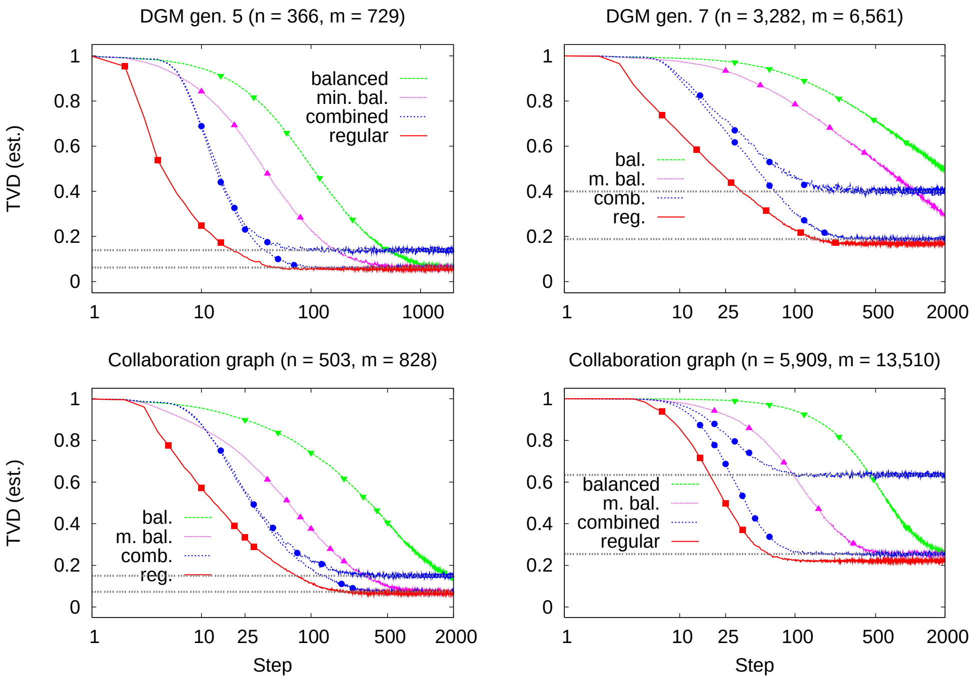


Figure 3.9: Values of $\Delta_{\mathrm{est}}(t)$ for the balanced (▼), minimal-balanced (▲), combined (●), and regular (■) chains. The estimate is calculated over a set of $I = 15{,}000$ independent walks in two DGM and two collaboration graphs, all starting from a fixed vertex, initially chosen at random. The bias of the uniform distribution estimate (Equation 3.25) for each graph is shown as the lower dotted line. Note that for the combination walk (based on the minimal-balanced RW, $k = 4$, $\epsilon = 0.25$), the expected number of instances on the sampling side of the combined walk is $\alpha I \leq 15{,}000$ and hence the bias is larger (the upper line). For ease of comparison, we also ran the combination walk for $I' = \alpha^{-1} I$ to achieve the same expected bias with the balanced walks.

walks are based on a greater number of independent instances than those of the combination walk. Using such estimation, the stationary distribution

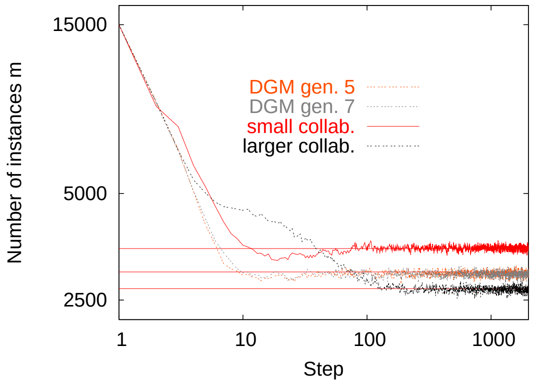


Figure 3.10: The number of walk instances out of the total of $I = 15{,}000$ instances that are on the sampling side of the combined walk at each step; the data used is the same than in Figure 3.9. We use $\epsilon = 0.4$; this gives $\alpha \approx 0.20$ for DGM generations 5 and 7, $\alpha \approx 0.23$ for the smaller collaboration graph and $\alpha \approx 0.18$ for the larger. The theoretical value to which the plots are expected converge, $I \cdot \alpha$, is shown in the plot as a vertical line for each value of α.

to which the combination walk should converge is the uniform distribution. Figure 3.9 shows the estimate for the DGM construction, generations five and seven, and for two collaboration graphs based on natural data. As the bias of the estimate depends on the number of instances for which it is calculated, we have plotted the actual number of instances on the sampling side of the combination walk for the data of Figure 3.9 in Figure 3.10.

3.3 APPLICATION AREAS

Random sampling is a powerful tool in the construction of efficient algorithms for demanding computational problems [74, 75, 131, 177, 178]. Sampling methods are useful for example in lossy data compression [24]. They also helps to analyze and understand properties of large combinatorial objects. Lately sampling of large natural networks has been actively studied in order to better understand the function of such systems.

The Internet has been an object of intensive study ever since its economical value was recognized in the 1990s. Paxson and Floyd [256] found in 1997 the topology of the Internet difficult to characterize due to the constant change and growth that are essential qualities of the network. They motivate the research on Internet simulation by the possibility to approach "complicated scenarios that would be either difficult or impossible to analyze" [256]. Just a few years later the size of the network is already significantly bigger and the problems related to its topology are more urgent than ever; more efficient protocols are needed for the ever-growing amount of traffic. Krishnamurthy *et al.* [196] discuss sampling of the Internet, with emphasis on how small of a sample can still be useful and informative. It would be helpful to be able to predict the future evolution of the network in order to design better hardware, traffic protocols, and routing algorithms [112, 308]. For a broader discussion of modeling the Internet as well as the World-Wide Web, we recommend a recent book by Baldi *et al.* [21].

Another widely studied network is the World-Wide Web (WWW), with either individual pages or entire websites as vertices, and links as edges. The interest in mathematical models is justified by the size of the resulting network; in 1999 even the best search engines could cover only about one third of the estimated size of the WWW [203]. Since then, the network has grown significantly and full indexing is impossible. The *Web Graph* [48] is a graph representation of the WWW, with webpages as nodes and hyperlinks as directed edges. A compact review of some of the numerous generation models proposed is given in [306].

It is a problem of great interest to construct a method to sample webpages uniformly [148]. In addition to estimating the properties of the Web Graph in general, for example the coverages, overlaps, and relative sizes of different search engines could be estimated fairly reliably based on a large uniform random sample of webpages, as suggested by Bharat and Broder [32]. Just running a random walk for a while will not trivially adapt to the Web Graph, as it is directed and has been found to have a scale-free degree distribution. As a surrogate for the "real" uniform sample, Bharat and Broder discuss as an alternative sampling from the search engines themselves, still being able to estimate the relative sizes and overlaps of the engines. This approach suffers from search engine coverage being neither independent nor random, as they point out. Any method that is based on sampling from a search engine will also require access to the database of the engine beyond the regular search interface.

Essentially the distribution over the webpages produced by random walks in search engine databases as done by Bharat and Broder [32] is close to that of the *PageRank* weights of the pages. The computation of a PageRank value is such that the value of the weight is proportional to the probability that a "random web-surfer" hits the page and hence proportional to the number of times a random surfer hits the page [47].

Henzinger *et al.* [149] strive to modify this approach so that the sampling distribution is near-uniform. They use a multi-threaded web crawler to perform the random walks; upon entering a page, their crawler with probability $1-p$ selects one of the links uniformly at random, and with probability p, performs a "jump" to a random webpage that is selected from all pages crawled so far by any of the threads. Mathematically, the probability that a certain webpage w is sampled into the sample set S is conditional on whether it is in the set of crawled pages $\mathcal{C}$,

$$\Pr[w \in S] = \Pr[w \in \mathcal{C}] \cdot \Pr[w \in S \mid w \in \mathcal{C}] = \Pr[w \in S \cap \mathcal{C}]. \tag{3.57}$$

In order to argue on the distribution of $\Pr[w \in S]$, one must be able to estimate $\Pr[w \in \mathcal{C}]$. Denoting the PageRank value of w by $r(w)$, the number of times w gets visited in a random walk of ℓ steps is approximately $\ell \cdot r(w)$, as $r(w)$ is by definition comparable to the probability of hitting w on a random web-surf. In practice, large values of ℓ are infeasible. Supposing that ℓ is relatively small, Henzinger *et al.* argue that each page should be visited at most once and hence the expected number of visits per page should be in $[0, 1]$. Hence they approximate

$$\Pr[w \in \mathcal{C}] \approx \mathrm{E}[\text{visits to } w] \approx \ell \cdot r(w). \tag{3.58}$$

Therefore, sampling pages in such a manner that

$$\Pr\left[w \in S \mid w \in \mathcal{C}\right] \propto r\left(w\right)^{-1} \tag{3.59}$$

should yield a near-uniform distribution. If the PageRanks of the pages are not known, they need to be estimated along the crawl as well, which poses another challenge, discussed in [149] but omitted here. Henzinger *et al.* [149] study the uniformity of the distribution obtained by the above method experimentally by generating random scale-free graphs. Theoretical studies of the behavior of crawlers for different web graph models also exists [82].

A walk-combination approach similar to ours has been taken by Wei, Erenrich and Selman [316] in sampling satisfiable evaluations for a SAT instance. They use a random walk mixed with the Walk-SAT [282] algorithm[4] to obtain a near-uniform sample of satisfiable truth assignments, whereas using Walk-SAT alone would result in a nonuniform one. We expect our combination chain to provide a starting point for similar constructions.

As future work, it would be useful to study how well different kinds of sampling methods preserve different structural properties of large (natural or generated) networks. Chakrabarti *et al.* [59] propose the *NetMine* tool for analyzing large graphs by providing views on different structural properties of graph instances given as input. Among other features, the tool aims to identify vertices or subgraphs that are structural "outliers", i.e., pointing out nonuniformities in the network structure. The scalability issues in the implementation of such tools could be avoided if efficient sampling mechanisms were first employed to obtain a "preview" of the plots that are assumed to be of interest and then selecting the measures to be calculated for the full data set based on observations made on the preview plots.

[4]Walk-SAT is a popular satisfiability solver introduced in mid-1990s, i.e., an algorithm to find satisfying truth assignments to the variables of a logical formula.

4 CLUSTERING

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*. 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 [106], in which the task is to recognize some form of regularity from a given data set, usually by diving the data into a set that forms a pattern and another set that is considered noise or background. Such systems are usually trained by providing pre-processed samples to the system that show the correct separation into pattern and noise. If the system is given feedback on the classifications it makes in the training phase, the learning process is *supervised*, and if the system is only given the data and no interaction takes place, the learning is *unsupervised*. In cluster analysis, commonly no *a priori* information on the clusters is available, meaning that there a training set of data elements with predetermined clusters is not usually available. In the presence of such training set, a clustering method may adapt to a specific application and learn to classify unlabeled data elements to appropriate initial clusters that are based on the training data.

Clustering has become an important tool in *data mining*, in which large data sets with various attributes per data element are searched for regularities. The types of data vary. One application is clustering of *text documents*, where the working hypothesis is that documents that are relevant with respect to each other form clusters under properly defined semantic similarity measures [61]. If the data set consists of speech or handwritten characters, a common task is to identify which sound in the speech correspond to the same phoneme or which characters of the writing correspond to the same letter. In such applications, the number of classes in which to classify the data elements is known *a priori*, but in general this is not the case. Hence methods that do not expect a fixed, predefined number of clusters are motivated.

Formally, given a data set $\mathcal{D}$, the goal of clustering is to divide it into clusters $\mathcal{C}_1, \ldots, \mathcal{C}_k$,

$$\bigcup_{i=1}^{k} \mathcal{C}_i = \mathcal{D} \tag{4.1}$$

such that the elements assigned to a particular cluster $\mathcal{C}_i$ are similar or connected in some predefined sense. It depends on the application area whether the clusters $\mathcal{C}_1, \ldots, \mathcal{C}_k$ should form a *partition* such that

$$\mathcal{C}_i \cap \mathcal{C}_j = \emptyset \text{ if } i \neq j, \tag{4.2}$$

or alternatively a *cover* of the data set $\mathcal{D}$ — the latter option allows for one data element $d \in \mathcal{D}$ to belong to *more* than one cluster, but each d needs to be assigned to *at least one* cluster.

Computing a clustering does not always produce a single partition or cover, but rather a hierarchical cluster structure, where each top-level cluster is composed of subclusters and so forth. Methods that produce such clusterings are called *hierarchical*, opposed to the *flat* clusterings only comprising of 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.

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

(1) *top-down or* divisive *algorithms that split the dataset iteratively or recursively into smaller and smaller clusters, and*

(2) *bottom-up or* agglomerative *algorithms that start with each data element in its own* singleton cluster*, iteratively merging the clusters into larger ones.*

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. Possibilities for such quality measures are discussed in Section 4.3. A partitional hierarchical clustering is best represented by a *dendrogram*, which is a tree that shows the clusterings at different levels; an example is shown in Figure 4.1 — dendrograms can also be drawn with superclusters on the bottom or horizontally.

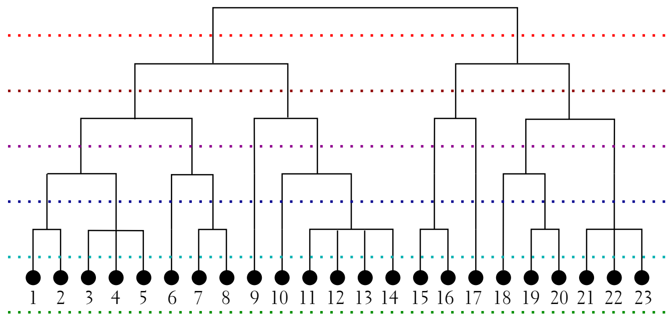


Figure 4.1: An example dendrogram that groups 23 elements into clusters at four intermediate levels, the root cluster containing the entire dataset and the leaf clusters each containing one data point. Any level of the dendrogram, indicated by dotted lines in the picture, can be interpreted as a clustering, grouping together as a cluster those elements that remain in the same branch of the dendrogram tree above the line.

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. The former approach does 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 an 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, always be able to provide some solution, at least a crude approximation, and incrementally incorporate newly added data into the clusterings [44]. 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 [61]. These have been suggested for web page classification in [319]. It is noteworthy that in online clustering, the order in which the data are processed can significantly affect the resulting clusters. Especially immutable cluster assignments suffer from such sensitivity. Hence it is common that the existing partial clustering is constantly optimized with respect to some carefully selected global measure as new data are processed.

4.1 GRAPH CLUSTERING

Our emphasis is on graph clustering. We will discuss different possibilities of constructing and evaluating clusterings on graphs, defining similarities and fitness functions, and ultimately using these define local methods for graph clustering. Our methods rely on density measures and eigenvectors.

For many types of datasets, fluent representations as graphs exists. For example, 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 neighbors*[1] [161]. Also different range-based and nearest-neighbor methods are basic building blocks for transformation algorithms that construct graphs from set of n-dimensional vectors.

Usually the data elements $d \in \mathcal{D}$ are represented by the vertices, and an edge is placed between two elements depending on their *similarity* (large values for similar data elements) or *distance* (smaller values for similar elements) under some measure, selected according to the application. Any data for which a similarity measure has been defined can be trivially transformed into a 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, weighted complete graph can be formed by representing each datum d_i by $v_i \in V$ and using

$$\omega(\{v_i, v_j\}) = \rho(d_i, d_j) . \tag{4.3}$$

[1] Two points are Voronoi neighbors if their *Voronoi cells* are adjacent [18]. 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.

For asymmetrical similarities, the resulting graph is directed and hence the number of edges is doubled. A problem is that such a number of edges is quadratic in the size of the data set, and for large data sets, this is computationally infeasible. The number of edges in the graph can be controlled by setting a threshold value ξ:

$$\{v_i, v_j\} \in E \text{ if and only if } \rho\,(d_i, d_j) \geq \xi, \tag{4.4}$$

although choosing the value of the threshold is application-specific and not always easily justified. Also more complex methods than using a mere threshold are available for making a connectivity graph sparser. One possible approach is to place 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 [23].

In the field of applied graph theory, clusters are also called *communities*, and identifying the clusters in a graph is known as determining the *community structure* of the graph [239, 240]. In an ideal graph cluster, all cluster members share edges with each other and have few if any connections with the rest of the graph [175, 191, 239]; the edges that connect vertices within a cluster are called *intra-cluster* edges and those crossing cluster boundaries are called *inter-cluster* edges.

Should the edges have weights, cutting an important edge when separating a cluster is to be punished more heavily than cutting a few unimportant ones. 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. For a distance measure, the cluster boundary should be located in an area where including more of the outside vertices would drastically increase the intra-cluster distances (for example, the sum of squares of all-pairs distances).

The naïve definition of a cluster as a subgraph with maximum density fails in many ways: trying to grow the subgraph containing v with maximum density, one will always reach the maximum value of 1 after the addition of any single neighbor of v. Attempting to find the largest subgraph with density one is also infeasible for large graphs, as deciding whether a graph contains a clique of a given order is **NP**-complete [130] and optimization to find the maximum clique is **NP**-hard, and hence the computation needed would be exponential in n. For many optimization problems, good fitness measures are **NP**-complete, and as density is a natural clustering measure, it makes a useful component in defining a well-behaving fitness measure for clustering.

Clusters in graphs can also be defined through *connectivity*, 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, whereas there should not be many paths connecting them to vertices outside the cluster [145]. For a better understanding and intuition on what properties good clusters exhibit, we will first review methods for *global clustering*. Such methods output a clustering for the entire vertex set of the input graph. Later we will discuss methods where the output is restricted to the clusters of specific "seed vertices".

In order to cluster an unweighted graph, Newman and Girvan [241] impose weights on the edges. The weight they use is the *(edge-) betweenness* of the edge, which is the number of shortest paths connecting any pair of vertices that pass through the edge. If there are k shortest paths connecting v and w, each of them will have weight $\frac{1}{k}$ in calculating the betweenness measures of the included edges. The current algorithms to compute betweenness for an edge operate in $\mathcal{O}(nm)$ time; also a method based on random walks rather than exact computation has been proposed by Newman [235].

Newman and Girvan assume edges with high betweenness to be links between communities instead of internal links within a community: the several shortest paths passing through these edges are the shortest paths connecting the members of one community 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 an algorithm polynomial in n and m for clustering. Of course one still must decide when to stop the partitioning, just as for the hierarchical clustering method. Newman [239] proposes computing a quality measure called *modularity* over the entire clustering (namely Equation 4.38 discussed later in Section 4.3) and stopping the clustering when there is no improvement.

Clauset *et al.* [78] present another hierarchical, agglomerative method that has complexity $\mathcal{O}(mk \log n)$, where k is the depth of the generated dendrogram. They state that for sparse, natural graphs the method runs in $\mathcal{O}(n \log^2 n)$ time in practice, presenting results for a graph with over 400,000 nodes and some two million edges. The method performs greedy optimization of modularity similarly to the method of Newman [239].

Spectral clustering [175] is based on computing eigenvectors (usually that corresponding to the second-largest eigenvalue) of a matrix representing the graph structure, such as a modified version of the adjacency matrix or the transition matrix of a regular random walk on the graph. The component values of the resulting eigenvectors are used in a sense as similarity values among the vertex set to determine the clustering, usually by identifying a cut and performing divisive hierarchical clustering [259, 288]. Spectral methods are in general computationally demanding, although a distributed algorithm for decentralizing the computational load has been proposed [183]. For example clustering and other analysis of the network of the Internet autonomous-system domains has been done with spectral methods [135, 223]. Analysis on why spectral methods work well for clustering is currently somewhat scarce, but the field is becoming better understood [139, 175].

Goh, Kahng and Kim [137] have studied the spectrum created by the Barabási-Albert generation method for scale-free graphs with two outgoing edges per added vertex. They are able to find the exact spectrum for graphs up to 5,000 vertices and determine the first few of the largest eigenvalues for graphs of order as high as 400,000. Hence we do not expect spectral methods to scale up to massive graphs. Also theoretical results on the spectra of graphs with defined degree distributions is available [71]. Another alternative is to use *maximum-flow* [87] calculations between vertex pairs as a starting point for clustering [45, 121, 302].

Clustering methods can also be based on or evaluated by computing the conductance of a graph (or namely of the corresponding Markov chain for a regular random walk, defined in Equation 3.21 on page 23) [45, 175]. A variation of conductance is known as *normalized cut* [147]. Conductance corresponds to an **NP**-complete problem [284], as do practically all good measures for clustering computations. The normal procedure in a divisive, hierarchical, conductance-based method is to remove the cut that gives the graph's conductance; this splits the graph into two clusters which are then further iterated upon to split them into smaller clusters.

We denote a cluster in a graph $G = (V, E)$ by $\mathcal{C}$, where $\mathcal{C} \subseteq V$ is the vertex set. We usually think of the internal structure of the cluster as the subgraph induced by the vertex set $\mathcal{C}$, and use the cut $(\mathcal{C}, V \setminus \mathcal{C})$ to evaluate the connectivity of the cluster to the rest of the graph, often directly using the cut size $c\,(\mathcal{C}, V \setminus \mathcal{C})$.

4.2 LOCAL CLUSTERING

Traditionally, clustering algorithms have been designed to obtain a *global* clustering for a given graph, i.e., to assign each vertex to a cluster. For large graphs, this becomes computationally difficult, as the running time of an algorithm should not grow faster than linearly in the order of the graph n in order for it to be at all scalable; sublinearity is strongly preferable. For example, the World-Wide Web has billions of vertices and many more edges, setting it out of reach of the global algorithms. The existing global approaches a capable of dealing with up to a few millions of vertices on sparse graphs [156, 239, 240]. Another issue is that many large networks 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 WWW for search-engine construction [67, 298].

For many applications, only a small subset of vertices needs to be clustered instead of the whole graph. These include locating documents or genes closely related to a given "seed" data set. This motivates the use of a *local* approach 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. The scalability problem is avoided, as the graph as a whole does not need to be processed unless a single cluster is composed of nearly the entire graph. Also, clusters for different seeds may be obtained by parallel computation.

4.2.1 Clustering by local search

Local search methods are probabilistic algorithms designed to find near-optimal solutions in large, complex state spaces. Instead of performing a complete, exhaustive search of all possible states, they "navigate" the search space by moving from one state to a "neighboring" state using a heuristic, usually involving transition probabilities, for choosing the next state to visit. At each visited state, a *fitness function* is evaluated to determine whether that state provides a good solution. Out of all the visited states, the one yielding the best fitness value is returned as the solution of the local search.

The decisions to be made in formulating a local search method are defining the neighborhood relation over the space of possible solutions, aiming to define one that can be navigated with light computation, the choice of the fitness function to choose good solutions with tolerable computational cost, and the guidance of the search in the neighborhood. If the neighborhood definition produces local optima, the search must be guided in such a manner that it is possible to "escape" a local optimum — a simple *greedy search* always proceeding to the neighbor with the highest fitness value will always return the first local optimum it reaches. When running a local search method, usually several iterations with either fixed or somehow randomized start states are performed, each iteration composing of a predetermined maximum number of steps to be taken. For a good review on local search methods, we recommend the book by Aarts and Lenstra [1].

To locate a cluster containing a given vertex $v \in V$ from a graph $G = (V, E)$, we stochastically examine subsets of V containing v, and choose the candidate with maximal $\mathcal{F}(\mathcal{C})$ to be $\mathcal{C}(v)$; note that the same procedure is applicable to any fitness function that assigns higher values to better clusters. The extent to which the graph is traversed depends on the local search method applied.

We define an initial cluster $\mathcal{C}'(v)$ of a vertex v such that it contains v itself and its neighborhood $\Gamma(v)$. We then allow each search step to either add a new vertex that is adjacent to an already included vertex, or remove an included vertex. Upon the removal of $u \in \mathcal{C}'(v)$, $u \neq v$, the connected component containing v becomes the next cluster candidate. Note that also other definitions of an initial cluster can be used, such as starting from a singleton cluster with the seed vertex v only, and also the neighborhood definition can be modified to fit an application area.

One option for guiding a local search procedure (in a manner that aims to avoid getting stuck on local optima) is the *simulated annealing* method [186] that allows the search to proceed to a lower-fitness neighbor 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 [193]. An application of simulated annealing to optimizing a cluster fitness function is given in Section 4.2.4, with pseudo code in Table 4.1.

With the proposed method, also clusterings for partially unknown graphs are fluently obtained. The method's strongest asset is the possibility for online computation of clusters that are with high probability locally optimal with respect to the fitness measure. It is also noteworthy that the method requires no parameters, whereas typically parameters such as a density threshold or the number of clusters need to be user-defined for clustering algorithms.

4.2.2 Similarity-based clustering

For general data, it is not always immediately clear what would be the proper similarity measure. The case is equally confusing on graphs. We begin be discussing similarity measures and similarity-based clustering for point sets to shed light on the foundations on which similarity-based methods for graphs are commonly built.

If the data can be fluently mapped into vectors, for example in $\mathbb{R}^n$, a suitable distance measure such as the *Euclidean distance* of two n-dimensional vectors $\mathbf{x}$ and $\mathbf{y}$

$$\text{dist}_{\text{Eucl}}(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|^2 = \sqrt{\sum_{k=1}^{n} |x_k - y_k|}. \tag{4.5}$$

can be used as the clustering similarity measure. It works well for datasets that are spread evenly in the range of X in all dimensions and are not sensitive to rotations or translations. For strings, for example, there are several alternative definitions of similarity than can be employed, such as variations of the *edit distance* [142]. The choice will eventually depend on the application. Using a distance as a similarity measure, a good clustering is such where the distance of elements that belong to the same cluster is small in comparison to the inter-cluster distance.

A widely-spread method for clustering points in space is the *k-means clustering* [106, 144]. The method clusters a given set of vectors $X \subset \mathbb{R}^n$ into k classes by iteratively constructing a set of k *mean vectors* $\mu_1, \ldots, \mu_k$ using the Euclidean distance to optimize the positions of the mean vectors with respect to the total distance to the elements of X. After computing the mean vectors, the dataset X is reclassified accordingly: a vector $\mathbf{x}_j \in X$ belongs to class i if and only if μ_i is the nearest mean vector to $\mathbf{x}_j$ with respect to the Euclidean instance.

If the exact number of clusters, which is the value of the parameter k, is not known beforehand, it may be estimated by running the algorithm for several different values of k, but this may be costly for large datasets. Another option is minimizing the *Davies-Bouldin index* [93] to choose the best value of k [265, 180]. The k-means method is also sensitive to the initial cluster assignments. There are also a number of approximation algorithms for the k-means problem [6], some of which operate online [140, 247].

For clustering large datasets of points in space, Nagesh *et al.* [232] suggest a parallel clustering algorithm called pMafia that clusters the data in a bottom-up fashion, attempting locally to identify clusters in subspaces of the data instead of handling all of the data at once. Also Zhang *et al.* [328] propose a local method, based on a "closeness" value of data elements, computable during the clustering. The clusters of Birch are dense regions of the data space, and isolated points may be optionally altogether ignored. Both of these methods are reported to be efficient and to produce high-quality clusters. They are however somewhat complex to implement.

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 (i.e., solve the Kirchoff equations) for all of the vertices, and cluster the vertices based on the potential differences [240, 241]. The problem is the placement of a battery into the system; how to choose the source and the sink of the current? The idea of computing voltages can however be fluently applied to local clustering [322]: fix the seed vertex at a high potential and a random *sink vertex* 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. The 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 pitfall is that if the random sink may be selected "too close" to the seed vertex, the voltage values do not work for clustering purposes; this can be avoided by repeating the calculation over a set of random sinks — for each repetition, the cluster of the seed vertex is calculated by locating the biggest drop of the voltage levels and majority vote is used to combine the repetitions [322]. Another straightforward approach is to take the averages of the voltages for each vertex over the repetitions and cluster with the final values.

It is simple to convert an unweighted graph into a circuit understood by the SPICE [264, 263] resistor circuit. 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 `rv1v2`, uses the vertices as "intersections" in the circuit (labeled `1` and `2`). In SPICE syntax,

```
rv1v2 1 2 1k
```

creates such a resistor with label `v1v2`. Each edge corresponds to such a resistor-definition line, and a voltage source (such as a battery) is placed between a source vertex (in this case with label 3) and a sink vertex (vertex 4) by

```
v1 4#PLUS# 3#MINUS# dc 2
```

with adjustable battery voltage (here set at two volts). The simulator can then calculate and report the voltage for each resistor, among other things.

The problem of choosing a random sink can be avoided by considering diffusion in an *unbounded* medium instead of a finite electrical circuit. We fix a seed vertex v_s at zero potential and obtain similarities as the solution to the discrete Dirichlet problem [68] imposing the boundary condition of keeping v at zero potential on G [250]. The transition matrix $\mathbf{P}$ corresponding to a graph $G = (V, E)$ is

$$p_{v,w} = \begin{cases} \frac{1}{\deg(v)}, & \{v, w\} \in E, \\ 0, & \text{otherwise.} \end{cases} \tag{4.6}$$

Fixing a seed vertex v_s to zero potential corresponds to setting $p_{v_s,w} = 0$ for all $w \neq v_s$ and $p_{v_s,v_s} = 1$. We find the right eigenvector $\mathbf{f}$ that corresponds to the second largest eigenvalue of modified $\mathbf{P}$ and use the values of the vector as the similarity values for finding the cluster of the seed vertex v_s. For more intuition on electrical networks, we recommend the book of Doyle and Snell [104].

Recall from Section 3.1.1 that the second eigenvalue is closely related to the mixing time of the chain, providing an intuition that the components of the second eigenvector serve as some kind of "proximity" measure for how long it takes for the walk to reach each vertex. This observation is the foundation of the global (usually hierarchical and divisive) spectral clustering methods. We propose a local method based on the same technique.

We call the components of the vector the *Fiedler values* of the vertices of the graph with respect to the seed vertex v_s, with reference to the work of Fiedler on matrices and graph connectivity [117, 118]. The Fiedler values of vertices in $\mathcal{C}(v_s)$ differ clearly from those outside the cluster, as will be demonstrated later in this section as well as in Section 4.4. Similar work with Fiedler vectors on document clustering of the WWW is reported by He *et al.* [147], who also observe a clear difference in the Fiedler values of those vertices that belong in different clusters for a 2-classification task. Our goal is, however, to define a local method to identify the cluster of a given seed vertex, whereas He *et al.* do not find clusters for determined seed vertices.

Another approach to computing the same vector is to use the Dirichlet matrix [250]. Denote the *(Dirichlet-)Fiedler* vector for a given graph G and seed vertex v_s by $\mathbf{f}$. It has $\mathbf{f}(v_s) = 0$. The vector elements with values close to zero for correspond to vertices that are "close" to v_s in the sense that they intuitively belong to the cluster of v. In the case that there are more seed vertices, they all can be fixed to zero to find their shared cluster. This however may produce unnatural results if the seed vertices themselves should not share a cluster.

The Fiedler vector $\mathbf{f}$ can be computed by minimizing the degree-adjusted *Rayleigh quotient* [69, 68]:

$$\sigma_1 = \inf_{\mathbf{f}} \frac{\sum_{\{v,w\}\in E}(\mathbf{f}(w) - \mathbf{f}(v))^2}{\sum_w \deg(w) \cdot \mathbf{f}^2(w)}, \tag{4.7}$$

where the infimum is computed over such vectors $\mathbf{f}$ that satisfy the boundary condition $\mathbf{f}(v_s) = 0$. This equality allows local approximation of $\mathbf{f}$. As the normalization of the final Fiedler vector $\mathbf{f}$ is unrestricted, we constrain the minimization to vectors that satisfy

$$|\mathbf{f}| \doteq \sum_w \deg(w) \cdot \mathbf{f}^2(w) = 2m, \tag{4.8}$$

where $2m$ is chosen as it gives an upper bound to the value of the sum when $\mathbf{f}(w) \in [0, 1]$. Hence we wish to find a vector $\mathbf{f}$ for which

$$\mathbf{f} = \operatorname{argmin}\left\{ \sum_{\{v,w\}\in E} \big(\mathbf{f}(w) - \mathbf{f}(v)\big)^2 \mid \mathbf{f}(v_s) = 0 \wedge |\mathbf{f}| = 2m \right\}. \tag{4.9}$$

A related method for global clustering with similar computations is proposed by Ding and He [99]. However, we perform neither exact nor global computations. The first useful observation is that Equation 4.9 can be approximated by weighting the constraint $|\mathbf{f}| = 2m$ with $c > 0$ and using the *gradient-descent* method to optimize the following function:

$$\mathcal{F}(\mathbf{f}) = \frac{1}{2} \sum_{\{v,w\}\in E} \big(\mathbf{f}(w) - \mathbf{f}(v)\big)^2 + \frac{c}{2}\Big(2m - \sum_w \deg(w) \cdot \mathbf{f}(v)^2\Big). \tag{4.10}$$

A second, even more fruitful observation is that the objective function has partial derivatives

$$\frac{\partial \mathcal{F}(\cdot)}{\partial \mathbf{f}\,(w)} = -\sum_{\{v,w\}\in E} \mathbf{f}\,(v) + (1-c)\cdot \deg\,(w)\cdot \mathbf{f}\,(w)\,, \tag{4.11}$$

a simple form which allows *local computation* of the gradient-descent step for each vertex v, only accessing the current estimate for $\mathbf{f}\,(v)$ together with the estimates for vertices in $\Gamma\,(v)$. We initialize a gradient-descent iteration with $\mathbf{f}\,(v_s) = 0$ and $\mathbf{f}\,(v) = 1$ for all $v \in V$, $v \neq v_s$, choosing a descent-speed parameter $\delta > 0$, and taking the iteration steps defined by

$$\tilde{\mathbf{f}}\,(w)_{t+1} = \min\left\{1,\, \tilde{\mathbf{f}}\,(w)_t + \delta\cdot\gamma\right\}, \tag{4.12}$$

where the minimum is taken to ensure that no value exceeds one and

$$\gamma = \sum_{\{v,w\}\in E} \tilde{\mathbf{f}}\,(v)_t - (1-c)\cdot \deg\,(w)\cdot \tilde{\mathbf{f}}\,(w)_t\,. \tag{4.13}$$

Should $\tilde{\mathbf{f}}\,(\cdot)$ become negative, the descent is too rapid and a smaller value for δ should be chosen. The effect of the normalization $|\mathbf{f}| = 2m$ is that most vertices in G will have $\mathbf{f}\,(v) \approx 1$, assuming that

$$|\mathcal{C}\,(v_s)| \ll n, \tag{4.14}$$

which holds for many natural networks that have a clustered structure.

Note that we only need to compute the descent step at time t for vertices v that have $\tilde{\mathbf{f}}\,(w)_{t-1} < 1$ for some $w \in \Gamma\,(v)$. Simple Matlab codes for computing both the exact values and our approximation are given in appendices. Note that in an efficient implementation it is only needed to store the estimates that are less than one. If the cluster $\mathcal{C}\,(v_s)$ is small compared to the order of the graph, this brings considerable savings in both memory usage and computational load especially for large instances. We observed clear improvement in runtime when including this optimization in our implementation of the method.

The apparent disadvantages of the method are having to choose the weight constant c and the descent-speed parameter δ, as well as deciding when to stop the iteration. The parameters need to be chosen such that the descent is not too rapid, i.e., that the estimate vector $\tilde{\mathbf{f}}$ remains non-negative. The changing either one of the constants c and δ will affect the iteration. We used the following heuristics to choose initial values for both and hand-tuned these in situations where too rapid of a descent occurred: estimate the average degree $\bar{k}$ of the graph, set $c = \bar{k}^{-2}$, and $\delta = 0.1c$. For determining when to stop iteration, one can either limit the maximum number of steps or setting a threshold ξ such that we stop the iteration when the maximum change in the estimate vector $\tilde{\mathbf{f}}$ is no greater than ξ; we usually started with $\xi = 0.1\delta$ and adjusted it manually if the descent did not appear to converge before reaching the threshold.

The estimates of the Fiedler vector thus obtained can be used as a similarity measure for clustering, similarly to the aforementioned voltages of Wu and Huberman [322], but with local computation. For both these methods, it remains an open problem what would be the best way to choose the cluster for v_s given such similarity values. From Figure 4.2 it is easy to see that the vertices included in the cluster have Fiedler values clearly differing from those not included in the cluster, and hence the task is relatively simple. A standard 2-classifier such as the k-means algorithm for points in space (in both of these cases, the data is one-dimensional) can be used to separate $\mathcal{C}(v_s)$ from the rest of the graph, but also other options are available. Ideally the separation process could be combined with the calculation of the similarities. The stronger the contrast between the similarity values, the easier the classification task becomes; the approximate Fiedler vectors have a strong contrast between the cluster members and the outsiders for each cluster (as illustrated in Figure 4.2) and hence gives an easy input set for the classification task.

Figure 4.2 shows in addition to the Fiedler and Wu-Huberman values a matrix for the mean absorption times of Markov chains corresponding to the same graph. As explained in Section 3.1, the mean absorption time of a state is the mean first-passage time in a modified chain where that state is made absorbing. Mean first-passage time can be expressed in terms of the stationary distribution vector π, which is the left eigenvector corresponding to the eigenvalue 1 of the transition matrix, and the unit vector $\mathbf{E} = (1, 1, \ldots, 1)$, which is the corresponding right eigenvector as [73]

$$f_i = \pi i(\mathbf{I} - \mathbf{P}_{i,i})^{-1}\mathbf{E}. \tag{4.15}$$

Hence using $\mathbf{P}$ of a modified chain with i as an absorbing state yields the mean absorption time in terms of the stationary distribution. The Fiedler values are the elements of the eigenvector related to the second eigenvalue of the same $\mathbf{P}$. As seen in Figure 4.2, both of these measures indicate a cluster structure, but it is more emphasized in the Fiedler vectors.

The local search approach outlined in Section 4.2.1 can also be employed for clustering with exact or approximate Fiedler values. First assume that in addition to $G = (V, E)$ and v_s, also the corresponding Fiedler vector (possibly approximated) $\mathbf{f}$ is given. We base our fitness function on the the *weighted Cheeger ratios* [68]

$$\mathcal{R}(\mathcal{C}(v_s)) = \frac{\displaystyle\sum_{w \in \mathcal{C}(v_s)} \sum_{\substack{\{v,w\} \in E \\ v \notin \mathcal{C}(v_s)}} \omega(\{v, w\})}{\displaystyle\sum_{w \in \mathcal{C}(v_s)} \sum_{\{v,w\} \in E} \omega(\{v, w\})}, \tag{4.16}$$

where the edge weights $\omega(\{v, w\}) > 0$ can be chosen appropriately. Low $\mathcal{R}(\mathcal{C}(v_s))$ implies little connectivity between the cluster and the rest of the graph and high density within the cluster (both in a weighted sense), which agrees with the criteria used above to define clusters. Hence we choose an edge-weight function — for example,

$$\omega(\{v, w\}) = \Big(|\mathbf{f}(w) - \mathbf{f}(v)| \Big)^{-1} \tag{4.17}$$

gives good experimental results — and use simulated annealing with the same modification steps as used above to minimize $\mathcal{R}(\cdot)$ of Equation 4.16 to select $\mathcal{C}(v_s)$.

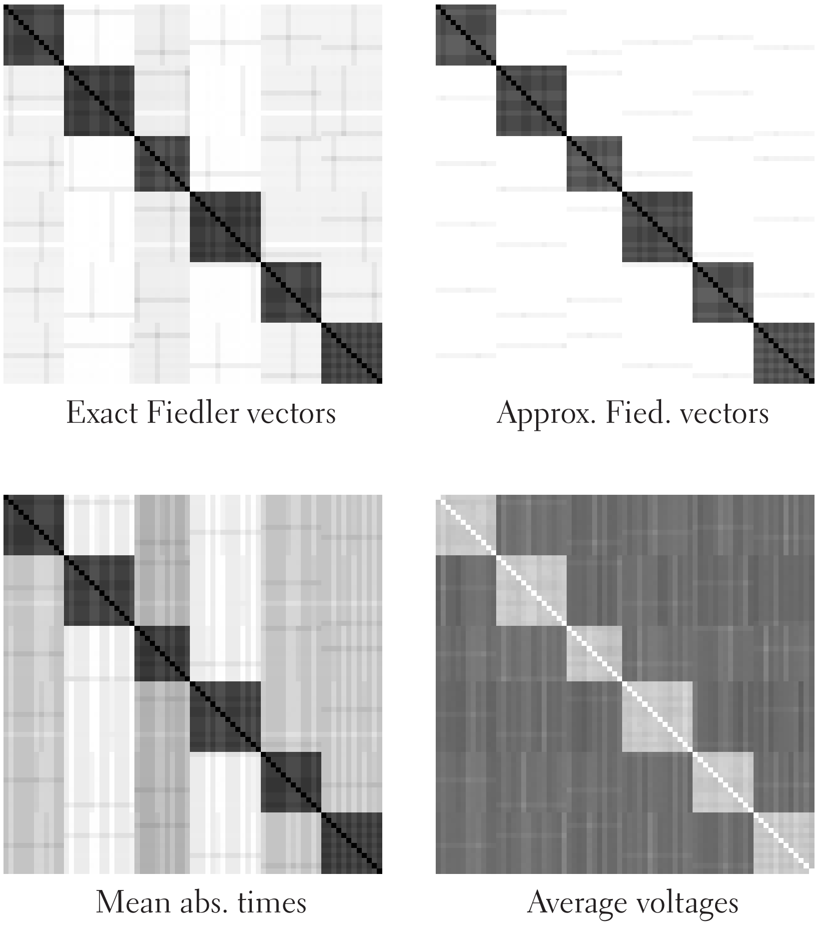


Figure 4.2: Visualizations of different similarity measures for a 75-vertex graph with a clear six-cluster structure, vertices sorted by cluster and the similarity vectors computed for each vertex composed into a matrix where the value range has been mapped into the unit line [0, 1] which has been further mapped into 256 gray-scale colors, with one mapped to white and zero mapped to black. The top-row matrices show the Fiedler values, both exact and approximate, computed with Matlab (code included in the appendix). The bottom-left matrix shows the mean absorption times (see Equation 3.5 along with the explanation of mean absorption times on page 21) from the other vertices to the seed vertex for a random walk on the graph, and the bottom-right has the voltages used in the clustering algorithm by Wu and Huberman [322], averaged over 500 random sink-source pairs.

It is also possible to perform essentially the same process *without* precomputed Fiedler estimates setting the initial estimates as in the gradient descent and computing the descent steps for vertices on demand. For example, an iteration can be performed for v and w every time $\omega(\{v, w\})$ is needed for the computation of the fitness function. The longer the simulated annealing runs, the closer the Fiedler estimates get to those obtained by the local preprocessing described above, which implies that vertices that are not neighbors of the current cluster candidate do not need to be accessed. Updating the estimates every time an edge weight is needed ensures that the estimates for the current cluster candidate are frequently updated, and since $v_s \in \mathcal{C}(v_s)$ at all times during the search, these are the same vertices that would be expected to get a Fiedler value lower than one in the preprocessing as well. Note that those vertices which at no point become neighbors of a cluster candidate are not accessed at all during the clustering.

4.2.3 Cluster fitness functions

Instead of calculating similarities for the vertices on which to base the clustering, it is also possible to define fitness functions that evaluate the suitability of a given (connected) subgraph containing v_s as $\mathcal{C}(v_s)$. In order to locate the cluster for a seed vertex v in $G = (V, E)$, we define a fitness function that "rates" (connected) subgraphs of G so that we may stochastically examine possible solutions. The *stochastic search* algorithm then chooses the candidate that gives the best value for the fitness function among those examined; due to its stochasticity, the search will not explore the entire solution space but will attempt, with probabilistic decision-making, to study a limited region that assumably contains good if not the best solutions. A similar approach for points in space has been suggested [42, 187].

The cluster fitness functions should be *locally computable*, meaning that the fitness of a cluster candidate should not depend on global properties of the graph. We begin the derivation of possible fitness functions by listing measures that can be calculated for a given subgraph resorting *only* to information on the included vertices and their neighbors. For unweighted graphs, this means that only the adjacency lists of the included vertices may be accessed. For edge weights, the weight of each edge that has at least one endpoint in the subgraph is considered locally available, and for vertex weights, the weights of the included vertices *and* their immediate neighbors are considered *locally available*.

We begin by considering measures for determining whether a cluster $\mathcal{C}$ is a good cluster for a particular vertex $v \in \mathcal{C}$ in a given graph $G = (V, E)$. We classify the edges incident on v into two groups: *internal* edges that connect v to other vertices in $\mathcal{C}$, and *external* edges that connect v to vertices that are not included in the cluster $\mathcal{C}$,

$$\begin{aligned}
\deg_{\text{int}}(v, \mathcal{C}) &= |\Gamma(v) \cap \mathcal{C}| \\
\deg_{\text{ext}}(v, \mathcal{C}) &= |\Gamma(v) \cap (V \setminus \mathcal{C})| \qquad (4.18) \\
\deg(v) &= \deg_{\text{int}}(v, \mathcal{C}) + \deg_{\text{ext}}(v, \mathcal{C}).
\end{aligned}$$

Clearly, if $\deg_{\text{ext}}(v) = 0$, $\mathcal{C}$ is a good cluster, as v has no connections outside of it. Also, if $\deg_{\text{int}}(v) = 0$, there is no reason why v should be included in $\mathcal{C}$ as it is not connected to any of the vertices in it. In general, the higher the internal degree $\deg_{\text{int}}(v)$, 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 neighbors that a vertex could have in $\mathcal{C}$, namely $|\mathcal{C}| - 1$, to obtain a measure in $[0, 1]$:

$$\delta(v, \mathcal{C}) = \frac{\deg_{\text{int}}(v, \mathcal{C})}{|\mathcal{C}| - 1}. \tag{4.19}$$

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 also 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:

$$\rho(v, \mathcal{C}) = \frac{\deg_{\text{int}}(v, \mathcal{C})}{\deg(v)}. \tag{4.20}$$

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 (Equation 4.19):

$$\begin{aligned} \delta_s(\mathcal{C}) &= \frac{1}{|\mathcal{C}|} \sum_{v \in \mathcal{C}} \delta(v, \mathcal{C}) \\ &= \frac{1}{|\mathcal{C}|(|\mathcal{C}| - 1)} \sum_{v \in \mathcal{C}} \deg_{\text{int}}(v, \mathcal{C}). \end{aligned} \tag{4.21}$$

We define the *internal degree* of a cluster $\mathcal{C}$ to be the number of edges connecting vertices in $\mathcal{C}$ to each other:

$$\deg_{\text{int}}(\mathcal{C}) = |\{\{v, w\} \in E \mid v, w \in \mathcal{C}\}|. \tag{4.22}$$

The sum of the internal degrees of vertices in $\mathcal{C}$ is clearly twice the internal degree of the cluster, as each internal edge is counted independently by both of its endpoints. This simplifies Equation 4.21 into

$$\delta_s(\mathcal{C}) = \frac{1}{|\mathcal{C}|(|\mathcal{C}| - 1)} \cdot 2\deg_{\text{int}}(\mathcal{C}) = \frac{\deg_{\text{int}}(\mathcal{C})}{\binom{|\mathcal{C}|}{2}} = \delta(\mathcal{C}) \tag{4.23}$$

obtaining exactly the *local density* of the subgraph induced by the vertex set $\mathcal{C}$. Clearly, optimizing $\delta(\mathcal{C}) \in [0, 1]$ alone makes small cliques superior to larger but slightly sparser subgraphs, which is often impractical. 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 natural cluster of a vertex, however, 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 of two included edges being connected, and the higher the probability, the more tightly connected the cluster.

Also algorithms that search for maximal subgraphs that have a density higher than a preset *threshold* have been proposed [185]. Without a threshold such an algorithm would search for complete subgraphs, 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. Another approach was proposed by Matsuda *et al.* [218] who consider p-quasi complete subgraphs as clusters; a graph $G = (V, E)$ is *p-quasi complete* for $p \in [0, 1]$, if for all $v \in V$,

$$\deg(v) \geq p(n-1). \tag{4.24}$$

The connection probability p is given as a parameter to their algorithm. They show that it is **NP**-complete to determine whether a given graph has a 0.5-quasi complete subgraph of order at least k. Hence they conclude that approximation algorithms are the only feasible approach for locating such subgraphs [218]. Holzapfel *et al.* [155] discuss the computational complexity of a related problem, namely the detection of clusters that have average degree above a given threshold.

The introversion of a cluster $\mathcal{C}$ can be similarly characterized by summing the suitability measures of Equation 4.20 and scaling with the cluster order to remain in [0, 1]:

$$\rho_s(\mathcal{C}) = \frac{1}{|\mathcal{C}|}\sum_{v\in\mathcal{C}}\rho(v,\mathcal{C}) = \frac{1}{|\mathcal{C}|}\sum_{v\in\mathcal{C}}\frac{\deg_{\text{int}}(v,\mathcal{C})}{\deg(v)}. \tag{4.25}$$

If we calculate the capacity of the cut $(\mathcal{C}, V \setminus \mathcal{C})$ (Equation 3.20 on page 23) for a uniform random walk with transition probabilities as in Equation 3.26 on page 25, we obtain

$$\begin{aligned} Q(\mathcal{C}, V\setminus\mathcal{C}) &= \sum_{\substack{v\in\mathcal{C},\\ w\in V\setminus\mathcal{C}}} p_{v,w} \\ &= \sum_{\substack{v\in w,\\ w\in(\Gamma(v)\cap(V\setminus\mathcal{C}))}} \frac{1}{\deg(v)} = \sum_{v\in\mathcal{C}}\frac{\deg_{\text{ext}}(v,\mathcal{C})}{\deg(v)}. \end{aligned} \tag{4.26}$$

As $\deg_{\text{ext}}(v,\mathcal{C}) = \deg(v) - \deg_{\text{int}}(v,\mathcal{C})$ by definition, there is an obvious relation between the capacity of the cut and Equation 4.25 above. A cluster is properly introvert if Equation 4.25 has a high value and the capacity of the cut is low. The cut capacity used above as an introversion measure is used to define the conductance of a graph, which has been used in global clustering algorithms [170], for example, recursively removing a cut from the similarity graph of the data set such that the removed cut has a conductance almost at low as the minimum over all cuts [62]. Another related measure used for clustering is the *relative density* $\rho(\mathcal{C})$ [223] that is defined in terms of the internal degree $\deg_{\text{int}}$ of Equation 4.22 and *external degree* of a cluster $\mathcal{C}$, which is the size of the cut $(\mathcal{C}, V \setminus \mathcal{C})$,

$$\deg_{\text{ext}}(\mathcal{C}) = \left|\{\{v,w\} \in E \mid v \in \mathcal{C}, w \in V\setminus\mathcal{C}\}\right|, \tag{4.27}$$

as the ratio of the internal degree to the number of incident edges:

$$\rho(\mathcal{C}) = \frac{\deg_{\text{int}}(\mathcal{C})}{\deg_{\text{int}}(\mathcal{C}) + \deg_{\text{ext}}(\mathcal{C})} = \frac{\sum_{v \in \mathcal{C}} \deg_{\text{int}}(v, \mathcal{C})}{\sum\limits_{v \in \mathcal{C}} \deg_{\text{int}}(v, \mathcal{C}) + 2\deg_{\text{ext}}(v, \mathcal{C})}. \quad (4.28)$$

For cluster candidates with only one vertex as well as all larger independent sets, we set $\rho(\mathcal{C}) = 0$. Also this measure favors subgraphs with few connections to other parts of the graph. All of these 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 4.25. 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.

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

$$\sum_{i=1}^{k} \Big(\deg_{\text{int}}(\mathcal{C}_i) + \deg_{\text{ext}}(\mathcal{C}_i) \Big) = m + \sum_{i=1}^{k} \deg_{\text{ext}}(\mathcal{C}_i), \quad (4.29)$$

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 favor 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 $\frac{\deg_{\text{int}}(\mathcal{C}_i)}{\deg_{\text{ext}}(\mathcal{C}_i)}$. 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 Equation 4.28.

A good cluster has both high density and high introversion (i.e., low capacity). Possible combinations of the above measures are numerous; in this work we concentrate on examining the behavior of the product of local (Equation 4.23) and relative (Equation 4.28) densities as a cluster quality measure:

$$\mathcal{F}(\mathcal{C}) = \delta(\mathcal{C}) \cdot \rho(\mathcal{C}) = \frac{2\deg_{\text{int}}(\mathcal{C})^2}{|\mathcal{C}|\ (|\mathcal{C}| - 1)(\deg_{\text{int}}(\mathcal{C}) + \deg_{\text{ext}}(\mathcal{C}))}. \quad (4.30)$$

For weighted graphs with weights in [0, 1], we simply replace the internal and external edge counts by the sums of the weights of the corresponding edges.

The local density can be interpreted as the probability that two randomly chosen cluster members are connected by an edge, whereas the relative density is the probability that a randomly chosen edge incident on the cluster is an internal 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 (which can be formulated into a quality measure, as will be discussed in Section 4.3). 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.

Therefore, when building clusters one by one, we aim to form every single cluster such that it has only a few edges connecting it to other clusters (i.e., high relative density) and that as many as possible of the cluster members would share an edge (i.e., high local density). A low value for either measure indicates that the cluster boundary is badly chosen (allowing too big of a portion of the edges to connect to outside vertices or including poorly connected vertices in the same cluster). Optimizing the product of Equation 4.30 yields the desired behavior in a simple form.

The complexity of optimizing Equation 4.30 can be studied through the decision problem of whether a given graph G has a k-vertex subgraph $\mathcal{C}$ for which $\mathcal{F}(\mathcal{C}) \geq \gamma$ for some fixed $k \in \mathbb{N}$ and $\gamma \in [0, 1]$. Especially, we are interested in knowing whether there is such a subgraph that contains a given vertex v. Both $\delta(\mathcal{C})$ and $\rho(\mathcal{C})$ alone correspond to **NP**-complete decision problems of the following form [284]:

Maximum Density (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 $\delta(S) \geq \xi$?

When the density measure used is $\delta(\mathcal{C})$, we call the problem LOCAL DENSITY; it is **NP**-complete since for $\xi = 1$ it coincides with the **NP**-complete CLIQUE problem [181]. Using $\rho(\mathcal{C})$, we call the resulting problem RELATIVE DENSITY, which is clearly a problem in class **NP** since a nondeterministic algorithm can guess a cut $S \subseteq V$ of order k and verify in polynomial time that the relative density is above the threshold ξ. The following minimum bisection problem on cubic graphs is known to be **NP**-complete [50] and can be reduced to RELATIVE DENSITY in polynomial time:

Minimum Bisection for Cubic Graphs (CUBIC MIN BISECTION)
Instance: A cubic graph $G = (V, E)$ with n vertices and positive integer b.
Question: Is there a cut $S \subseteq V$ such that $|S| = \frac{n}{2}$ and the cut size is smaller than the bound b, $\deg_{\text{ext}}(S) \leq b$?

Given a CUBIC MIN BISECTION instance G of order n, and a positive integer b, a corresponding RELATIVE DENSITY instance consists of the same graph G, with parameters $k = \frac{n}{2}$ and

$$\xi = \frac{3n - 2b}{3n + 2b}. \tag{4.31}$$

For *any* subset $S \subseteq V$ such that $|S| = k = \frac{n}{2}$ it holds that the total number of edges incident on S is

$$\frac{3\,|S| - \deg_{\text{ext}}(S)}{2} = \frac{3n - 2\deg_{\text{ext}}(S)}{4} \tag{4.32}$$

due to G being a cubic graph. This allows us to write the relative density in terms of the cut size only, as the external degree and the internal degree of S together form the set of all incident edges:

$$\rho(S) = \frac{3n - 2\deg_{\text{ext}}(S)}{3n + 2\deg_{\text{ext}}(S)}, \tag{4.33}$$

which combined with the threshold of Equation 4.31 yields $\rho(\geq)\,\xi$ if and only if $\deg_{\text{ext}}(S) \leq b$.

4.2.4 A stochastic algorithm

Calculation of the fitness measure of Equation 4.30 only requires the adjacency lists of the included vertices. Therefore, a good approximation of the optimal cluster for a given vertex can be obtained by local search with the method discussed in Section 4.2.1, using Equation 4.30 as the fitness function of a simulated annealing procedure. The neighborhood over cluster candidates for the seed vertex v is defined such that if $\mathcal{C}$ can be transformed into $\mathcal{C}'$ by adding a vertex adjacent to at least one vertex included in $\mathcal{C}$ or by removing an included vertex along with any vertex that is not in the connected component of the seed vertex v after the removal (easily determined by a depth-first search). The pseudo code of a "skeleton implementation" is given in Table 4.1.

The method is well-suited for memory-efficient implementation: if the graph is stored as adjacency lists of the form

$$\langle v \,:\, w_1, w_2, \ldots, w_{\deg(v)} \rangle, \tag{4.34}$$

only one such entry at a time needs to be retrieved from memory. For n vertices, the entries can be organized into a search tree with $\mathcal{O}(\log n)$ access time. The search needs to maintain only the following information:

(a) the list of currently included vertices $\mathcal{C}$,

(b) the current internal degree $\deg_{\text{int}}(\mathcal{C})$ (Equation 4.22), and

(c) the current external degree $\deg_{\text{ext}}(\mathcal{C})$ (Equation 4.27).

When a vertex v is considered for addition into the current cluster candidate $\mathcal{C}$, its adjacency list is retrieved and the degree counts for the new candidate $\mathcal{C}' = \mathcal{C} \cup \{v\}$ are calculated as follows:

$$\deg_{\text{int}}(\mathcal{C}') = \deg_{\text{int}}(\mathcal{C}) + k, \qquad \deg_{\text{ext}}(\mathcal{C}') = \deg_{\text{ext}}(\mathcal{C}) - k + \ell, \tag{4.35}$$

where $k = |\mathcal{C} \cap \Gamma(v)|$ and $\ell = \deg(v) - k$. The removal of vertices from a cluster candidate is done analogously, subtracting from the internal degree the lost connections and adding them to the external degree.

Table 4.1: An algorithm that finds the cluster $\mathcal{C}(v)$ of a specified seed vertex v in a given graph G, assuming that the graph is represented as a set of adjacency lists L. The subroutine Modify takes as a parameter the seed vertex and a cluster candidate and selects a neighboring cluster candidate. R is the number of iterations taken by the simulated annealing algorithm, each consisting of S modification steps. $\mathcal{T}_0$ is the initial temperature of the simulated annealing algorithm, and α is the cooling constant. $\mathcal{F}(\mathcal{C})$ is the fitness function of Equation 4.30. The procedure UniformRandom() acts as a random variable $X \sim \text{Uniform}(0, 1)$: each call made to it returns a new uniformly distributed real number.

$\mathcal{C}_b := \{v\}$. `// Best cluster found`
$\mathcal{F}_b := 0$. `// Fitness of the best cluster`
For $i \in [1,\ R]$, `// Iterations`
    $\mathcal{C} := \Gamma(v) \cup \{v\}$.
    $\mathcal{T} := \mathcal{T}_0$.
    $\mathcal{F}_p := \mathcal{F}(\mathcal{C})$.
    For $j \in [1,\ S]$, `// Steps per iteration`
        $\mathcal{C}' := \text{Modify}(v, \mathcal{C})$.
        $\mathcal{F}_c := \mathcal{F}(\mathcal{C}')$.
        If $\left((\mathcal{F}_c \geq \mathcal{F}_p) \vee \left(\text{UniformRandom}() < e^{\frac{\mathcal{F}_c - \mathcal{F}_p}{\mathcal{T}}}\right)\right)$,
            then $\mathcal{C} := \mathcal{C}'$, $\mathcal{F}_p := \mathcal{F}_c$.
            If $(\mathcal{F}_c > \mathcal{F}_b)$,
                then $\mathcal{F}_b := \mathcal{F}_c, \mathcal{C}_b := \mathcal{C}$.
Return $\mathcal{C}_b$.

The memory consumption of the local algorithm is determined by the local structure of the graph. The order of the initial cluster is limited from above by the maximum degree of the graph Δ plus one; in natural graphs, usually $\Delta \ll n$ and $|\mathcal{C}| \ll n$. Hence examining the adjacency lists of the vertices included in the final cluster candidate takes $\mathcal{O}(\Delta \cdot |\mathcal{C}|)$ operations. Redefining

$$\deg_{\text{ext}}(\mathcal{C}) = |\{\langle v, w\rangle \in E \mid v \in \mathcal{C}, w \in V \setminus \mathcal{C}\}| \tag{4.36}$$

allows for clustering directed graphs. The method has been applied to obtain a clustering of a 32,148-vertex directed graph representing the Chilean inter-domain link structure [305].

4.3 EVALUATION OF CLUSTERINGS

For traditional methods of clustering points in space, clusters that are of very different sizes or shapes often produce difficulties, as well as clusters that overlap each other [123]. In graph clustering, this implies that when the clusters are of different orders and have varying densities, global methods tend to run into difficulties in correctly classifying them. Attempting to overcome difficulties caused by different densities or other properties of the data set, most clustering algorithms require as input several parameters in addition to the data to be clustered. Determining proper values for the parameters is usually nontrivial or even impossible, and the methods may be highly sensitive to the parameter values. Hence a practical clustering algorithm should require few if any parameters. It should also be insensitive to small changes in the parameter values.

Cluster fitness functions, when not used in the clustering algorithm itself, can also be used to evaluate the clusterings produced and especially to choose between two alternative clusterings, preferring those clusterings that yield high-fitness clusters. Extending the definitions of internal and external degrees for a global clustering $\mathcal{C}_1, \ldots, \mathcal{C}_k$ of a graph $G = (V, E)$ as

$$\begin{aligned}
\deg_{\text{int}}(\mathcal{C}_1, \ldots, \mathcal{C}_k) &= \sum_{i=1}^{k} \deg_{\text{int}}(\mathcal{C}_i) \qquad \text{and} \\
\deg_{\text{ext}}(\mathcal{C}_1, \ldots, \mathcal{C}_k) &= \tfrac{1}{2}\sum_{i=1}^{k} \deg_{\text{ext}}(\mathcal{C}_i),
\end{aligned} \tag{4.37}$$

we may use the local or relative densities or the combination fitness function of Equation 4.30 to evaluate the quality of a given global clustering. Although optimizing such measures is computationally demanding, evaluating them for a given clustering of a given graph is a light-weight operation.

In addition, given a clustering $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k$ for $G = (V, E)$, the following measure [238] (known as *modularity*) may be used to evaluate its quality:[2]

$$\mathcal{Q}(\mathcal{C}_1, \ldots, \mathcal{C}_k) = \sum_{i=1}^{k} \mathcal{E}_{ii} - \sum_{\substack{i \neq j \\ i,j \in \{1,\ldots,k\}}} \mathcal{E}_{ij}, \tag{4.38}$$

[2]Many formulations of essentially the same quality measure exist, depending on whether the graph is weighted and whether minimization or maximization is used.

where

$$\mathcal{E}_{ij} = \sum_{\substack{\{v,w\} \in E \\ v \in \mathcal{C}_i,\, w \in \mathcal{C}_j}} \omega\left(\{v, w\}\right), \tag{4.39}$$

with a single edge $\{v, w\} \in E$ only included once in the summation. In terms of our previous definitions, for an undirected and unweighted graph, using constant unit weights we have

$$\begin{aligned} \deg_{\text{int}}\left(\mathcal{C}_i\right) &= \mathcal{E}_{ii} \\ \deg_{\text{ext}}\left(\mathcal{C}_i\right) &= -\mathcal{E}_{ii} + \sum_{j=1}^{k} \mathcal{E}_{ij}. \end{aligned} \tag{4.40}$$

The higher the value of $\mathcal{Q}\left(\mathcal{C}_1, \ldots, \mathcal{C}_k\right)$, the better the clustering, as the total weight of intra-cluster edges is large and the total weight of inter-cluster edges is small. The measure coincides with the intuition that vertices in the same cluster should be connected to each other with high probability and that the probability of a randomly chosen edge connects two clusters is low. This intuition of inter-cluster sparsity combined with intra-cluster density has been used by Brandes *et al.* [45] both with formulations similar to the above equation and conductance-based notions to evaluate the performance of clustering algorithms.

Equation 4.38 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 n-dimensional space [161], closely related to the Davies-Bouldin index [93]. Implementing a clustering algorithm to directly optimize the measure would suffer from the same scalability problems than the general clustering methods, but for a moderate size graph a stochastic search could be implemented assigning vertices to initial clusters and then moving vertices from one cluster to another, using Equation 4.38 as a fitness function. Such a search space would unfortunately be large, as it would consist of all possible partitions of the vertex set, and global information would be needed to evaluate the fitness. Nonetheless, approximate methods may be build to optimize this measure [78, 239], as discussed earlier in this chapter.

Other measures for evaluating a single cluster are distance measures such the average or maximum distance, 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.* [320] compares the differences in average path lengths of the original graph and a graph where each cluster is contracted into a single vertex and distances are calculated by having that single vertex represent all of its "member vertices". This measure is called the *distortion of the graph geodesics*.

4.4 EXPERIMENTS

In this section we describe some of the experiments performed with the local methods presented earlier in this chapter. We have conducted experiments on natural and generated nonuniform random graphs.

4.4.1 Test data

We have used two main data sets for natural data: one being a large *collaboration graph* [234], which can be reasonably assumed to have the topology of a social network, and the other being a graph representing the inter-domain links of the Chilean Web Graph [305]. In this section we introduce the constructions of both of these natural graphs as well as explain the method used to generate artificial test data.

Our collaboration graph is based on bibliography files from The Collection of Computer Science Bibliographies [2], using only the mathematical bibliographies in BibTeX format.[3] The sample includes 379 files from the Ftp server of the Department of Mathematics at the University of Utah[4] and about 50 other files. The collaboration graph was constructed by processing the author fields of the files, attempting to ignore authors that are not persons (such as institutes and committees), to simplify the spelling of the names, ignoring Roman numerals, and to interpret which word is the first name and which is the last name of an author.

As the bibliographic data was somewhat diverse and especially all exotic names have varying forms of spelling even within just one bibliography file, we represented all authors with the same first initial and surname by the same vertex. For comparison we also tested a construction in which only the surname was used. Dashes and other such characters in the names were removed, and special Unicode characters were replaced by their Ascii counterparts, for example, replacing á and ä by a. Even with the above simplifications, more than 170,000 bibliographic entries with multiple authors were found. Each such entry is represented by a line in the parser output that defines the "vertex labels", which are the simplified last names of the authors, with duplicates eliminated. Multiple and reflexive edges were omitted.

The graph that results from joining all the above BibTeX files with just the surname as author identification has 78,758 vertices and 331,551 edges (referred to as G_{last}). Adding the first initials increases it to 129,215 vertices and 350,914 edges (G_{init}). More details of the construction are given in our earlier work [306]. The largest connected component of G_{last} has 73,707 vertices and 327,891 edges, therefore covering 93.6 % of the network. For G_{init}, the connected component covers 84.1 percent of the graph with 108,624 vertices and 333,546 edges.

The *Chilean Web Graph* (Cwg) was constructed from a data set of a complete crawl of all Chilean websites in 2002 [20]. We extracted from each web page its domain and placed a directed edge between two domains if the source domain contains a page that has a link to the target domain, i.e., an edge $\langle v, w\rangle$ signifies that there exists a link from at least one webpage under v to a webpage under w. Links pointing to non-Chilean websites, pages from non-Chilean websites, self-loops, and edge multiplicities (that could be used as weights in clustering) were ignored.

An early attempt in social sciences to capture the clustering properties of social networks was the *caveman graph*, produced by linking together a ring

[3]The data was collected from `http://liinwww.ira.uka.de/bibliography/Math/` on December 2, 2002; eight bibliographies were unavailable at the time.

[4]Available online at `ftp://ftp.math.utah.edu/pub/tex/bib`.

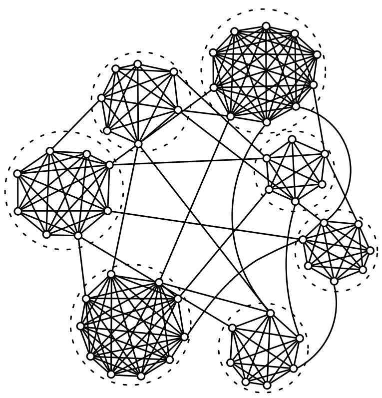


Figure 4.3: An example caveman graph with 55 vertices and 217 edges; each cave (encompassed by a dotted line) is correctly identified as a cluster by the local optimization of Equation 4.30.

of small complete graphs called "caves" by moving one of the edges in each cave to point to another cave [313]. In earlier work [306], we present a construction for creating hierarchical, probabilistic versions of caveman graphs. We call these the *generalized caveman graphs*. 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. For an example of a first-level construction, see Figure 4.3.

A cave at a certain level ℓ of the hierarchy is formed of a random number $r \in [n_{\min}, n_{\max}]$ of subcaves with connection probability $\min\{s \cdot p', 1\}$, where p' is the connection probability at level ℓ. Each subcave is either a hierarchical cave, or at the bottom level, a random graph of type $\mathcal{G}_{n,p_b}$ with a random $n \in [n_{\min}, n_{\max}]$; p_b is the connection probability of the bottom level. Caves that consist of subcaves are randomly connected into a larger graph; the connections are placed as in the $\mathcal{G}_{n,p}$ model, considering the subcaves as single vertices, the inter-cave connection being assigned to a random member at each subcave. These graphs all have high clustering and relatively short path length by construction unless both the initial connection probability and the scaling factor are set to produce sparse caves and a sparse hierarchy.

4.4.2 Locality and stability

For generalized caveman graphs, local optimization of Equation 4.30 correctly identifies any dense "cave" regardless of the starting point; an example is shown in Figure 4.4. As local search procedures are not prohibited from traversing further away in the graph or revisiting parts of the graph, we wish to examine whether the extent to which the search traverses the graph has a significant effect on the clusters that the algorithm chooses. Hence we clustered the largest connected component of G_{init} (a collaboration graph) by optimizing Equation 4.30 with simulated annealing, varying the number of independent restarts $R \in \{20, 40, \ldots, 100\}$ per search vertex and the number of cluster modification steps S taken after each restart for simulated annealing from 200 to 1,000 with increments of 100.

The Figure 4.5 shows the ratio of the number of vertices visited during the search to the final cluster order, averaged over 100 vertices selected uniformly at random; the final orders are plotted for reference. Figure 4.5 plots the ratio of the number of vertices visited and the final order of the selected cluster using R restarts of S steps averaged over 50 randomly selected vertices. The extent to which the graph is traversed grows much slower than the number of modification steps taken, implying high *locality* of the search. As the iteration count is increased, the relative difference gets smaller, which indicates that the number of vertices visited practically stops growing if the increasing possibility for random fluctuations is ignored. The distributions of the cluster orders over three R/S-pairs of the same graph are shown in Figure 4.6; the distribution hardly changes as the parameters are varied, indicating high stability of the method.

4.4.3 Comparison with global methods

We compared the clusterings obtained with the local optimization of Equation 4.30 to the clusterings of two global methods: GMC *(Geometric Minimum Spanning Tree Clustering)* with additional linear-time post-processing [45] and ICC *(Iterative Conductance Cutting)* [175]. The data set used consisted of generalized caveman graphs of different orders. For each graph, we compared the clusters of each vertex obtained with the three methods by calculating their *overlap*, i.e, what fraction (shown in percentages) of the vertices of a cluster $\mathcal{A}$ determined by one method are also included in the cluster $\mathcal{B}$ determined by another method.

Table 4.2 shows the results for a caveman graph with 1,533 vertices and 50,597 edges; the results for the smaller graphs allow the same conclusions. ICC splits the caves into small clusters, which is a sign that it fails to recognize the cave boundary on which the density jump takes place. GMC and the local method agree in a majority of cases exactly in cluster selection, and even when they differ, one is usually a subset of the other. GMC and ICC agree poorly with each other. For an illustration of this effect, Figure 4.7 shows a single cave, correctly identified as a cluster by the local method regardless of

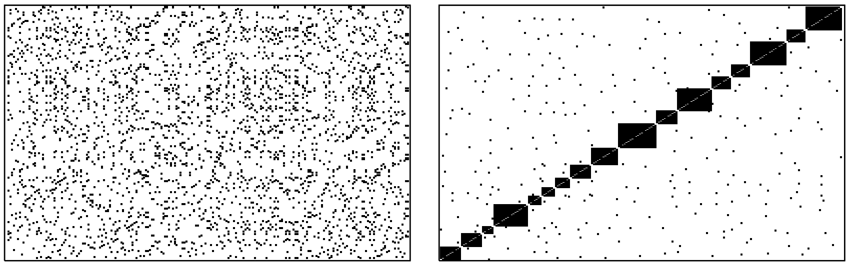


Figure 4.4: The adjacency matrix of a caveman graph with 210 vertices and 1,505 edges; the left one uses random vertex order, reflecting little structure, whereas the one on the right is sorted by the clusters found by locally optimizing Equation 4.30 — the sorted matrix clearly reveals the cave-based structure.

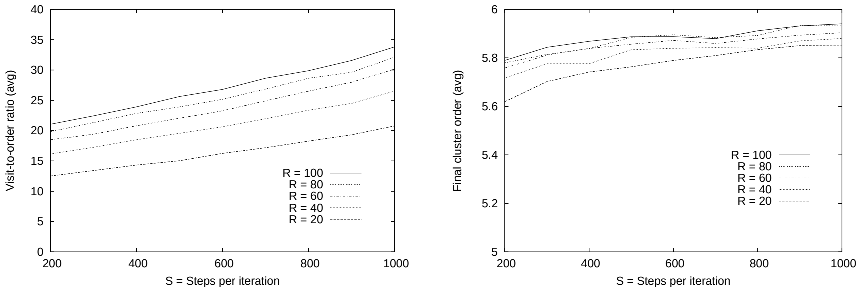


Figure 4.5: On the left, the ratio of the number of vertices visited (i.e., the visit count for an R/S-pair) to that of the number of vertices selected in the final cluster (i.e., the cluster order) averaged over 100 vertices selected uniformly at random and repeated 50 times per vertex. On the right, the average final cluster orders of the same experiment set.

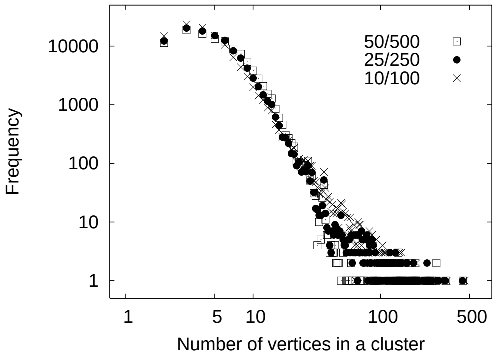


Figure 4.6: The distribution of the number of vertices per cluster for the largest connected component of G_{init} for three different R/S-pairs, where $R \in \{10, 25, 50\}$ and $S = 10R$.

the seed vertex, but split into disagreeing clusters by both global methods.

4.4.4 Fiedler clusters

For the Fiedler clustering method, we experimented on three types of networks: generalized caveman graphs, subsets of the collaboration graph, and a small social interaction graph.

Figure 4.11 on page 70 represents some of the results of the approximate Fiedler vector calculations on a 144-vertex caveman graph, starting from three different source vertices. For visual effect, the vertices are color-coded so that dark colors correspond to small approximated Fiedler potential values, with the source node in each case colored black. The parameter values used in this case were the standard ones derived from $\bar{k} = 6.14$ (i.e., $c = 0.027$, $\delta = 0.003$, $\varepsilon = 0.0003$, as explained on page 51).

It can be seen in the illustrations how well the method distinguishes the natural clusters embedded in the graph. The vertices selected by the Cheeger-ratio heuristic for the relevant clusters in each of the three cases are indicated by thickened node boundaries; also the clusters determined in this manner can be seen to correspond to the natural ones. A small graph of order 144

Table 4.2: Denote by $\mathcal{A}$ the cluster chosen by one method for vertex v, and by $\mathcal{B}$ the cluster chosen for v by another method. If the two methods agree, the *overlaps* $a = |\mathcal{A} \cap \mathcal{B}|/|\mathcal{B}|$ and $b = |\mathcal{A} \cap \mathcal{B}|/|\mathcal{A}|$ are high. For three clusterings of a caveman graph, the percentages p of vertices for which the values a and b fall into a certain range are shown. The values are to be interpreted as follows: if $a = a_1$ and $b = b_1$, then a_1 percent of cluster $\mathcal{B}$ (the method of the right column) is included in $\mathcal{A}$ (the method of the left column) and b_1 percent of cluster $\mathcal{A}$ is included in $\mathcal{B}$.

Local	GMC	%	Local	ICC	%	GMC	ICC	%
a	b	p	a	b	p	a	b	p
all	all	74	all	$(11, 14)$	45	all	$(5, 14)$	71
all	$(74, 95)$	14	all	$(22, 27)$	12	all	$(22, 34)$	10
all	$(2, 24)$	4	all	$(5, 7)$	36	all	$(40, 55)$	2
$(86, 97)$	all	5	all	$(46, 54)$	6	$(80, 91)$	$(7, 31)$	7
$(3, 57)$	$(5, 87)$	3				$[50, 67)$	$(5, 20]$	4
						$(71, 89)$	$(45, 55)$	3
						$(9, 46)$	$(2, 100]$	3

was chosen for illustrative purposes. Results on a bigger graph of order 1,533 are qualitatively similar, but the bigger graph is infeasible to be represented in such a drawing.

Another perspective on the data is provided by Figure 4.8, where the components of an approximate Fiedler vector corresponding to the middle one of the three local clusterings shown of Figure 4.11 are plotted. On the left, the components of the Fiedler vector (i.e., the Fiedler values) are ordered simply by node index, and on the right they are sorted in increasing order by component value. The vertical lines indicate the vertices selected for the cluster of the source node (which in this case has index 95) by the Cheeger ratio heuristic. In our generating process for the synthetic caveman graphs, vertices deemed to belong to the same cave are assigned consequent indices, and

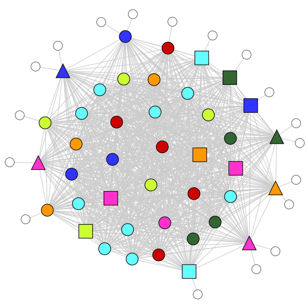


Figure 4.7: A single cave detached from a 649-vertex caveman graph; the small circles are neighbors in other caves. The shape of the vertex indicates its cluster for the post-processed GMC (with three clusters overlapping the cave) and the color indicates the clustering of ICC (seven clusters overlap); locally optimizing Equation 4.30 selects an entire cave using of any included vertex as the seed vertex.

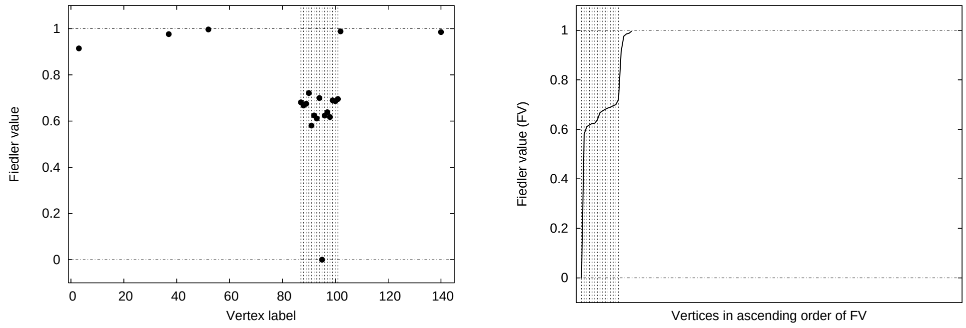


Figure 4.8: Components of a Fiedler clustering vector sorted by index (left) and in ascending order (right). The vertical lines represent the cluster selection of the Cheeger-ratio local optimization method based on the Fiedler values.

hence good clusterings should group in a band-like formation as observed on the left in Figure 4.8.

A 503-vertex subgraph of the collaboration network is shown in the left panel of Figure 4.9. The right panel shows three small collaborative clusters identified by the local Fiedler clustering method, starting from three distinct source vertices; the clusters are non-overlapping in the sense that none of the source vertices gave values less than one for any of the members of the other two clusters. The parameter values were set to $c = 0.09$, $\delta = 0.1c$, and $\xi = 0.1\delta$.

The third example (Figure 4.10) represents the friendship relations among 34 members of a university karate club [326]. Due to internal tensions, the club split into two during the study period, and some of the members joined the former instructor of the club in establishing a new organization. This network is frequently referred to as an example by community discovery literature [237, 240, 322]. In the graphs of Figure 4.10, the actual partition of the club is indicated by the shape of the vertices: square vertices correspond to the club members who stayed in the original club together with its administrator, and circular vertices correspond to the members who moved to the new club.

On the left side in Figure 4.10, the vertices are colored according to their approximate Fiedler values in the cluster of the original club's administrator, and on the right, according to the club's instructor. As can be seen, the correlation between the shapes and colors of the vertices is quite good in both cases; only a few vertices in the middle are "undecided" as to which club they belong to, but this may correspond also to the actual social reality of the situation.

We wish to emphasize that the small size of our example graphs here is due to the requirements of illustration. The presented methods are *local* and hence their running time scales relative to the order of the resulting *cluster*, and does *not* depend on the order of the graph.

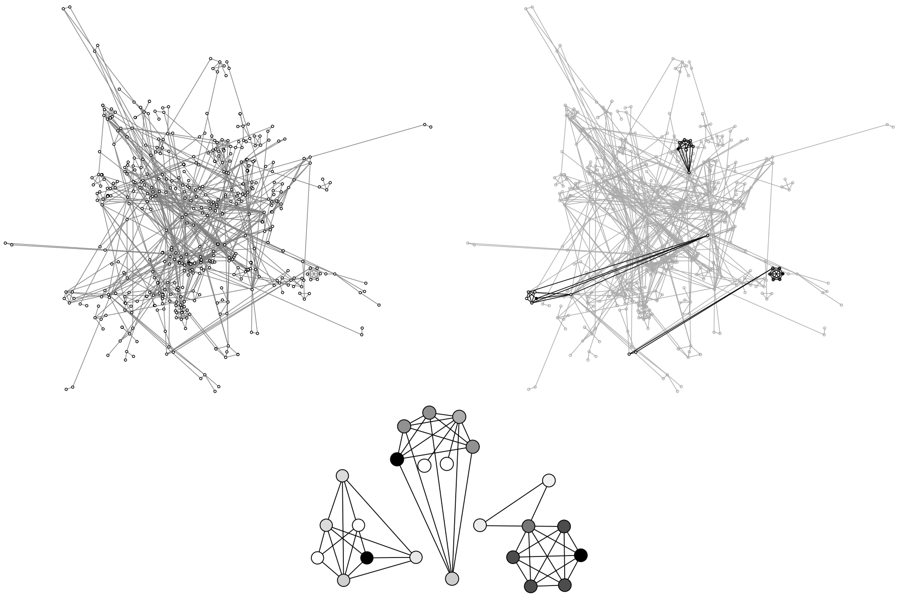


Figure 4.9: Local Fiedler clusters in a 503-vertex collaboration graph; on the bottom, a closeup view of the three clusters with distant and overlapping vertices rearranged to allow a better view of the structure of the induced subgraphs.

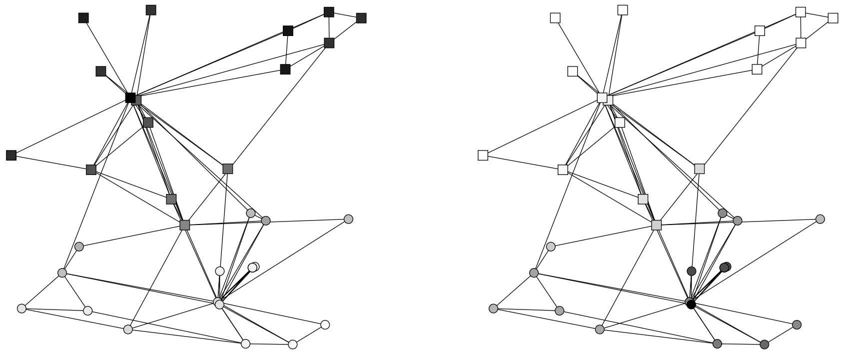


Figure 4.10: Fiedler values in a 34-vertex karate club social network.

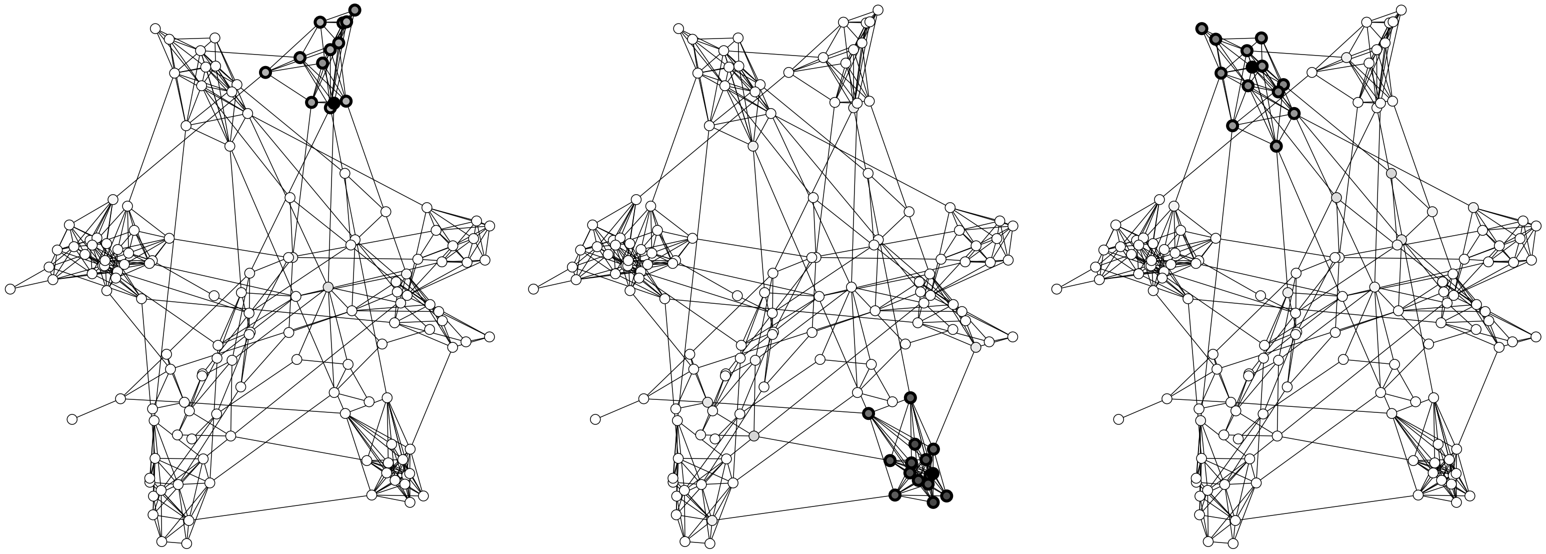


Figure 4.11: Local Fiedler clusters in a 144-vertex caveman graph.

4.5 APPLICATIONS

For example the performance of services provided on the World-Wide Web can be improved by clustering web clients in order to identify groups of clients that request similar information and reside at nearby locations with respect to each other, in which case e.g. a mirror server can be set to serve each such cluster of clients. Such a clustering has been studied by Krishnamurty and Wang [195].

Flake *et al.* [120] identify communities on the web, i.e., groups of websites with more internal links than links to "outside" websites, using a network flow approach. A similar definition for web communities is given in [48], in which also using density as a clustering measure is suggested. In document databases that record the inter-document references in addition to the document contents, link-based local clustering could be combined with content-based similarity measures when serving relevance queries where the user wishes to locate documents similar or related to a given document.

In this section we discuss three application areas for clustering, namely the employment of clusterings for routing in mobile networks, clustering of graph databases to facilitate efficient queries, and the assessment of graph generators.

4.5.1 A clustering protocol for ad hoc networks

An *ad hoc network* is formed by communication nodes residing in the same region with no or little central control in the formation and the function of the network. For an introduction to such systems, we recommend Perkins's book [258]. In ad hoc networks, there are no centrally maintained *routing tables* telling the nodes how to communicate to other nodes. Methods for constructing routing tables that can be used to locate other nodes in ad hoc networks need to be scalable and dynamic, as in many applications the nodes are mobile and hence there are frequent topology changes in the system. It is known that grouping the nodes into clusters allows for savings in the size of the routing tables while maintaining good efficiency in the choice of the communication paths [192], also reducing the number of messages that the nodes need to send in order to form and maintain the routing tables [174, 202, 292]. Proposals for and analysis of cluster-based routing in dynamic networks include [194, 292] — an introductory review is provided by Steenstrup [290].

We employ the ideas of local clustering presented earlier in this chapter to design an efficient scheme for forming clusters in a mobile ad hoc network. When clustering is introduced to an ad hoc routing system, a locally computable clustering is a necessity in order to avoid generation of further control traffic. In the ideal case, each arriving node is able to determine the appropriate cluster simply by consulting its immediate neighbors, who will not need to communicate further to determine the best cluster. The scheme works similarly to that of Lin and Gerla [209] — our contribution is in cluster construction that allows for topology changes at any step of the execution of the algorithm as it functions well under incomplete information, as will be shown in our experiments later in this section. The method resembles

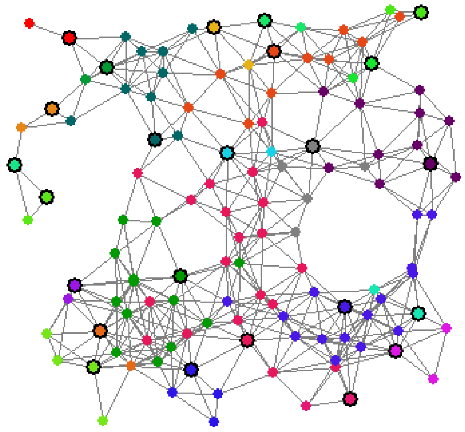


Figure 4.12: Each cluster has its own color; the nodes have been added one by one, with existing nodes updating their clusters after the newcomer selects a cluster. Cluster heads are drawn with a black border. In these examples, all nodes have a fixed communication range and they do not move.

that of Ohta *et al.* [248], with the difference that we choose between cluster candidates by optimizing a density-based fitness function.

If we are able to produce connected clusters that have high local density with only few links to the rest of the network, the routing task is simplified. For intra-cluster routing, it becomes possible to use link-state algorithms, such as OLSR [76], which require dense and relatively small networks in order to be efficient [273]. If the clusters are stable enough, this gives good performance. Inter-cluster routing, on the other hand, may well use on-demand routing protocols.

By optimizing Equation 4.30, we may cluster ad hoc networks without extra messages, since the required messages are simple enough to be piggybacked on link layer or routing messages. Based on our simulations, this produces intuitively good clusters, thereby minimizing address changes and allowing us to optimize routing traffic.

A simple protocol for clustering is the following [280]: upon arriving to a new location or waking up from sleep, a new node probes its neighborhood. All existing nodes that hear the probe, if any, send their cluster identifier together with three integers: the number of nodes in the cluster $|\mathcal{C}'|$, the internal degree $\deg_{\text{int}}(\mathcal{C}')$ of the cluster, and the external degree $\deg_{\text{ext}}(\mathcal{C}')$ of the cluster. Since these values take so little space, they could be easily embedded into existing messages.

Based on the messages the arriving node receives, it constructs a neighbor list and calculates the fitness that each of the neighboring clusters would obtain if it were to join that particular cluster. It is also able to deduce the current fitness of each cluster, and chooses to join the cluster which "gains" the most (or "suffers" the least) due to the arrival of the new member. If a node has no neighbors or no neighboring cluster accepts it, that is, it receives no answers to the probe, it starts its own cluster. The acceptance criteria may for example include a maximum cluster size or a threshold on how much the cluster fitness may decrease upon a new arrival. Such parameters to the

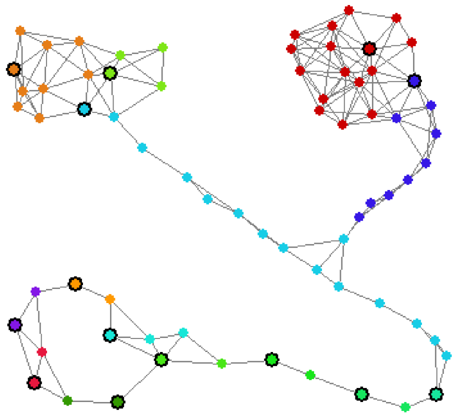


Figure 4.13: Clusters selected by a method for some anomalous network structures.

cluster formation serve to regulate the amount of routing traffic in link-state based routing schemes. Once the node has solved the cluster it wants to join to, it sends a message indicating its desire. This can be implemented as a routing-protocol message, such as an OLSR HELLO message [76].

The cluster-selection protocol described above should be repeated as the network evolves. The cluster memberships can be updated, for example, on regular intervals or upon the creation or loss of connections. Detection of a new edge causes a node to re-execute the cluster-selection protocol; a node will only change its cluster if the new cluster arrangement is (much) better than the current one. This reduces the amount of routing and address management traffic.

Clearly, if a cluster splits, each node must select a new cluster, and such information should propagate along the disconnected component. In order to determine which "half" of a separated cluster should select a new cluster identity, one node in each cluster needs to hold a "cluster head" status. A node that has no intra-cluster paths remaining to the cluster head must re-initiate the selection protocol, hereby alerting its neighbors to check whether they still have a proper path to the cluster head. If a full link-state protocol is used within a cluster, routing information allows trivial partitioning detection.

To examine what the resulting clusters look like, we have built a simulator to visualize clusterings. Figure 4.12 shows an example of a randomly generated graph and Figure 4.13 shows the clustering for some anomalous shapes. We also built an `ns-2` implementation [219] of the algorithm for more realistic experiments. Our experiments with simulation tools are promising: the clusters achieve a proper sense of locality in space and their structure corresponds well to the intuitive global clusterings of the network.

In the `ns-2` simulations, we used networks with 30 nodes in a one square-kilometer area and set the minimum cluster order to be five and the maximum to eight nodes; the simulator was very slow for larger networks. We had each node probe its neighborhood, within a range of 250 meters, on five-second intervals and the cluster heads broadcasting a status message for intra-cluster flooding also on five-second intervals. A node i executing the cluster-selection protocol would only switch from its current cluster $\mathcal{C}(i)$ to another cluster $\mathcal{C}'$ if the switch is *quality-increasing*, the sum of the fitnesses of the clusters $\mathcal{C}(i)$ and $\mathcal{C}'$ will grow as i switches from $\mathcal{C}(i)$ to $\mathcal{C}'$. Having received the cluster-offer message from a node in $\mathcal{C}'$, i has the information needed for such a computation. The minimum and maximum cluster orders were enforced such that a switch was forced to happen if $\mathcal{C}(i)$ was too small and no switch was made if $\mathcal{C}'$ was already at maximum order.

Observing the behavior of the clustering method on the simulators, it also seems feasible to approximate the fitness function using estimates of the required values $|\mathcal{C}'|$, $\deg_{\text{int}}(\mathcal{C}')$, and $\deg_{\text{ext}}(\mathcal{C}')$. Such an approach with "lazy updates" of the fitness-function inputs would allow for a more relaxed control traffic within the cluster, as not all nodes need to be immediately aware of newcomers, departing nodes, or changes in edges. The effects of outdated information can be deduced from the fitness function (Equation 4.30): a node can have an outdated value for cluster order $|\mathcal{C}'|$, for the internal degree $\deg_{\text{int}}(\mathcal{C}')$, or the external degree $\deg_{\text{ext}}(\mathcal{C}')$; the magnitude of the difference between the actual value and the assumption made at a single node depends on the rate of change in the clustering as well as the frequency with which updated information is propagated in the network. We traced a set of `ns-2` runs and computed at each time step the true values of the above measures and compared those to the "belief" of each node, calculating the absolute value of the difference. Small deviations between the observed and actual values of the fitness-function input do not drastically affect the fitness value. Therefore a practical clustering can be achieved as long as all nodes maintain an approximate "guess" for each required value, and the error of the estimate and the actual value does not diverge rapidly.

Figure 4.14 shows that the estimates do not diverge over time and hence the clustering does not suffer from outdated information; the nodes are able to function well with their limited information on the values needed to evaluate cluster fitness. However, in cluster splits, there is a risk that the original cluster will not notice the departure of some nodes. In situations where splits are frequent and the departing nodes will often become completely detached from the old cluster, not even remaining in the neighborhood, the cluster heads should send out time-stamped beacon messages containing the cluster member list that are propagated by broadcast within the respective clusters, and the member nodes respond (through a broadcast tree formed by the order in which the nodes received the beacon message from each other) by stating which of those members are currently their neighbors and how many other neighbors they have. The number and contents of these response messages informs the cluster head of the current cluster order, internal degree, and external degree. The cluster may then include these three values in the next beacon message.

Such a mechanism allows for the entire cluster to maintain a more up-to-date view on the cluster topology. The cluster head should not send out a new beacon before it receives the replies to the previous ones; the waiting time should be reset upon the arrival of a reply and the computation of the current values should only be done after a timeout occurs with no further reply arrivals. If however the cluster head receives a reply *after* the timeout, it should increase the waiting time for the next beacon round. A mechanism

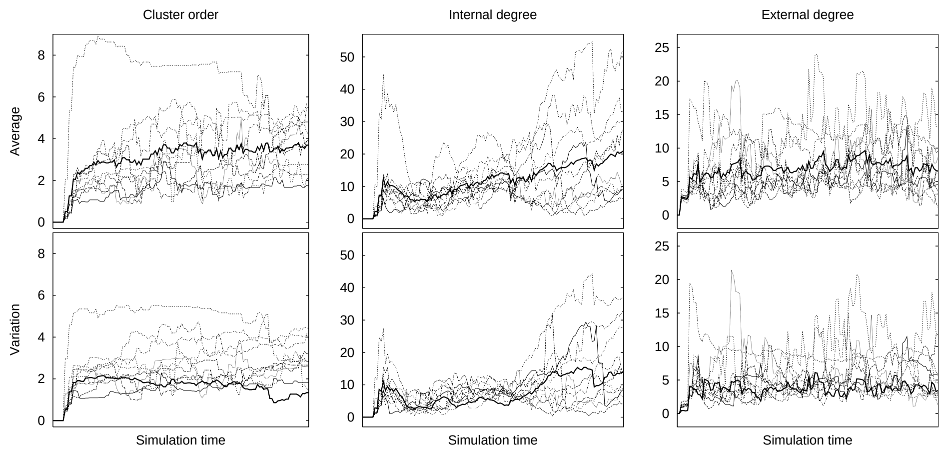


Figure 4.14: The absolute values of the average differences μ_i for $|\mathcal{C}'|$ (left), $\deg_{\text{int}}(\mathcal{C}')$ (middle), and $\deg_{\text{ext}}(\mathcal{C}')$ (right) over a set of $I = 10$ runs of $D = 250$ seconds; the average over the runs is drawn thicker (as respectively is the variation of the variations).

for reducing the time if all replies arrive quickly could also be included. Note that by adding a hop counter to the beacon messages, incremented by each forwarding node, nodes can include the value of the counter upon their first reception of the message to their replies and thus inform the cluster head of their "effective" distance from the cluster head; this information could also be used to adjust the waiting time at the cluster head.

For moderately sized clusters (at most 256 nodes) and 64-bit cluster identifiers, all of the required information can be fit into 16 bytes. This can be easily included in existing link-layer frames, IP layer address resolution or neighbor-discovery messages, or routing messages; the information could be carried in Wireless LAN beacon frames or in IPv6 neighbor-discovery messages.

We studied the quality and stability of the clusterings produced by a series of `ns-2` simulations, studying cluster density, fitness, and stability as the main indicators. We ran $I = 10$ simulations to study the *performance* of the algorithm under four different *mobility models*. The mobility models utilized were reference-point group mobility (GM) model with nodes moving in small groups, random direction (RD) model, random walk model (RW), and random way-point (RWP) model [57, 246].

In all our scenarios the nodes move with speed uniformly distributed in [0, 15] m/s after an initial period of [0, 5] seconds. In GM, each individual node moves as in RWP, but within a restricted area of 200 m^2 surrounding the group imaginary reference point, while reference points also move as in RWP, but within the whole simulation area. For RW, node change direction on one-second intervals.

We report averages and variations of the measured indicators, denoting the average over values of variable x_i with $i \in \{a, a+1, \ldots, a+b\}$ by $\text{Avg}\,[x_i]_a^{a+b}$ and employing the following measure of variation:

$$\varrho\,[x_i]_a^{a+b} = \sqrt{\frac{\sum_{i=0}^{k}\left(\text{Avg}\,[x_i]_a^{a+b} - x_i\right)^2}{b+1}}. \tag{4.41}$$

Table 4.3: Measures of graph (Equation 4.42) and cluster stability (Equation 4.43) for the four mobility models (MM) used; the values are averages over the set of $I = 10$ experiments, each with duration $D = 600$ seconds.

MM	$\tilde{\mathcal{B}}/D$	$\tilde{\mathcal{B}}_{\text{int}}/D$	$\tilde{\mathcal{E}}/D$	$\tilde{\mathcal{E}}_{\text{int}}/D$	$\tilde{\mathcal{T}}/D$	$\tilde{\mathcal{T}}_{\text{int}}/D$	$\tilde{\mathcal{Q}}$	$\tilde{\mathcal{F}}$	$\tilde{\mathcal{S}}$	$\tilde{\mathcal{T}} \cdot \tilde{\mathcal{S}}$
GM	1.53	0.25	1.55	0.21	3.08	0.46	0.03	0.01	0.04	0.13
RD	1.04	0.43	1.06	0.20	2.10	0.46	0.10	0.08	0.18	0.39
RW	1.13	0.47	1.15	0.42	2.28	0.90	0.04	0.02	0.07	0.15
RWP	1.50	0.59	1.51	0.26	3.01	0.86	0.09	0.07	0.16	0.48

The metrics measured over a period of $D = 600$ seconds were the following, with k denoting the cluster count (at a certain time step) and a measurement was taken for time steps $t \in \{0, 2, 4, \ldots, 600\}$: the cluster order $\text{Avg}\left[|\mathcal{C}i|\right]_1^k$, the cluster fitness $\text{Avg}\left[\mathcal{F}(\mathcal{C}i)\right]_1^k$, and the local density of the clusters versus the density of the graph $\left(\text{Avg}\left[\delta(\mathcal{C}i, \mathcal{C})\right]_1^k - \delta(G)\right)$. For the density difference, the range is $[-1\,, 1]$ and a positive value indicates that dense subgraphs have been selected as clusters; if the value is close to one, almost all links present in the graph are internal to some cluster.

The results are shown in Figure 4.15 with separate plots for each mobility model and each metric. All mobility models produced clusters with much higher local density than the density of the entire graph. Unexpectedly, the group mobility model produced large clusters with very high density, whereas random way-point and random direction produced small, but dense clusters. Similarly, clusters in the group mobility scenario had consistently much better fitness than in other mobility models.

We also studied the *stability* of the graph and cluster topologies, recording the total amount of link breakages $\mathcal{B}_i$ and new link establishments $\mathcal{E}_i$ and the average topology change rate $\mathcal{T}/D$ by considering the graph variations occurred per second in experiment i, $i \in \{1, 2, \ldots, I\}$:

$$\tilde{\mathcal{B}} = \text{Avg}\left[\mathcal{B}_i\right]_1^I, \; \tilde{\mathcal{E}} = \text{Avg}\left[\mathcal{E}_i\right]_1^I, \; \tilde{\mathcal{T}} = \tilde{\mathcal{B}} + \tilde{\mathcal{E}}. \tag{4.42}$$

We additionally recorded the number of topology changes that were *internal* to clusters, denoting these by $\mathcal{B}_{\text{int}}$, $\mathcal{E}_{\text{int}}$, and $\mathcal{T}_{\text{int}}$, respectively.

The cluster stability was measured by the number of cluster changes, distinguishing between two categories: $\mathcal{Q}_i$ is the number of quality-increasing cluster switches and $\mathcal{F}_i$ is the number of switches due to a cluster split:

$$\tilde{\mathcal{Q}} = \text{Avg}\left[\frac{\mathcal{Q}_i}{\mathcal{B}_i + \mathcal{E}_i}\right]_1^I, \; \tilde{\mathcal{F}} = \text{Avg}\left[\frac{\mathcal{F}_i}{\mathcal{B}_i + \mathcal{E}_i}\right]_1^I. \tag{4.43}$$

We denote $\tilde{\mathcal{S}} = \tilde{\mathcal{Q}} + \tilde{\mathcal{F}}$; note that as $\tilde{\mathcal{T}}$ is the average number of topology changes, $\tilde{\mathcal{S}} \cdot \tilde{\mathcal{T}}$ is the average amount of cluster changes in a single simulation run.

The results in Table 4.3 show that group mobility model and random walk model have the most stable clustering structure of the four models. The reasons, though, differ. Random walk creates mainly local movements, which means that the overall topology of the graph will tend to stay the same with small variations. It has as low rate of topology changes as random direction, but causes much less changes in the clustering structure. This is due to the local movements of nodes in random walk vs. global movements of nodes in random direction. Group mobility model creates global movements, but with certain groups of nodes staying close to each other. This causes a high rate of changes in the topology, but low rate of changes in the clustering. The random-direction model also produces global movements.

We experienced very few clusters splits and changes in general. Overall, the rate of changes in clustering is small. Group mobility and random walk cause changes in clustering in 4% and 6% of cases when topology changes and random direction and random way-point models in 16% and 18% respectively.

The simulations show that the clustering algorithm is capable of capturing the structure that may exist in the movements of nodes. This is especially marked by the observation that group mobility model had the highest rate of topology change, but still had least changes in clustering both per topology change and per unit of time. The algorithm also managed to create clusters with high local density, allowing us to partition the network into smaller subnetworks which are easily managed by proactive routing algorithms such as OLSR [76] that are designed especially for dense networks with small diameter.

We plan to study further how the proposed clustering algorithm could be used to further optimize routing and addressing management. On top of a base-layer clustering, the clusters may be clustered further to form a hierarchy of clusterings on different levels. Informally speaking, each cluster of the base-layer clustering is abstracted into a node and edges appear between adjacent clusters — essentially the cluster-formation protocol can be used, relying on routing the higher-level cluster requests to the cluster heads. Such a layering requires the cluster heads to perform additional tasks, as the higher-level clustering takes place among the cluster heads, some of which become also cluster heads on the meta-level, and so forth. We leave the details of such hierarchical clustering for further work.

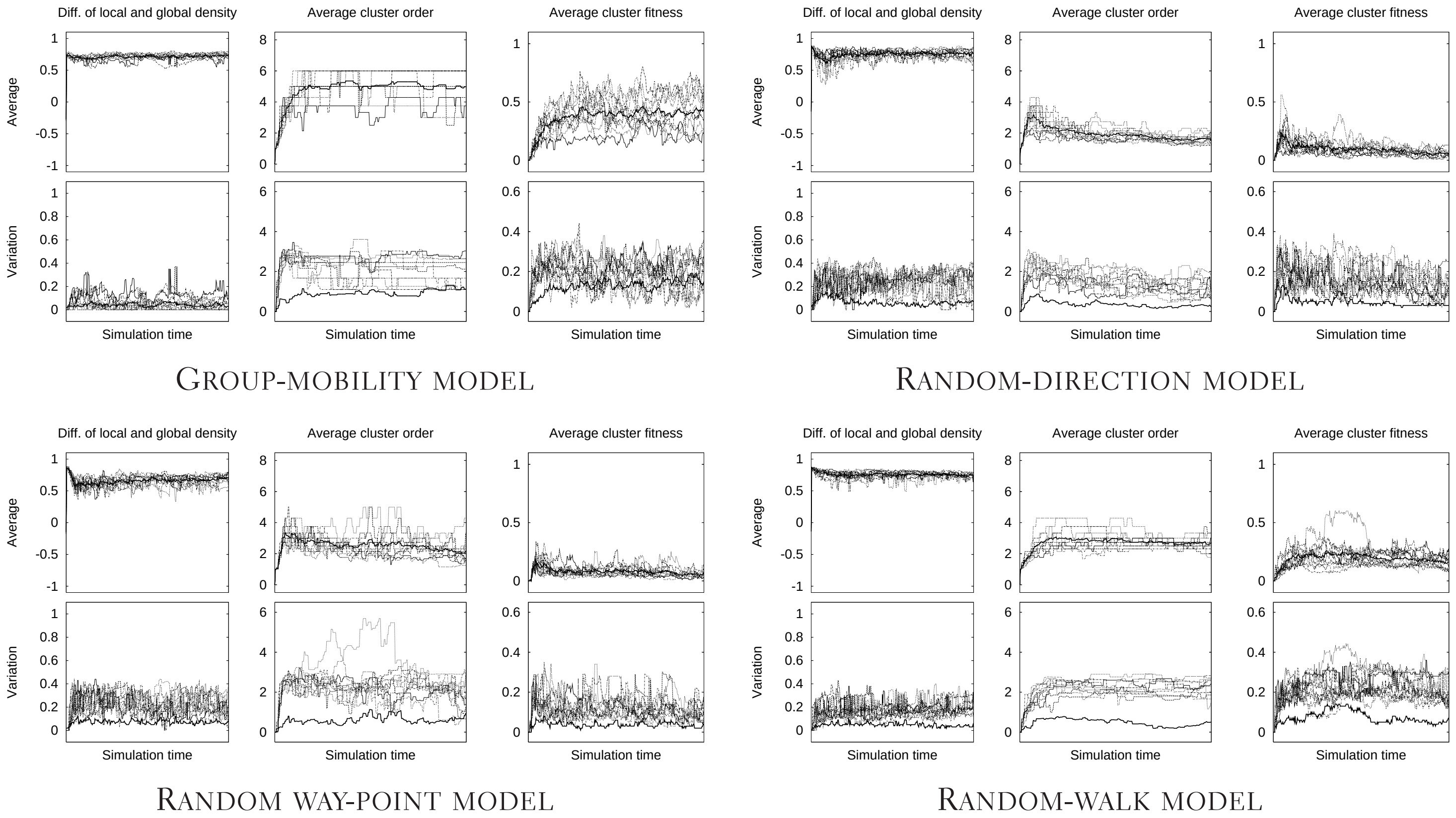


Figure 4.15: Grouped by the mobility model, averaged over the set of clusters for each time step in $\{0, 2, 4, \ldots, 600\}$ for each of $I = 10$ runs, the difference of the average local density and the global density of the graph, the cluster order, and the cluster fitness (average over the runs drawn thick).

4.5.2 Storing massive graphs for improved searchability

Current databases are well-suited to store relational data and to serve queries asking for all entries that fulfill given criteria. When storing graph data in such a database, we may for example build one relational table out of the vertices and their properties, and another table of the edge set. Such storage formats however do not support well *navigational queries* such as path searches, although finding paths or common neighbors for a given vertex pair are frequent subtasks of graph algorithms. Therefore it is of interest to develop database-like storage formats for massive graphs that support these kinds of queries with ease.

Knowing that the average path length in natural graphs is often small, it should be possible to store a graph in such a way that a path query will not need to access irrelevant parts of the graph. Also, as natural graphs often have a clear cluster structure, common-neighbor queries and queries on connectivity and relevance in general could be efficiently served storing each cluster together. In this section we discuss methods that take such structural observations into account in deciding how to store a large graph in external memory while facilitating efficient queries.

We separate two fundamental query types for graphs: path queries [56] and cluster queries in the sense of nearest-neighbor queries that are relevant for spatial data [272]. Path queries contain lookups of shortest or longest paths and other connectivity properties. Cluster queries contain neighborhood retrieval and "relevance" queries. An ideal graph database would support both of these query types with minimal page accesses in external memory, for example by storing precomputed clusters on nearby pages (if they do not fit on single pages) and using preprocessing of vertex distances to avoid retrieval of irrelevant pages during path construction.

The database should also be flexible for updates such as vertex insertions and deletions; maintaining the preprocessing status could be implemented in a relaxed fashion, as has been proposed for the balancing of tree-based index structures [201]. Graphs with different structural properties would benefit from different storage methods: a scale-free graph that has low clustering is efficiently accessed by hub-based grouping, whereas a graph with a more even degree distribution and strong clustering benefits from cluster-based grouping. For a highly nonuniform graph, different parts of the graph could be stored according to different heuristics for optimal performance.

Agrawal and Jagadish [7] propose storing massive graphs by decomposing them into *domains* with low diameter. The goal is to provide a data structure that facilitates efficient solving of *extremal path problems*, consisting of finding the shortest or longest paths between a given pair of vertices. The approach they choose is to precompute some of the distances over the graph at storage time and then employ this information to prune the path search at run time. This works particularly well with nonuniform graphs that allow a natural clustering, i.e., graphs that are formed of dense subgraphs that are only sparsely connected to each other. A similar idea was independently proposed by Wu *et al.* [320].

We assume each edge to have a real-valued *weight* $\omega(\{v, w\}) \in [0, 1]$

associated to it.[5] We set $\omega(\{v, v\}) = 0$ for all $v \in V$. The input graph $G = (V, E)$ to be stored is preprocessed as follows: the vertex set V is divided into subsets $S_1, \ldots, S_k$. Each subset S_i is called a *domain* and is assigned a *center* $\mathrm{cnt}(S_i) = v \in S_i$. The *radius* of a domain S with center v is defined as

$$\mathrm{rad}_v(S) = \max_{w \in S} \{\mathrm{dist}(v, w)\}, \tag{4.44}$$

where $\mathrm{dist}(v, w)$ is the total weight of the edges on the shortest path connecting v to w.

The construction of Agrawal and Jagadish [7] generalizes also to directed graphs, but we restrict our discussion to undirected graphs. The $\frac{1}{2}k(k-1)$ distances between the k domain centers $\mathrm{cnt}(S_1), \ldots, \mathrm{cnt}(S_k)$ are stored in a table that is used to obtain bounds on distances of arbitrary vertices when their domains are known.

The division into domains is based on heuristics, with three alternatives presented along the formulation of the algorithm [7]; the heuristics are all somewhat similar, with the first two ones fitted for weighted graphs, and the third (shown in Table 4.4) using *hop counts* (i.e., path lengths) to build domains around randomly chosen centers. As the algorithms for clustering undirected graphs have developed significantly since the proposal of Agrawal and Jagadish, we suggest as an alternative using an existing top-down clustering algorithm that has efficient implementations. If the initial, global domain assignment is done only once, it is not critical if this step takes a noticeable amount of computation time. Hence we suggest using the speed-up version [239] of the edge-betweenness algorithm originally proposed by Newman and Girvan [241].

For graphs that are too large for global computation, we resort to a variation of the algorithm of Table 4.4 based on the local clustering algorithm of Section 4.2.4, picking a random seed vertex v as in the heuristic, and instead of computing the domain based on hop counts and a threshold, computing $\mathcal{C}(v)$ with the algorithm of Table 4.1 (on page 60), and repeating until no vertices remain unassigned [278]. If desired, a minimum cluster order could be set such that no cluster smaller than that will be accepted, and vertices whose clusters are smaller than that are joined to a neighboring cluster (as in the heuristic), choosing among the nearest clusters by optimizing the local cluster-fitness function of Equation 4.30, for example.

Also other clustering algorithms could be used; what interests us is that the resulting global clustering constructed is of high quality even if not quite optimal with respect to the clustering quality measure of Equation 4.38 on page 61. Each cluster of the graph becomes a domain in the storage construction; what remains to be done is the selection of the center for each domain.

Agrawal and Jagadish [7] aim to minimize the distance between each vertex and the center of the corresponding domain. The goal is to construct low-diameter domains, whereas many natural networks have an altogether small diameter due to the small-world property. Hence simply thresholding on the diameter will not be useful. In the original algorithm the heuristics

[5] If the weight function of an application uses another interval of the real axis, a simple scaling to the unit interval can be done if an upper bound on the edge weights is known.

Table 4.4: The hop-based heuristic for domain creation of Agrawal and Jagadish [7] (named "Heuristic 3" in the original article). It takes as input a graph $G = (V, E)$, an integer d, a rational number $s \in [\frac{1}{n}, n]$, and an integer $h_{\max} \in [1, \operatorname{diam}(G)]$.

For each $v \in V$,
  $\mathcal{C}(v) :=$ *undefined*. `// Initially there are no domains`
$i := 0$.
While $(i < d)$, `// Iterate to obtain domain centers`
  $C := \{v \mid v \in V, \mathcal{C}(v) = \textit{undefined}\}$.
  Select $v \in C$ uniformly at random.
  $\operatorname{cnt}(\mathcal{C}_i) := v$.
  $\mathcal{C}(v) := \mathcal{C}_i$.
  $j := 1$.
  While $\left(\left(\frac{|\mathcal{C}_i|}{n} < s\right) \wedge (j \leq h_{\max})\right)$,
    $C := \{w \mid w \in V, \mathcal{C}(w) = \textit{undefined}, \operatorname{dist}(v, w) = j\}$
    Select $w \in C$ uniformly at random.
    $\mathcal{C}(w) := \mathcal{C}_i$.
    If $(C = \{w\})$,
      then $j := j + 1$.
  $i := i + 1$.
$C = \{v \mid v \in V, \mathcal{C}(v) = \textit{undefined}\}$.
For each $v \in C$,
  Select w s.t. $\operatorname{dist}(v, w)$ is minimal **and** $(\mathcal{C}(w) \neq \textit{undefined})$.
$\mathcal{C}(v) := \mathcal{C}(w)$.

that assign the domains also address the question of center selection, but as clustering algorithms simply deal with partitioning the vertex set, we select the center of a domain after the domain has been constructed, optimizing essentially the same criteria as Agrawal and Jagadish. We fix the *center* $\operatorname{cnt}(\mathcal{C}_i)$ of the domain $\mathcal{C}_i$ to be a vertex $v \in \mathcal{C}_i$ that has minimal total distance within the domain, resolving ties by vertex identifiers. This minimizing value is called the *fitness* of the center $\mathcal{F}(\operatorname{cnt}(\mathcal{C}_i))$ and is assumed to be stored in the domain graph structure,

$$\mathcal{F}(\operatorname{cnt}(\mathcal{C}_i)) = \min_{v \in \mathcal{C}_i} \left\{ \sum_{w \in \mathcal{C}_i} \operatorname{dist}(v, w) \right\}. \tag{4.45}$$

We begin by discussing the *addition* of a new vertex v into a graph $G = (V, E)$ that has currently domains $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k$. Assume that along with vertex v, a list of vertices $\Gamma(v) \subseteq V$ is given and that all edges $\{v, w\}$ such that $w \in \Gamma(v)$ will also be added to E. Denote the set of these edges by $F_v = \big\{\{v, w\} \mid w \in \Gamma(v)\big\}$. If the set $\Gamma(v)$ is empty, we simply create a new domain $\mathcal{C}_{k+1} = \{v\}$ and assign $\operatorname{cnt}(\mathcal{C}_{k+1}) = v$.

The *candidate domains* for including v are those domains $\mathcal{C}_i$ that contain at least one neighbor of v. The weight of their connectivity to v is defined as

$$\omega\left(v, \mathcal{C}_i\right) = \sum_{\{v,w\} \in F_v} \omega\left(\{v, w\}\right). \tag{4.46}$$

For each domain, we define a *fitness* using a unit-interval fitness measure $\mathcal{F}\left(\mathcal{C}_i\right) \in [0,\ 1]$. For unweighted graphs, we use Equation 4.30 on page 57 directly; it is straightforward to see that $\mathcal{F}\left(\mathcal{C}_i \cup \{v\}\right)$ is obtained as

$$\mathcal{F}\left(\mathcal{C}_i \cup \{v\}\right) = \frac{2 + \left(\mathcal{E}_{ii} + \omega\left(v, \mathcal{C}_i\right)\right)^2}{\left(\left|\mathcal{C}_i\right| + 1\right)\left|\mathcal{C}_i\right| \sum_{j=1}^{k} \left(\mathcal{E}_{ij} + \omega\left(v, \mathcal{C}_j\right)\right)}, \tag{4.47}$$

using $\mathcal{E}_{ij}$ to denote the total weight of edges connecting clusters i and j (Equation 4.39). Generalizations to weighted graphs are simple, given that the weights are bounded to some interval that can then be scaled to the unit interval such that the longest edges have the smallest weight; the clustering algorithm will then avoid "breaking" short edges when forming the clusters more than it will avoid separating vertices that are farther apart. For simplicity of presentation, we continue discussing the unweighted case.

For each domain candidate, we calculate the change in cluster fitness should v be added into that domain,

$$\Delta_v\left(\mathcal{C}_i\right) = \mathcal{F}\left(\mathcal{C}_i \cup \{v\}\right) - \mathcal{F}\left(\mathcal{C}_i\right). \tag{4.48}$$

If for some $\mathcal{C}_i$, $\Delta_v\left(\mathcal{C}_i\right) \geq 0$, we insert the new vertex v into the domain $\mathcal{C}_i$. If several domains qualify, we choose the one that has the highest value of $\Delta_v\left(\mathcal{C}_i\right)$. If all such values are negative, there are several options. If we are not worried about the number of domains growing, we may create a new domain for the newly arrived vertex and assign $\operatorname{cnt}\left(\mathcal{C}_{k+1}\right) = v$. The growth in domain count increases the size of the domain-to-domain distance table that needs to be maintained in the domain structure [7], so it may be of interest to allow domain quality to decrease using a predetermined threshold $\xi_{\text{fit}} \in [0,\ 1]$: if it holds that

$$\max_{i=1,\ldots,k} \left\{\Delta_v\left(\mathcal{C}_i\right)\right\} \geq -\xi_{\text{fit}}, \tag{4.49}$$

add v to the domain that has the maximum $\Delta_v\left(\mathcal{C}_i\right)$-value, otherwise create a new domain for v. Also, the additions could be carried out to the domain of the least-decreasing fitness, maintaining a counter on how many fitness-decreasing insertions have been made, and performing a complete reclustering of the graph whenever a critical update count is exceeded (as in relaxed balancing of index trees, during a low-access period on the database [201]).

If v was inserted into an existing domain $\mathcal{C}_i$, set

$$\operatorname{dist}\left(v, \operatorname{cnt}\left(\mathcal{C}_i\right)\right) = \min_{w \in \Gamma(v)} \left\{\omega\left(\{v, w\}\right) + \operatorname{dist}\left(w, \operatorname{cnt}\left(\mathcal{C}_i\right)\right)\right\}. \tag{4.50}$$

It may not be desirable to recalculate the optimal center for $\mathcal{C}_i$ every time an insertion occurs, as this would require the update of the global center-to-center distance table of the domain graph storage structure [7]. We suggest setting a threshold $\xi_{\text{dist}} > 1$ such that the center selection is only repeated upon the addition of a vertex v when

$$\operatorname{dist}(v, \operatorname{cnt}(\mathcal{C}_i)) > \xi_{\text{dist}} \cdot \mathcal{F}(\operatorname{cnt}(\mathcal{C}_i)). \tag{4.51}$$

If a vertex $v \in \mathcal{C}_i$ is *removed* from $G = (V, E)$, the procedure is the following. If $\mathcal{C}_i \setminus \{v\} = \emptyset$, the domain $\mathcal{C}_i$ is removed from the structure. If $\mathcal{C}_i$ remains non-empty but connected, i.e., there exists a path between any vertex pair within $\mathcal{C}_i$, no large changes are required on domain level, unless $v = \operatorname{cnt}(\mathcal{C}_i)$, in which case the center selection needs to be performed. In full-scale implementations it would be useful to recalculate the center status after several vertices have been removed from the same domain, but we omit this for simplicity as it would require introducing yet another threshold value or some other decision criterion.

If the removal of v splits $\mathcal{C}_i$ into connected components, but $v \neq \operatorname{cnt}(\mathcal{C}_i)$, the connected component containing $\operatorname{cnt}(\mathcal{C}_i)$ becomes $\mathcal{C}_i$ and the other connected components form new domains $\mathcal{C}_{k+1}, \mathcal{C}_{k+2}, \ldots$ and have centers selected respectively. If v is the center, one of the resulting components is chosen to be $\mathcal{C}_i$ and the others form new domains. If increase in domain count is undesirable, another option is to remove $\mathcal{C}_i$ altogether from the graph and add the vertices in $\mathcal{C}_i \setminus \{v\}$ back into the graph one by one, in the order of decreasing number of neighbors in $V \setminus \mathcal{C}_i$, preserving the neighborhoods of the original graph in neighbor lists given as parameters to the addition process described above.

When a single edge $\{v, w\}$ is *added*, no major changes are needed. If the vertices v and w belong to the same domain $\mathcal{C}_i$, we do nothing, although some criterion for deciding whether to recalculate $\operatorname{cnt}(\mathcal{C}_i)$ would be useful in an industrial-strength implementation. If on the other hand $v \in \mathcal{C}_i$ and $w \in \mathcal{C}_j$, $i \neq j$, a decision criterion for whether to reselect the domains for the vertices (equivalently as upon vertex addition) is needed. For example, the protocol could be executed only on the endpoint that currently has lower domain quality, resolving ties by vertex identifiers.

For the *removal* of an edge $\{v, w\}$, nothing at all should be done if the two vertices are in different domains. If $\{v, w\} \subseteq \mathcal{C}_i$, it needs to be checked whether its removal will disconnect $\mathcal{C}_i$ into two components. If so, the component that contains $\operatorname{cnt}(\mathcal{C}_i)$ becomes $\mathcal{C}_i$ (with center-recalculation if desired), and the remainder becomes $\mathcal{C}_{k+1}$ and calculates its center.

Due to the unfortunate lack of benchmark datasets for graph clustering problems, we again resort to generalized caveman graphs to evaluate how the algorithm functions on a graph of order 1,533 that has a clear cluster structure of 30 caves and diameter three.

We computed the initial domains and centers with a hop-based heuristic of Agrawal and Jagadish [7] as well as the centers using the caves as domains (as a good clustering algorithm will identify each cave as a cluster — we use the locally computed clusters used in Section 4.4.3). We chose the hop-based heuristic as the caveman graph is unweighted and hence all distances are hop-distances in any case. The hop-based heuristic of Agrawal takes three parameters: the number of domains d, a "share" $s \approx \frac{n}{d}$, and a distance threshold $h_{\max}$. The center election is combined in the domain creation; the algorithm was given in Table 4.4 on page 81.

Table 4.5: The values of the parameters d (the domain count) and s (the filling threshold) of the heuristic algorithm of Table 4.4 used in the experiments. The bottom row shows $\frac{n}{d}$, which Agrawal and Jagadish [7] used as a starting point in adjusting s; we rounded the rational values to the closest integers.

d	5	10	15	20	25	30	35	40	45	50	Set
s	300	150	100	80	70	60	45	40	35	30	A
s	200	120	90	70	55	50	40	30	25	20	B
s	150	100	80	60	50	40	30	20	15	10	C
n/d	306	153	102	77	61	51	44	38	34	30	

As the number of caves is 30, and the caves are used directly as domains in our version, we use for the heuristic of Table 4.4 several "near-by" values, including 30, and some further-away values to examine whether the heuristic algorithm obtains better performance with a number of domains different from the number of the caves. The selection of the s-parameter is more difficult; we know that the smallest caves have only a dozen vertices, whereas the largest ones are closer to a hundred (one having 94 vertices), but there are no means to account for such nonuniformity in the heuristic construction of Agrawal and Jagadish [7]. For each value of d, we chose three different s values smaller than or close to $\frac{n}{d}$; the hope was that with small values, many of the caves could form their own domains already in the center-assignment phase. We fixed the distance threshold to two, as one is limiting and three is already the diameter of the test graph. The parameter sets are shown in Table 4.5; note that successful parameter selection requires the user to have some approximate knowledge of global properties such as the diameter and cluster-structure of the input graph.

We then calculated the distribution of estimate errors over the complete set of vertex-to-vertex distances using these domain structures, using the basic upper-bound formula of Agrawal and Jagadish [7], which simply states that going to w from v cannot require a path longer than the one that passes through the centers of the respective domains:

$$\begin{aligned} \operatorname{dist}(v,w) \leq\ & \operatorname{dist}(v, \operatorname{cnt}(\mathcal{C}(v))) + \\ & \operatorname{dist}(\operatorname{cnt}(\mathcal{C}(v)), \operatorname{cnt}(\mathcal{C}(w))) + \\ & \operatorname{dist}(\operatorname{cnt}(\mathcal{C}(w)), w). \end{aligned} \tag{4.52}$$

Remember that the storage algorithm stores all intra-domain instances and all center-to-center distances, so these values are known and have zero error; we leave them out of the average to eliminate the effect of the size of these "routing tables" in the averages. We compute each unknown distance (only in one direction, as the graph is undirected) by Dijkstra's algorithm [87] and then compute the corresponding upper bound with Equation 4.52, take the difference of the two, and average over all computed distances.

The results of the estimate-error experiment are shown in Figure 4.16. The cluster-based approach achieves a small error and the decision on the

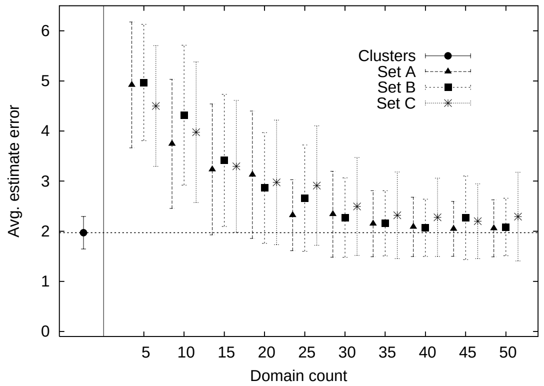


Figure 4.16: The estimate-error distributions of the domain-construction heuristics with different d/s-pairs (as listed in Table 4.5) and using clusters as domains for a caveman graph of order 1,533.

number of domains does not have to be made beforehand, but is derived from the graph structure by the clustering algorithm. It is possible to achieve similar estimate errors with the heuristic of Agrawal and Jagadish [7], but this requires adjusting the parameters with some *a priori* knowledge or assumptions on the graph structure. It is also noteworthy that as the domains become smaller, the domain centers tend to be closer to the members of the domain, and hence the increase in domain count will inevitably reduce the estimate error, but with the cost of having to manage more domain tables and a larger center-to-center table. Hence, with respect to this quality measure, the cluster-based approach works better, especially considering the ease of use.

In addition we made random modifications to the graph and studied the evolution of the estimate error. The operations were vertex splits and merges, as we wanted to mimic a more natural evolution of the graph than mere random removals and addition of elements. In each split, a vertex v is selected uniformly at random and a new vertex v' is introduced such that with probability $p = 0.75$ for each $w \in \Gamma(v)$, the edge $\{v', w\}$ is included in the graph. We always link v and v' to ensure that the graph remains connected.

In a merge operation, two distinct vertices v and w are chosen at random and combined into a vertex v' such that with probability p for each $u \in (\Gamma(v) \cup \Gamma(w))$, the edge $\{v', u\}$ is included in the graph, whereas v and w along with all of the incident edges are removed. In order to keep the graph connected for as long as possible, we required that the result of a merge always links to at least one vertex in $\Gamma(v) \cup \Gamma(w)$. Also, as the merge may cause some edges to disappear, we monitor the graph and restart the experiment should the graph become disconnected at any point.

In order to slowly increase the density of the graph, a trend which has been observed to be a natural evolution for graphs [204, 221], we also added an edge from the newly created vertex (the result of a split or a merge) to any other vertex uniformly at random with probability 0.01, hence slowly breaking the cave structure of the graph.

After a split, the new vertex v' chooses its domain. For the cluster-based algorithm, an arriving vertex always chooses a domain among the domains

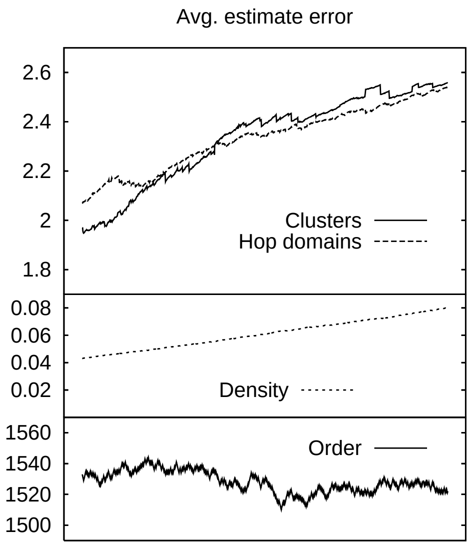


Figure 4.17: The evolution of the average and standard deviation of the estimate error for a caveman graph during 1,000 modification operations. The density and order of the graph after each modification step are also plotted to reveal the type of modifications that occurred; the start order is 1,533 and the initial size 50,597.

of its neighbors maximizing the increase (or minimizing the decrease) of cluster fitness over the set of possible domains; the domain centers are *not* recomputed. In the hop-based heuristic, the choice is made as in the final part of the algorithm of Table 4.4, joining the domain of a random neighbor. In a merge, if neither vertex was a center, the newly created vertex behaves as if it was an arriving vertex. For each domain center taking part in a merge, a center is recalculated after the removal of the current center by minimizing the total distance in the domain; this is done *before* the new arrival proceeds with the domain selection procedure.

Figure 4.17 shows the behavior of the average estimate error up to 1,000 modifications; at each step either a split or a merge was carried out with equal probabilities. The cluster-based experiment used the caves as the initial domains, whereas for the hop-based heuristic, we used a parameter combination that has a lower average estimate error than the setups with less domains but is not inferior to those with more domains in Figure 4.16, namely $d = 40$ and $s = 30$ (fixing $h_{\max} = 2$ as before). Both of the approaches suffer from the loss of structure in the graph, but neither one clearly dominates the other.

For future work, it would be of interest to use a real-world weighted graph to compare the performance of different base clusterings and different fitness functions for the dynamic modification of the clusters upon topology changes. Another issue is that not all natural graphs are composed of clusters. For example in a tree-like scale-free graph with little or no clustering, good domains are formed by identifying the hubs rather than by clustering.

In such a graph, one could run a regular random walk for $k \gg n$ steps and then choose the highest-degree vertex visited to be a domain center and include all of the neighbors to the domain. The goal is to identify hubs and use them as domain centers, iterating such a selection procedure until no vertex with degree above a predefined threshold is found over a few random-walk trials, after which the remaining vertices would join the closest domain. Hence such a graph storage structure would first need to "probe" the graph to discover what structural properties it has, and then choose a domain-formation strategy based on the observations. For highly nonuniform graphs, different parts of the graph could be processed with different strategies. For example, one could first run a random walk using the endpoint as a seed vertex for local clustering, and in the presence of a high-quality cluster, use the cluster as a domain. If no high-quality cluster is found for the seed vertex, one would instead resort to the hop-strategy for domain formation,

4.5.3 Assessment of web-like graph generators

During the recent years, analysis and modeling of the World-Wide Web has received growing interest in several disciplines. One starting point for the modeling work was a paper by Kleinberg *et al.* [190] in 1999, in which the authors construct a graph to mimic the Web Graph. Their motivation is improving search performance on the WWW as well as providing more accurate topic-classification algorithms; they foresee a "growing commercial interest" in web modeling, which is still a major driving force in network research [190].

Broder *et al.* [48] were the first to map the structure of the Web graph. They confirmed that the degree distributions follow power laws, but the main contribution is the study of the connected components, both directed and undirected, of the Web graph. The largest strongly connected component (SCC) of the graph — i.e., a subgraph in which any page can be reached by hyperlinks from any other page — they call the *central core* of the graph. The next two major subgraphs are called IN and OUT. There exists a sequence of hyperlinks connecting each page in IN to the central core but not vice versa, and respectively OUT contains those pages that can be reached from the central core but do not have hyperlinks pointing back at the SCC. The rest of the Web graph Broder *et al.* call the *tendrils* of the World-Wide Web; these consist of pages that cannot reach the central core or vice versa.

It is known that the Web Graph has many small cuts [32], and hence can be expected to allow meaningful clusterings. The Chilean Web Graph (CWG) is a coherent subset of the Web Graph, discussed in Section 4.4.1. We have performed a global clustering [305] on the CWG by local optimization of Equation 4.30. Comparing the obtained cluster distribution to that of two models of the Web Graph allows for qualitative comparison of different generation models proposed for mimicking the Web Graph — a good generation model should produce a cluster distribution similar to that of the real Web Graph. Also the number of bipartite cliques in graphs has been used for theoretical model assessment for web-like graphs [198].

Most of the components of the CWG are tiny; in addition to 6,904 single-vertex components, there is only one 2-vertex component in addition to the

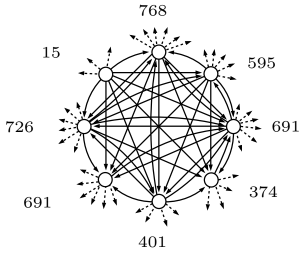


Figure 4.18: An example cluster found in one of the ten stochastic clusterings of G with the number of outside links shown. Five of the websites are `*.gotolatin.com` domains, which explains their interconnectivity.

giant component G of order 32,148. Hence G, in which each vertex is connected to at least one other included vertex, clearly dominates the CWG. Our clustering analysis was conducted on G.

In a web graph, such as the CWG, an intuitive cluster is a set of websites (or webpages, depending on the granularity of the graph construction) that are densely connected by hyperlinks, but have relatively few links to other parts of the web. We obtained flat and partitional stochastic global clusterings by iteratively optimizing the product of local and relative densities (Equation 4.30 on page 57) *without* a seed vertex, i.e., letting the local search move freely towards better solutions, and then removing the best cluster found, iterating until the graph became empty.

The cluster order was limited from above to 50 for computational ease. The initial clusters used to start the stochastic search contained up to 10 random neighbors of a random start vertex. At each step, one vertex was removed from or added to the cluster candidate, maintaining connectivity. The algorithm was repeated 30 times for each initial cluster, taking 250 modification steps per each iteration.

The distribution of cluster sizes we obtained is quite stable; the general shape and position of the distribution do not change over independent runs when varying the repetition counts and the order cutoff. For comparison, we generated graphs of the same order and similar density as G by two web-like graph models.

The first model studied was the simple evolutionary web-like generation model [205] that combines a preferential-attachment process with a non-preferential one. The model builds on the foundation of Simon's early model for generation of scale-free distributions [285], considering the process as a *urn transfer model*, and aims to match the data measurements reported by Broder *et al.* [48]. Assume initially that there is a countable number of urns where balls can be placed. The urns are labeled with the integers $i = 1, 2, 3, \ldots$ and each ball in urn $\mathcal{U}_i$ has exactly i pins attached to it. The process will be discrete-time and has two parameters: $\alpha > -1$ and $p \in (0, 1)$. Initially at time $t = 1$, urn $\mathcal{U}_1$ contains one ball and the other urns are empty.

Denote by $F_i(t)$ the number of balls in urn $\mathcal{U}_i$ at time t. At time $n \geq 1$, a new ball with one pin is added to urn $\mathcal{U}_1$ with probability

$$p_{t+1} = 1 - \frac{(1-p)\displaystyle\sum_{i=1}^{t}(i+\alpha)F_i(t)}{k(1+\alpha p)+\alpha(1-p)}, \tag{4.53}$$

if $p_{t+1} \in [0, 1]$. If the event does not occur or the value is not in the unit interval, one ball from $\mathcal{U}_i$ is transferred to urn $\mathcal{U}_{i+1}$ with an additional pin — the urn $\mathcal{U}_i$ is selected randomly with probability that depends on the number of balls in $\mathcal{U}_i$:

$$\Pr\left[\mathcal{U}_i \text{ is chosen}\right] = \frac{(1-p)(i+\alpha)F_i(n)}{k(1+\alpha p)+\alpha(1-p)}. \tag{4.54}$$

At each time step, exactly one new pin appears, either along the new ball inserted in the first urn or as a result of transferring a ball one urn up. Therefore at time t, there are in total exactly t pins attached to the balls in the urns. In terms of the Web graph, the pins are incoming links, adding a new ball corresponds to the creation of a webpage with one incoming link, and moving a ball to the next urn corresponds to adding a new incoming link (the level of preferentiality depending on α) to an existing webpage.

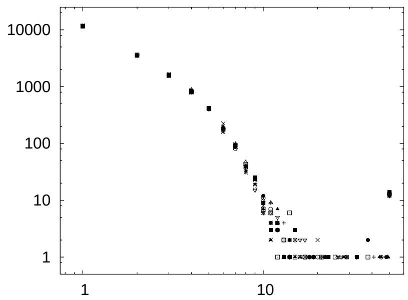
The second model was the Barabási-Albert construction for scale-free networks [25] based on preferential attachment alone. The initial graph[6] $G_0 = (V_0, E_0)$ at time $t = 0$ consists of a small initial set of vertices, $|V_0| = n_0$. At time step t, a new vertex v_t is added to V and assigned d edges; the probability that v_t is connected to $w \in V_{t-1}$ is

$$\Pr\left[\{v, w\} \in E_t\right] = \frac{\deg(w)}{\displaystyle\sum_{u \in V} \deg(u)}. \tag{4.55}$$

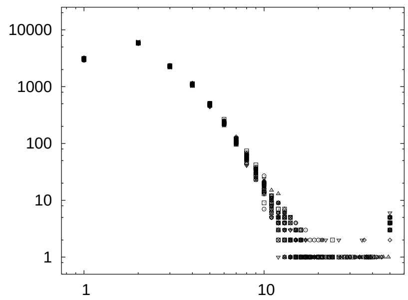
We also included in the comparison the uniform random graph model $\mathcal{G}_{n,m}$ that can be expected to differ from the real data with respect to the cluster distribution. The resulting distributions are shown in in Figure 4.19.

As the CWG is a subset of the complete Web Graph, it is likely that some of the properties of the CWG can be generalized to the Web Graph. As the cluster distribution of CWG may be characteristic of the Web Graph as well, we may assess the validity of web-like graph generators by the obtained distributions; an assessment of Internet graph generators by clustering is reported in [223]. According to Figure 4.19, the evolutionary model provides the best match to G. When choosing a graph generator to be used for a specific application or adjusting the parameters of a generator, a good way to guarantee that the generated instances reflect the properties of the network to be modeled is to measure several different structural properties of the natural data available — preferably distributions instead of plain averages, sampling if necessary due to time or memory constrains — and making choices and adjustments after observing the level of agreement of the generator and the natural data.

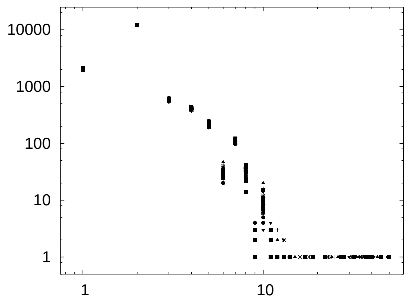
[6]Barabási and Albert [25] do not define what the initial graph is; $E_0 = \emptyset$ is implicitly suggested. This however causes problems as the sum of vertex degrees is initially zero.



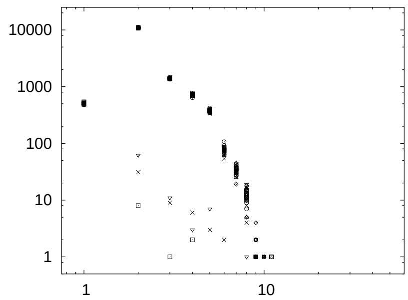
CWG connected component



Evolutionary web-like graphs



Barabási-Albert graphs



Uniform random graphs

Figure 4.19: The cluster distributions (size on the x-axis versus frequency on the y-axis) for the CWG (10 independent clusterings) and the comparison graphs (6 independently generated graphs per model, each clustered 6 times). Cluster order was limited to 50; hence the cutoffs in the distributions.

5 SEARCHING AND ROUTING

Searching in a graph is the process of locating a desired *target vertex* w by traversing the edges of the graph, starting at an *initial vertex* v, as efficiently as possible, i.e., using a strategy that minimizes the number of intermediate vertices visited during the search. A simple search procedure only examines one vertex at a time, referred to as the *current* vertex. Once a search procedure has examined a vertex v and moved on to another one, we say that vertex v is a *visited* vertex. In many applications, several parallel searches or some branching mechanism are practical for improved search efficiency — however, this may cause problems such as network congestion [17].

In general, a search procedure is only allowed to access *local information* of the graph, such as the adjacency list of the current vertex, possibly including the degrees of the neighbors. In some situations, when there is sufficient memory available, it is useful to allow the procedure to remember parts of the graph visited thus far and even record properties of some or all of the the visited vertices. The *search time* is usually defined as the number of edges traversed before the search reaches the desired vertex w.

If the task is not only to locate vertex w starting from v, but also to determine a minimum-length path that connects them, the problem is called a *shortest-path problem* [87], variations of which include the *single-source shortest paths problem* where the shortest paths from a given vertex v to all other vertices are requested, and the *all-pairs shortest path problem*, where a shortest path is to be found over all vertex pairs. As the distance between two vertices is defined as the length of the shortest path connecting them, many distance-based calculations begin with or contain solving shortest-path problems.

A naïve solution to finding the distances from a given vertex v is to perform a *breadth-first search* (BFS) [87] from v until either a given target vertex w is encountered (for the case of a single path construction) or all vertices have been reached. The breadth-first search essentially constructs a spanning tree T for an unweighted graph $G = (V, E)$ the root of which is v. It initializes by placing the edges from v to all of its neighbors not yet in the tree, storing the processed vertices in a FIFO queue, then iterating by popping the top-of-the-queue vertex and adding edges from it to those of its neighbors not yet connected to v in the tree. Each examined neighbor w of v is appended to the end of the queue if and only if $\{v, w\}$ is included in T. Note that after $(n - 1)$ edges have been included in T, the construction is finished, as introducing additional edges would form cycles.

Two general conclusions of much of the existing literature on search and distance problems are that natural networks tend to have surprisingly small diameters [10] and that the network topology may have a significant effect on the search time [94]. The average time to locate a desired vertex in a simplistic small-world network [314] has been studied by de Moura *et al.* [94], who find that a small-world topology minimizes the search time for two scenarios, namely when the computation needed to browse the neighbor list of the current vertex during the search process is negligible with respect to the computation needed to “follow” an edge, and a setting in which both computational costs are significant. Another practical setting is that in which querying an edge weight is the dominating-cost operation of solving such problems [293].

A measure proposed by Sneppen *et al.* [271, 287] for the *searchability* of a graph is a formulation of the (information) *entropy* [283], characterizing the amount of information available for locating a given target vertex v when starting a search at a vertex w. The general definition of the entropy of a random variable X_d with discrete outcomes $x_1, \ldots, x_k$, given the probabilities $\Pr[X_d = x_j]$, is

$$\mathcal{H}(X_d) = -\sum_{j=1}^{k} \Pr[X_d = x_j] \log_2 \Pr[X_1 = x_j]. \tag{5.1}$$

Blindly choosing which neighbor of w to go to when searching for v, we have probability $\frac{1}{\deg(w)}$ for choosing any particular neighbor of w. Continuing from a chosen neighbor u, we know that returning to w is not the answer and hence have $\frac{1}{\deg(u)-1}$ probability over the other neighbors. Hence Sneppen *et al.* define the probability of finding a path from w to v in blind random walk as

$$\Pr[w \to v] = \sum_{P \in \mathcal{P}} \left(\frac{1}{\deg(w)} \prod_{u \in P} \frac{1}{\deg(u) - 1} \right) \tag{5.2}$$

where $\mathcal{P}$ is the set of (degenerate) paths from w to v, and define the information gained by knowing a path from w to v as

$$\mathcal{H}_S(w \to v) = -\log_2 \sum_{P \in \mathcal{P}} \Pr[w \to v]. \tag{5.3}$$

From the above definition they generalize further the searchability of a graph $G = (V, E)$

$$\mathcal{H}_S(G) = \frac{1}{n^2} \sum_{v \in V} \sum_{w \in V} \mathcal{H}_S(v \to w) \tag{5.4}$$

and the *access information* of a vertex v as

$$\mathcal{H}_S(v) = \frac{1}{n} \sum_{w \in V} \mathcal{H}_S(v \to w). \tag{5.5}$$

They employ these measures to analyze the searchability of natural networks as well as adjusting the network topologies to optimize searchability measures.

If no information on the network topology is available, searching for a particular vertex can be done by conducting a random walk on the graph. If the system has memory available, a variation of a "blind", naïve random walk is a *self-avoiding* random walk, where the walk is not allowed to re-visit a previously visited vertex. Such walks, however, suffer from "dead ends" in the graph structure, such as leaf vertices. A milder version of a self-avoiding is achieved with the idea of *tabu search* [1, 136], which is a local search method that keeps a buffer of the k most recently visited vertices, and will not return to those, but may return to any vertex visited earlier on in the walk. This approach avoids getting stuck in small cycles, but is not as prone to run into a dead end as a pure self-avoiding walk. More variations of random walks are easily constructed for specific applications, varying the amount of information that is considered to be available to the walker, as well as allocation of computational or memory resources to the vertices or the walker itself. If for example each vertex also knows its neighbors and the walker has access to a large memory, we may implement a *neighbor-avoiding* random walk with the aim of quickly exploring the entire graph.

There have been various research efforts to study the behavior of different types of random walks that employ different information of the graph topology and their efficiency in locating a given vertex with respect to the average length of the walk needed and the probability or frequency of reaching the target vertex. Random walks with discrete power laws for the walk lengths have been formally analyzed by Annibaldi and Hopcraft [15].

Tadić [294] studies how random walks with *adaptive* move strategies proceed in directed networks resembling the Web graph. The test networks have been generated by her own model of *directed* scale-free networks that grow and rearrange, using preferential attachment both in growth and rearrangement [295]. The model produces power laws for both the in-degree and the out-degree distributions. The adaptive random walk is allowed to use *locally available* information, in particular the out-degree of the current vertex and the in-degrees of the neighboring vertices, to decide where to proceed. The edges are assigned weights to make high in-degree vertices more likely to be visited than they would be under the regular random walk.

Tadić finds that for certain parameter values of her model, indicating a high degree of “rewiring” in the graph, an adaptive random walk proceeds to some fixed level of hierarchy in the graph considerably quicker than a regular random walk. The difference in access time is some orders of magnitude [296]. Hence she concludes that such an adaptive walk can pass messages efficiently for that particular class of Web-like graphs when the degree of rewiring is large [294, 296].

Adamic *et al.* [5] study the behavior of search algorithms in power-law graphs such as the Barabási-Albert scale-free graphs; similar work was done by Kim *et al.* [184]. The goal to find efficient algorithms for that particular ensemble as so many natural networks have been shown to display a power-law degree distribution. The emphasis is on *distributed search* that lacks global knowledge or control, which was also the starting point of Kleinberg’s lattice model [189]. Distributed search algorithms, not requiring a central server to have complete knowledge of the network topology, are necessary in ad hoc networks that are important in mobile communication [88] (cf. Section 4.5.1). Methods for “augmenting” graphs to be locally navigable were recently addressed by Duchon *et al.* [105].

Adamic *et al.* propose a decentralized algorithm that exploits the power-law topology to make the search more efficient. Even though searching with a regular random walk is more likely to visit high-degree vertices, they impose scaling to emphasize high-degree vertices during the search and construct a message-passing algorithm based on this principle and apply variants of this to power-law graphs. The variation is mainly on the knowledge that a vertex possesses of its neighborhood while passing a message. The scaled approach passes messages somewhat faster than a regular random walk for the power-law ensemble.

Properties of self-avoiding walks on scale-free networks have been studied by Herrero [150], with emphasis on the path attrition problem. He uses approximate analysis to deduce the number of walks surviving after N steps, i.e., walks that are still able to proceed in contrast to those that have reach a "dead end" they cannot escape from without revisiting vertices. Yang [325] studies the behavior of random walkers on complex networks and finds that the self-avoiding walk is the optimal strategy if no global information is available. Addition of preferentiality does not help in improving the search according to Yang.

Zhu and Huang [329] study search using only local information for the Watts-Strogatz small-world model, assuming that each vertex knows how to get to any vertex that is at distance k or closer, having the vertices use this information in making routing decisions when sending and forwarding messages in the network.

A *peer-to-peer* (P2P) network is a distributed system composed of independent computers that work together to achieve a common goal, usually involving the sharing of computing, file or network resources. The possible architectures of such systems are [215]:

1. networks with centralized topology and content information and centrally governed evolution of the structure, such as the original Napster, which is based on a full directory of peers;

2. decentralized but structured networks, such as Freenet [72], where the topology is imposed in a central manner but the network functions in a decentralized manner; and

3. decentralized and unstructured networks, such as Gnutella [267, 268].

For a comprehensive review, we recommend the survey of Androutsellis-Theotokis and Spinellis [13]. Traditionally, file-exchange P2P network performance (namely routing of requests for content) is achieved by replicating or moving content. According to Cooper [81], there exist three alternative techniques for improving P2P performance:

(i) reorganizing the network topology to make random-walk search efficient;

(ii) biasing the document count to find the "content hubs", giving preferentiality to where there is a lot of content when choosing where to proceed with a given search task;

(iii) and using memory to maintain a search cache of some kind, such as only remembering the document counts for the k "best" peers.

Cooper concludes that setting the degree of a network peer proportional to the square root of the "popularity" of the contents that it holds provides optimal topology for random walk searches. The maintenance of a search cache coincides with the idea of lookahead buffers discussed later in this chapter In P2P-systems, the choice of the stored entries depends not only on the topology of the network but also on the resources provided by the peers.

Flooding of resource request, i.e., performing a BFS over the entire network by having all peers forward to all of their neighbors any new request messages they receive, has been widely used in early-generation P2P systems, such as Gnutella, despite its poor scalability [215]. A naïve method for trying to limit the circulation of messages in the network, especially if the peers can not be expected to recognize previously seen messages, is to assign a *time-to-live* (TTL) to each message, i.e., a counter on how many peers it is allowed to pass through, and have each peer decrement this counter as it forwards the message, such that any peer receiving it with TTL zero will not forward it further. Choosing a TTL value is not easy when there is little or no information on network structure — especially considering the small diameters of such networks. And even with TTL, the amount of messages sent may be large in certain network topologies. Reachability problems caused by TTLs are discussed by Annexstein *et al.* [14].

In order to limit the generated traffic, probabilistic approaches on when and to whom to forward an arriving request message have been proposed. Banaei-Kashani and Shahabi [22] provide theoretical results on probabilistic flooding and self-avoidance of random walks on P2P systems. Wang *et al.* [311] use a variation of probabilistic flooding resembling simulated annealing. Lv *et al.* [215] present a method that uses multiple random walks, achieving coverage of the system almost as quickly as flooding, but with traffic reduced by up to two orders of magnitude.

Gkantsidis *et al.* [134] propose using combinations of short random walks that finish by performing a shallow flooding (i.e., a BFS-procedure only to limited depth from the source vertex) that corresponds to having a lookahead table at the final peer. They conclude that even a one-step lookahead gives a significant advantage especially when the degree distribution is nonuniform. Sarschar *et al.* [276, 275] propose an algorithm for search in P2P networks using content caching, query implementation (on a short random walk), and bond percolation (i.e., probabilistic broadcast). Ganguly *et al.* [128] discuss search in P2P systems based on ideas of epidemic spreading, performing random walks for packets until a *profile similarity* in the present vertex is above a threshold. The topology is then modified to to move a vertex v "closer" to the peers who perform searches looking for content that v possesses. Chirita *et al.* [66] discuss personalization of PageRank-like preferences in P2P.

In finding a short path between two vertices in a given graph, there are many aspects to be considered. One may try to find a shortest path, but in many cases, the computational cost for this is high. This is especially the case in large or dynamically modified graphs. In a weighted graph, if the cost of determining the edge weights is high, then the algorithm must also try to minimize the number of edge-weight queries it makes when finding the shortest path [293]. Should there exist a good approximate solution, i.e., a path with length only a little above the optimal length, but with significantly lower construction cost, many application areas could benefit from using such an approach. The optimality of the path length becomes significant when communication is repeated over the path several times, whereas for one-time communication, keeping the computational cost of finding the path low is more relevant.

Finding paths by random walks usually produces paths much longer than the optimal, but with postprocessing, any cycle is trivially removed. Also, with access to the neighbor lists of the vertices included in the path, the path may be “straightened out” to achieve a shorter one. In distributed computing, such a path-straightening procedure can be executed while returning the information of the newly found path from the target vertex back to the source vertex by having each vertex on the way back take a short cut to the earliest of its neighbors on the path list instead of forwarding the message to the neighbor from which it originally received the path request message. We leave this idea for future work.

5.1 RANDOM WALKS WITH LIMITED LOOKAHEAD

In a radio-channel network, each node is aware of those neighbors only from whom it has received a beacon signal[1] in a determined time slot. Also the total number of neighbors of which it may be aware at a certain time may be limited due to memory constraints. Under such conditions, the path search is always done under incomplete information of the neighborhood structure.

Instead of conducting a blind random walk or a flooding among the presently acknowledged neighbors, this partial information could be used to limit the number of messages traversing the network without severely degrading the search time. For such conditions, communicating the currently known neighbors in each beacon signal transmitted allows for maintenance of a *lookahead* buffer of the second neighbors of each node.

In practise it is well-known that allowing a one-step lookahead significantly improves the efficiency of search, also when conducted as a random walk in a complex network. Manku *et al.* [216] show that greedy routing with one-step lookahead is optimal. An effective approach for search by random walks is to perform a short random walk followed by a shallow flooding, corresponding to a small lookahead [134]. Such observations have many applications, especially in the field of P2P networks.

In scale-free networks, even storing the list of the immediate neighbors is laborious for hubs due to their high degree. For any neighbor of a hub, the number of second neighbors is at least the same as the degree of the hub, and vertices with several hubs as neighbors suffer from an explosion in the number of second neighbors. These observations combined with the nature of radio-network communications discussed earlier suggest that in the real world, incomplete information of the second neighbors is a more realistic assumption for decentralized search.

With this motivation, we propose and study alternatives for imposing a limit on the number of second neighbors that each vertex is aware of. Our focus is on *distributed search* that lacks global knowledge or control, which was also the starting point of Kleinberg’s lattice model [189]. We study the sim-

[1] In radio networks it is customary that each node periodically broadcasts a message to inform the others of its presence and identity. This message is called a *beacon signal* and can usually be modified to carry additional information as well, such as information on nodes that the sender has communicated with previously.

plest possible search: blind random walks that at each vertex select uniformly at random one of the neighbors of the current vertex to move to, considering the following three variations:

1. A regular random walk that may only examine whether the target vertex is the neighbor of the current vertex.
2. A regular random walk with full second-neighbor lookahead at all steps, i.e., it may also check whether the target vertex is a neighbor of a neighbor.
3. A regular random walk that may check from a list of k second neighbors selected locally for each vertex whether the target is included, in addition to checking whether it is a neighbor of the current vertex.

In order to fill the k-buffer of vertex v, we examine the following methods to choose the included vertices from $\bigcup \Gamma(w)$ where $w \in \Gamma(v)$:

1. Uniformly at random.
2. Proportionally to $\deg(w)$, favoring large-degree vertices.
3. Proportionally to $\frac{1}{\deg(w)}$, favoring small-degree vertices.

No duplicate elements are allowed in the lookahead buffer. The justification of the above choices is that in a scale-free network, the degree of a vertex tends to dominate its role and properties in the functions performed on or by the network. Favoring large degree vertices is sensible if most search targets are hubs. Nonetheless, hubs are reached rapidly even without resorting to lookahead buffers, whereas the small-degree vertices are hard to arrive at by a random walk. Hence we include a heuristic that improves the "visibility" of these vertices by placing them more likely in the lookahead buffers of their neighbors. For comparison we include the uniform random selection, so that we can better evaluate whether the heuristic used to fill the buffers influences the search performance.

In ad hoc networks [88], a feasible way to communicate the second-neighbor information would be to attach a neighbor list to the beacon signal that the nodes send. If nodes have too many neighbors to fit in the beacon packet, they could select (using, for example, one of the above selection criteria) a subset to include at each beacon interval. Nodes that overhear this beacon signal would thus learn at least partial information of their two-hop neighborhoods and could fill their lookahead buffers (using a selection criteria, if the candidates outnumber the slots) upon receiving the beacon signals of their neighbors. It would be of interest in future work to study how such a dynamic environment with nodes joining and leaving the network and communicating within a specified range can be efficiently searched with dynamic and limited lookahead buffers; a further complication would be added by allowing node movement. In P2P networks one usually searches for a specific content on any node instead of a particular node [134]. In this case, the lookahead would not consist of knowing just the identity of the second neighbor, but also a (partial) listing of the content that it provides in the network.

We ran search by random walk for all vertex pairs in a 503-vertex collaboration graph and a 202-vertex neural graph of the nematode *C. Elegans* (constructions for both are given in previous work [306]) taking 30 repetitions and limiting the length of the random walk (TTL) to 300 steps. We ran the experiment set using no lookahead, full lookahead, and for $k \in \{5, 10, 20, 40, 80, 120\}$. The same was repeated for scale-free graphs with clustering [154] of similar order and size: one of order 503, using 0.7 clustering probability, linking rate of 2, and a seed graph of 7 vertices, and another of order 202, using 0.9 clustering probability, linking rate of 10, and a seed graph of 15 vertices. In the test set we also include the fourth and fifth generation DGM graphs [101]. See Table 5.1 for more details on the instances studied; their degree distributions are plotted in Figure 5.1. We keep the k-buffer contents static over the execution of a single experiment; we study the effects of reselecting the buffer contents later in this section.

Table 5.1: Properties of the three instances studied in the experiment set: the collaboration graph (Collab.), the neural network (Neural), the scale-free graphs with tunable clustering (CSF), and the deterministic scale-free DGM graphs.

Property	Collab.	Neural	CSF 503	CSF 202	DGM4	DGM5
Vertex count	503	202	503	202	123	366
Edge count	828	1954	999	1894	243	729
Maximum degree	49	47	48	75	32	63
Clust. coeff.	0.648	0.305	0.088	0.247	0.791	0.797

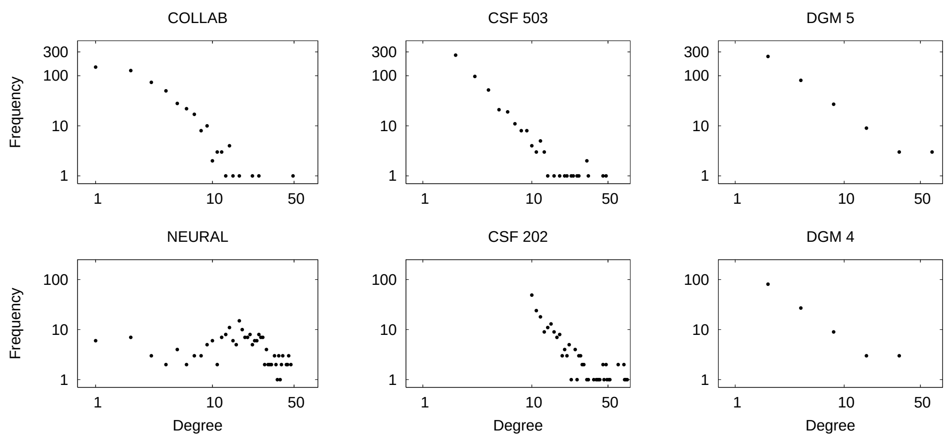


Figure 5.1: The degree distributions of the six instances studied: the collaboration graph (*Collab.*), the neural network of *C. Elegans* (*Neural*), the scale-free graphs with tunable clustering (CSF), and the deterministic DGM graphs.

The gray-scale matrices of Figure 5.2 (shown only for the three smallest graphs due to large size of the graphics) have been drawn in the following manner: if the position (i, j) is white, it means that no path from i to j was found during the 30 random walks, 100 steps each. If it is black, $i = j$ as all the paths were of length zero. The shades of gray represent path lengths between one and 102 steps (a 100-step walk may end in a vertex that knows the target to be a second neighbor, hence 102), with short average path length indicated by dark shades and long average path length by light shades. Hence, the darker the matrix as a whole, the better the search performance over the whole graph. From Figure 5.2 it can be seen that allowing a moderate-size lookahead buffer of second neighbors gives an advantage over the blind random walk: the color-map matrix gets noticeably darker, especially in the upper half. The values of k chosen are not close to the full lookahead, as many vertices have second-neighbor counts significantly above thirty, as illustrated in Figure 5.3.

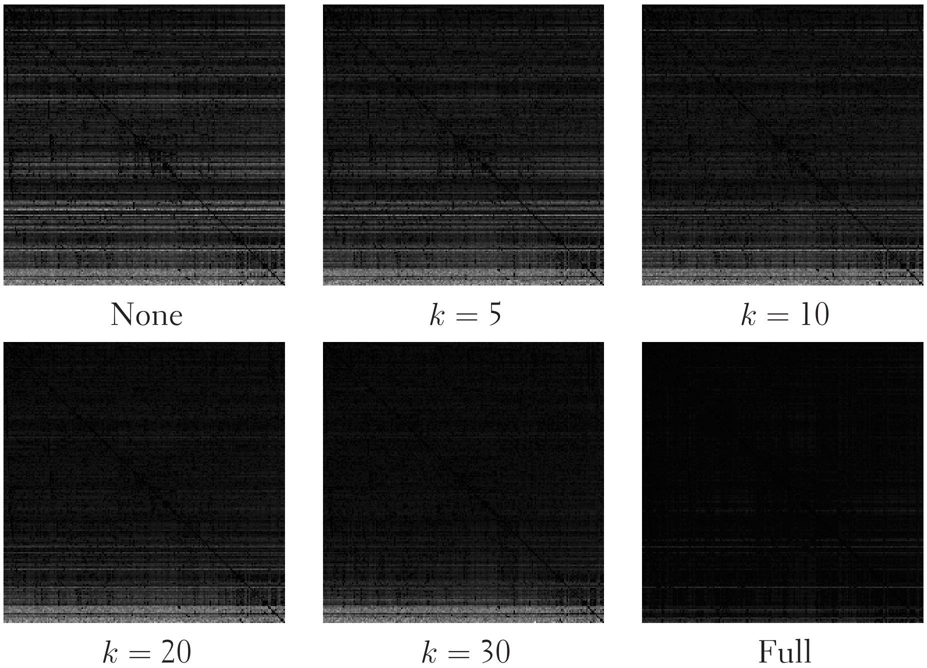


Figure 5.2: Average path-length matrices for the *C. Elegans* neural network; the top-left had no lookahead, the bottom-right matrix is based on full lookahead, and the others had a k-place buffer filled by uniform-probability selection. The color-code is the following: a black position is one where a path was found 100% of the time and a white one corresponds to no paths found. The gray-scale interval is regularly quantized to 256 shades of gray.

We also computed the average path length and the standard deviation over all the paths found in each experiment to obtain a single figure for the global performance of the different setups. A more complete data set is given in Table 5.2 showing the standard deviation and the percentage of walks that actually reached the target vertex; the averages alone are plotted in Figure 5.4 for ease of comparison. Note that for the collaboration graph, the full lookahead does not outperform the limited lookahead, unlike in the other two instances. We repeated the full lookahead search several times and always found similar results. Hence it is not a one-time anomaly, but rather a property of the graph. For the three larger graphs, 300-step walks were taken, whereas for the smaller, 100-step walks were used.

Table 5.2: The average (Avg.) over all paths found for each of the three larger instances, together with the standard deviation (SD) and the percentage (%) of successful searches (right).

k	Crit.	Collab.			CSF			DSF		
		Avg.	SD	%	Avg.	SD	%	Avg.	SD	%
No lookahead		110.23	88.72	41.16	84.23	76.00	84.59	73.34	74.69	86.77
5	1	97.29	87.86	46.43	77.76	75.53	85.27	65.29	72.89	87.68
5	2	97.37	87.74	47.11	77.54	75.57	85.28	63.53	71.91	88.19
5	3	98.80	88.23	45.76	78.68	75.69	85.15	67.47	73.37	87.52
10	1	93.68	87.72	47.86	74.08	74.90	86.31	59.80	71.01	88.82
10	2	95.21	88.05	47.10	75.86	75.25	86.05	64.10	72.73	88.07
10	3	95.06	87.86	47.63	75.86	75.43	85.88	63.86	72.24	88.16
20	1	90.99	87.26	51.71	68.43	72.56	89.87	55.01	69.51	89.58
20	2	90.50	87.06	51.93	67.64	72.48	89.82	53.33	68.93	89.68
20	3	91.49	87.28	51.24	69.45	72.86	89.68	58.83	71.08	88.99
40	1	87.86	85.62	58.03	57.02	66.58	93.17	48.44	66.39	91.42
40	2	87.75	85.65	58.03	57.82	67.25	93.01	47.61	66.39	91.18
40	3	89.55	86.18	57.35	58.90	67.46	92.89	52.07	68.96	90.54
80	1	82.90	83.33	64.62	41.45	55.35	97.10	38.87	60.32	93.92
80	2	82.63	83.23	64.72	41.25	55.36	97.13	37.45	59.44	93.98
80	3	83.57	83.75	63.79	42.40	56.26	97.01	41.15	62.09	93.61
120	1	81.36	82.09	68.60	34.12	47.50	98.82	33.56	55.89	95.35
120	2	81.24	82.06	68.67	33.82	47.20	98.85	32.96	55.62	95.44
120	3	81.24	82.07	68.60	34.02	47.14	98.85	35.69	58.03	95.00
Full lookahead		83.18	83.39	65.54	23.95	32.82	99.73	21.75	36.35	99.65

We also constructed average path-length matrices as those shown in Figure 5.2 to compare how the three selection heuristics compare. In the figures there are no apparent differences, and the numerical differences shown in Table 5.2 are small as well. It may also be an effect due to the randomness in the selection method and not a consequence of the selection criterion used.

Figure 5.4 reveals how the shapes of the curves for all three larger graphs are strikingly different; for the collaboration graph, the effect of using even

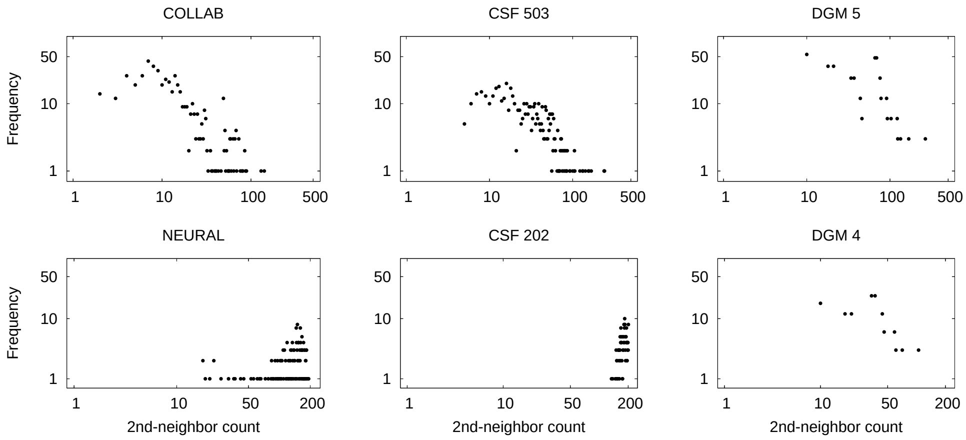


Figure 5.3: The frequencies of the second-neighbor counts of the six instances studied: the collaboration graph (*Collab.*), the neural network of *C. Elegans* (*Neural*), the scale-free graphs with tunable clustering (CSF), and the deterministic DGM graphs.

a small lookahead is bigger than for the others, whereas in the two artificial graphs the full lookahead search is by far the best. For the clustered scale-free construction, the biggest gain appears when using 80-slot buffers instead of 40, whereas for the deterministic construction the gain is relatively linear. There is no obvious difference in how the three selection criteria perform, although overall criterion three — preferring small-degree vertices — seems to have the poorest performance.

As the filling of the k-place lookahead buffer is probabilistic, it is also of interest to see how the performance varies over different buffer contents. Fixing $k = 80$ and using the uniform-probability filling strategy, we repeated the path search experiments for all three instances 30 times and plotted the average path length in Figure 5.5. The variations between the experiment sets are relatively small, only a couple on average, which indicates that the contents of the buffer do not usually have a drastic effect on the overall performance. Although it is evident that on a local level, the "searchability" of a particular node depends heavily on whether the nodes in its vicinity include it in their buffers.

We have examined the effect of limited lookahead buffers for search by random walk, allowing each vertex to remember k of its second neighbors, i.e., vertices that are two hops away. The benefit from this lookahead is evident in the experiments, but it is not clear whether one of the filling criteria would be significantly better than the others. In further work, it would be of interest to study how *weighted* random walks behave under limitations of the lookahead, especially considering the preferential walk for power-law graphs [5]. Also, allowing neighboring vertices to communicate in order to optimize the contents of their lookahead buffers would be of interest.

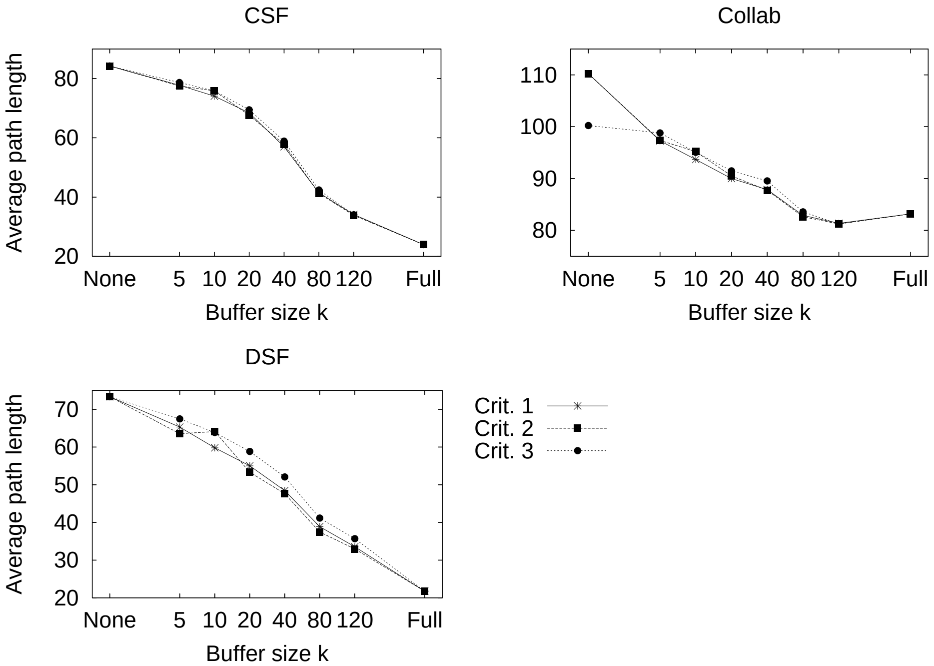


Figure 5.4: The average path lengths over all vertex pairs and the 30 repetitions for different values of k and different filling strategies for the three larger instances.

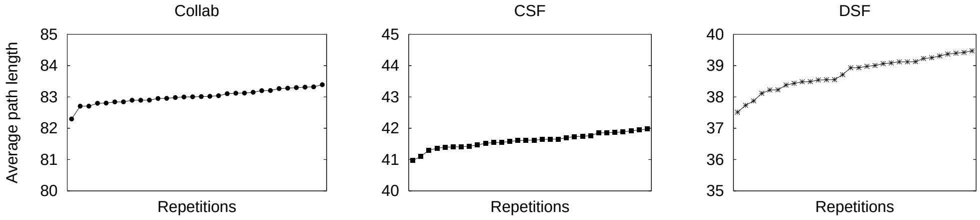


Figure 5.5: The averages of the lengths of all paths found over 30 repetitions of the experiment set using $k = 80$ and the uniform-probability filling strategy for the three larger graphs. Each experiment set consists of 30 repetitions per vertex pair and a cut-off at 300 steps as above. The repetitions are sorted in increasing order of the averages to better reveal the magnitude of the variation.

5.2 SPANNING TREES

We consider networks formed by nodes that are connected by edges. Each edge is assigned a weight that determines the cost of communicating a single packet along that edge. Usually, especially in range-based neighborhood formation, a node v may have various neighbors and it is necessary to determine which of these neighbors are in charge of forwarding communications received from node v — if every node would always forward any communication it receives, the network would easily be flooded and the energy wasted in making these transmission could quickly disable a battery-operated network. One way to decide how the nodes should route their traffic is to build an MST of the weighted graph and have each node adjust their transmission range such that their transmissions can be received and forwarded by their parent and children in the resulting tree [207, 318].

To construct a spanning tree for a given input graph $G = (V, E)$ with n vertices (each representing a node of the network), we must select a subset of $n - 1$ edges such that each edge connects two vertices that correspond to nodes that are able to communicate directly (the weight of the edge should capture the cost of the communication) and that every vertex is reachable through the set of selected edges from any other vertex. In an ideal case, the spanning tree construction could be done in a distributed manner having each of the vertices execute an algorithm that only resorts to local information. Construction of a certain kind of spanning subgraphs (for information dissemination) based on local information are discussed by Stauffer and Barbosa [289].

A task of special interest is to select among all possible subtrees a minimum spanning tree (MST). We are assuming the edge weights to be interpreted as distances or costs, hence preferring smaller values to larger ones (and disallowing negative weights). For unweighted graphs, however, all spanning trees are equal in this sense and another measure is needed to distinguish between different spanning trees when one needs to be chosen as the "backbone" of communications or for performing some other task on the graph. In this section, we will combine approaches that produce a "beneficial" spanning tree for an unweighted graph with those constructing an MST for a weighted graph.

5.2.1 Communication-cost models

In radio-communication networks, *routing* of the network traffic should be done efficiently with respect to the time consumed and the energy required from the nodes of the network that take part in passing on the information [172]. We denote the furthest distance to which a node can communicate by r and call it the *range* of the node. In this work we only deal with uniform nodes, that is, all nodes present in the network are assumed to have the same range. We do not address problems caused by interference of broadcasts or the broadcast storm problem of message propagation; such issues are addressed by many recent studies on ad hoc networks [158, 208, 211, 212, 245, 257], but rather discuss a fundamental building block of many routing protocols: the construction of a spanning tree of the network.

We represent such a network as a weighted graph, where each node is represented by a vertex and the nodes that are within the transmission range of a node will be connected to the representative vertex v with edges $\{v, w\}$ weighted by the Euclidean distances of the nodes in the physical space in which the network operates,

$$\omega\left(\{v, w\}\right) = \mathrm{dist}_{\mathrm{Eucl}}\left(v, w\right). \tag{5.6}$$

Traditional cost models take into account the transmission power needed for two nodes to communicate, which is proportional to the physical distance between the nodes. In the real world, the *energy consumption* $\mathcal{E}$ (often referred to as the *cost* of the communication) of making a transmission over a distance $\mathrm{dist}_{\mathrm{Eucl}}\left(v, w\right)$ is proportional to a *power* of the distance instead of being linearly dependent on it:

$$\mathcal{E} \sim \mathrm{dist}_{\mathrm{Eucl}}\left(v, w\right)^k. \tag{5.7}$$

In vacuum-like conditions $k \approx 2$, but in most real-world scenarios, a range of values $k \in [2, 4]$ has been observed [60] (although workarounds exist for some setups to lower the value of k [291]).

The efficiency with respect to battery usage of radio transmitters is usually better for high distances close to the maximum range than for small ones, as the range is proportional to the square-root of the transmission power. There are many different path-loss models proposed for analyzing and simulating radio networks [166, 274], but we limit to studying a simplified case fixing $k = 3$ for the transmission power.

Also, in addition to having a distance-dependent element in the cost of communicating between two nodes, there is also an *initial* transmission cost for any node that makes a broadcast that does not depend on the transmission range used, and additionally, an initial *receive* cost per each transmission that a node needs to receive. One estimate used in the literature is that the receive cost $\mathcal{R}$ is proportional to the maximum transmission range r of a node [114]:

$$\mathcal{R} = \frac{r^2}{5}. \tag{5.8}$$

Unfortunately such estimates are not universal, and hence no general rule exists for how big the initial transmission and receive costs are for a given application. In another simplified communication-cost model [141], the energy consumed by transmission per bit over distance $\mathrm{dist}_{\mathrm{Eucl}}(v, w)$ is

$$\mathcal{E}_t = \alpha_t + \alpha_{\mathrm{amp}} \cdot \mathrm{dist}_{\mathrm{Eucl}}(v, w) \tag{5.9}$$

and the energy consumed per bit in receiving is

$$\mathcal{E}_r = \alpha_r, \tag{5.10}$$

where α_t is the energy per bit consumed by transmitter electronics, α_{amp} is a coefficient for (again per bit) the energy consumed in amplifying the signal to reach the requested distance, and α_r is the energy consumed by the receiver electronics per bit.

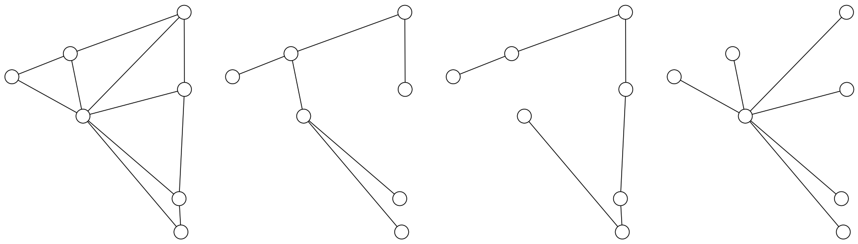
The initial transmission costs and receive costs have often been ignored and distance-based spanning trees used as a starting point for routing. After all, these costs are easily, although misguidedly, considered just an equal-size addition to all of the edge weights that would not affect the minimization. Minimizing the total weight of the tree will not be any different if all edge weights are multiplied by a constant or additive constants are introduced to all edges. Hence, with this approach, simple minimization of the total sum of edge weights given by Equation 5.6 has been employed.

Taking a slightly more realistic perspective, if there are two possible communication paths between a given pair of nodes that have the same total edge weight (of Equation 5.6), the one with *fewer* intermediate nodes should result in a lower total cost when the initial transmission and receive costs are taken into account. This motivates methods for construction of spanning trees that not only minimize the total weight of the included edges but simultaneously minimize the number of nodes that will take part in a communication between two arbitrary nodes [279]. One measure that helps in choosing a spanning tree that keeps the number of intermediate nodes low is the *average path length* of an *unweighted* version of the spanning tree T, $\mathcal{L}(T)$.

We call trees that are minimal with respect to the unweighted average path length *minimum-hop trees* (MHT). In the literature they are known at least as MAD trees and *minimum routing-cost spanning trees*; the construction of such trees is in general **NP**-hard [169] for undirected weighted graphs, but approximations and exact polynomial algorithms for special cases are known [91]. A related class of spanning trees are those that instead of minimizing the average hop-based distance, minimize the (hop-based) *diameter*, known as *minimum-diameter spanning trees* (MDST) [227] that can be constructed in $\mathcal{O}(mn + n^2 \log n)$ time [146]. However, constructing a minimum-diameter MST is **NP**-hard [226].

A third, largely independent consideration is the *load* of the nodes, i.e., the amount of traffic that passes through each node: the more frequently a node has to take part in communication between other nodes, the heavier its load becomes. If a node has high degree in the tree, it is likely to have a higher load — the maximum degree of minimum spanning trees is discussed by Robins and Salowe [270] and load in hierarchical lattice graphs by Arenas *et al.* [17].

One possibility to model vertex load is to use the *edge-betweenness* [241] of the vertices in an *unweighted* version of the spanning tree $T = (V, F)$ used



Load $\delta_\ell(T)$	854	588	1,350
Load $\delta_\ell(F)$	362	70	1,350
Avg. hop count	2.05	2.66	1.71

Figure 5.6: A small graph (on the left) with three possible spanning trees: a random one, a line, and a star topology. Below the pictures are the values for two possible global load measures (using the vertex-betweenness) and the average hop count (calculating the hop-distance for each pair of distinct vertices once).

to route the traffic: the load of an edge is proportional to the number of paths connecting any pair of vertices that pass through that edge; in a tree any path for a given pair of vertices is always the unique shortest path. The load of a vertex v can either be measured as the sum of the edge-betweenness of the edges incident on it in the spanning tree or by the *vertex-betweenness*, which is defined equivalently as the number of shortest paths that pass through (or begin or end) at v; we will use the latter definition.

In the optimal situation, all vertices would have the same load — this is of special importance when the communication nodes are battery operated, as a heavy-load node runs out of power faster after which it will not be able to take part in the function of the network. One way to measure how evenly the load has been distributed is sum of squares of the load differences; denoting the load of a vertex v by $\ell(v)$, we can use either the load over all distinct vertex pairs

$$\delta_\ell(T) = \sum_{v,w \in V} \big(\ell(v) - \ell(w)\big)^2 \tag{5.11}$$

or, denoting by F the edges that are included in the spanning tree T, over only the edges of the tree,

$$\delta_\ell(F) = \sum_{\substack{v,w \in V \\ \{v,w\} \in F}} \big(\ell(v) - \ell(w)\big)^2. \tag{5.12}$$

As argued above, good spanning trees for communication networks are those that use low-weight edges, have small unweighted average path length, and in which the load is uniformly spread over the network, meaning that $\delta_\ell(*)$ is small. Unfortunately, these goals are often contradictory, as illustrated in Figure 5.6, where the star topology clearly achieves the smallest average distance, but puts a heavy load on the central vertex.

The worst-case tree T with respect to the average hop count is a "line" of

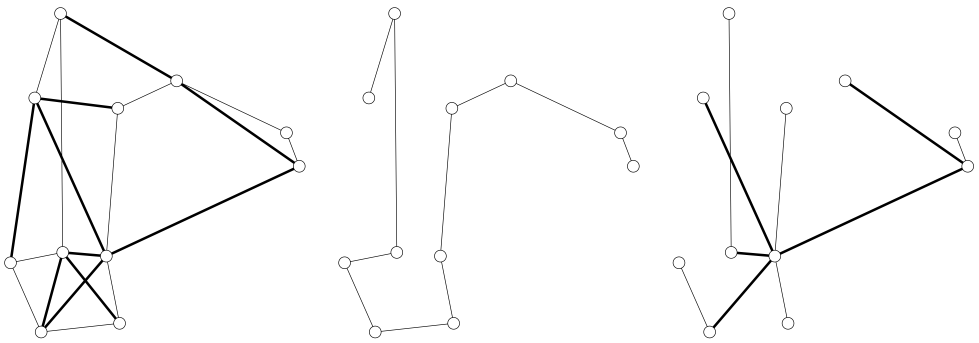


Figure 5.7: A 11-vertex graph $G = (V, E)$ and two possible spanning trees; the edges shown thick have weight $1 + \epsilon$ ($\epsilon > 0$) and the thin ones have weight 1. The MST (with total weight 10) is a path with average path length 4, whereas the other tree has unweighted average path length 2.36, weighted average path length $2.36 + 1.472\epsilon$, and total edge weight $10 + 5\epsilon$. For small values of ϵ, the latter tree is clearly better with respect to the number of hops needed on average to carry out communication between to vertices.

length $n - 1$, for which the average path length is always

$$\begin{aligned} \mathcal{L}(T) &= \frac{1}{n(n-1)} \sum_{i=1}^{n} \left(\sum_{j=1}^{i-1} j + \sum_{j=1}^{n-i} j \right) \\ &= \frac{1}{n(n-1)} \sum_{i=1}^{n} \left(\frac{i(i-1)}{2} + \frac{(n-i)(n-i+1)}{2} \right) \\ &= \frac{1}{n(n-1)} \cdot \frac{n(n^2-1)}{3} = \frac{n+1}{3}, \end{aligned} \tag{5.13}$$

where the vertices are denoted by their index $i \in [n]$, sequentially numbered from one end of the path to the other. It is easy to construct graphs where the minimum spanning tree is a path, but where there exists an alternative spanning tree with near-optimal cost and much smaller average path length; see Figure 5.7 for an example.

In this work we do not propose algorithms for load balancing, but rather use measures of the evenness of the load to evaluate spanning trees generated by optimizing other criteria. Our goal is to find methods that construct non-minimal spanning trees with respect to the edge weights, but with desirable properties such as those of the leftmost tree in Figure 5.6 and the rightmost one in Figure 5.7: moderate load, near-optimal total edge weight, and a small average hop length.

5.2.2 Centralized tree-construction algorithms

In order to gain a better understanding of the problem at hand, we first discuss optimizing each of the properties — total edge weight and average path length — separately with a centralized algorithm. The standard minimum spanning tree algorithms [157, 162, 179, 260] solve efficiently the weighted MST problem, always finding the optimal solution.

For minimizing the average hop count, we propose the following simple heuristic that produces a spanning tree $T = (V, F)$ for a given *connected*

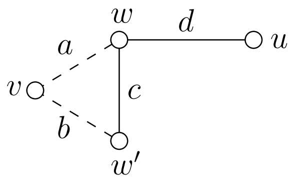


Figure 5.8: A small example graph where vertex v needs to choose whether to link to w or w' (the dashed edges) to form a spanning tree used to route communications. Edge weights (a, b, c, d) are shown next to the arcs; the length of the edges in the drawing is does not reflect the weights. We set $a > b$.

graph $G = (V, E)$ (ignoring possible edge weights) such that the average path length (in terms of hops) of T is low. It draws from the observation that scale-free networks usually have low average path length [25], attempting to construct a tree that is as close to a star topology as possible, promoting high-degree vertices to serve as hubs to their neighborhood.

For a disconnected graph, the following simple procedure can be repeated for each component to obtain a spanning forest. The main loop of the algorithm is executed at most $n-1$ times, as each repetition connects at least two vertices by an edge and the resulting tree necessarily has exactly $n-1$ edges.

(i) Initialize a union-find data structure such that each vertex v forms its own singleton set $\mathcal{C}(v)$ and mark all vertices as unused.

(ii) Select the unused vertex v that has the most neighbors *outside* its own set. Resolve ties by preferring smaller vertex label.

(iii) For each neighbor w of v (one at a time in increasing order of vertex labels) such that $w \notin \mathcal{C}(v)$, add the edge $\{v, w\}$ into the tree and merge the set $\mathcal{C}(w)$ into the set $\mathcal{C}(v)$ in the union-find structure. Mark v as used.

(iv) Repeat from step (ii) until all vertices are in the same set.

If a graph contains a star-topology subgraph, it always yields the hop-count optimal spanning tree. A solution given by the above approach could very likely be improved with stochastic strategies such as 2-opt [89] or 3-opt [210] moves, choosing a pair or a triplet of edges and rewiring the endpoints if the global solution is improved by the switch. A more sophisticated approach is the PTAS of Wu *et al.* [321], but for our small-scale experiments it suffices to use the above heuristic, as computing the optimum would be exponential in any case. We call trees that are not minimal with respect to the total hop-count, but that have *low* hop counts low-hop spanning trees (LHT).

A simple approach to centrally combine the two goals at hand (small edge weights and small hop counts) is to modify the edge weights to contain a component that depends on the degrees of the source and the target vertex w in addition to being proportional to a power of the inter-vertex distance $\mathrm{dist}_{\mathrm{Eucl}}(v, w)$. We want to favor close-by high-degree neighbors when deciding which edge to use to link a vertex into the spanning tree.

Our goal is to determine a natural and computationally efficient method to incorporate the degrees and the Euclidean-distance component, also taking into account the initial transmission and receive costs $\mathcal{T}$ and $\mathcal{R}$. Consider the simple example of Figure 5.8, namely the situation that vertex v needs to be linked to either w or w' in order to construct a spanning tree T. Denote the total cost of communication along a path $v, \ldots, w$ by $c(v, \ldots, w)$, including the initial transmission and receive costs per each path segment. In the example, $\omega(\{v, w\}) = a$ is larger than $\omega(\{v, w'\}) = b$, but also $\deg_{V\setminus\{v\}}(w) = 2$ is larger than $\deg_{V\setminus\{v\}}(w') = 1$. If v was to communicate once with all of the other vertices, the total communication cost (denoting $\tau = \mathcal{T} + \mathcal{R}$) would be

$$\begin{aligned} & c(v, w) + c(v, w, w') + c(v, w, u) \\ & \quad = \tau + \alpha a^3 + 2\tau + \alpha(a^3 + c^3) + 2\tau + \alpha(a^3 + d^3) \\ & \quad = 5\tau + \alpha(3a^3 + c^3 + d^3) \end{aligned} \tag{5.14}$$

should we choose to include $\{v, w\}$ in the tree, and

$$\begin{aligned} & c(v, w') + c(v, w', w) + c(v, w', w, u) \\ & \quad = \tau + \alpha b^3 + 2\tau + \alpha(b^3 + c^3) + 3\tau + \alpha(b^3 + c^3 + d^3) \\ & \quad = 6\tau + \alpha(3b^3 + 2c^3 + d^3) \end{aligned} \tag{5.15}$$

if we instead selected $\{v, w'\} \in T$. Hence, if vertex is just as likely to communicate with any other vertex in the graph, v should link to w if

$$\begin{array}{rcl} 5(\mathcal{T} + \mathcal{R}) + \alpha(3a^3 + c^3 + d^3) & < & 6(\mathcal{T} + \mathcal{R}) + \alpha(3b^3 + 2c^3 + d^3) \\ \alpha \cdot 3a^3 & < & \mathcal{T} + \mathcal{R} + \alpha \cdot 3b^3 + \alpha \cdot c \\ a^3 & < & b^3 + \frac{1}{3\alpha}(\alpha^2 \cdot c^3 + \mathcal{T} + \mathcal{R}), \end{array} \tag{5.16}$$

which can be simplified by eliminating the effect of the unknown distance c and simply stating that choosing a heavier edge a is cost-efficient in the situation depicted in Figure 5.8 if the cube of the distance is no more than $\frac{\tau}{3\alpha}$ higher than that of the cheaper edge b. Should the difference in degrees be larger, the savings in transmission and receive costs would grow proportionally to the difference in degrees.

We propose building spanning trees such that the tree construction accounts also for the path-formation possibilities instead of only considering the simple distances of Equation 5.6. Assume that the tree construction is done incrementally by joining components of a forest until the spanning tree $T = (V, F)$ is ready. Initially place each vertex in a singleton component in the forest. Two components may be joined by selecting an edge from the input graph $G = (V, E)$ that links a vertex in one component to a vertex in another component; the set of selected edges will form F. Edges connecting vertices in the same component are never considered as candidates for inclusion into T.

We denote the number of neighbors that v has in G that are not in the same component C with v by $\mathcal{D}_v = \deg_{V\setminus C}(v)$. It is of interest in a distributed tree formation for a vertex v to link to a vertex that has many neighbors that are not yet in the same component as v in the formation forest, as those paths that are already formed from v to the members of its component will not benefit in hop count from linking to that neighboring vertex, as cycles will not be introduced into the tree in any case. Note that knowledge of the whole vertex set is not actually needed, but only the lists of neighbors of the vertices in $\Gamma(v)$; this can be easily transmitted in an ad hoc network after the vertices have initiated communication.

As we consider *undirected* graphs, we would like to incorporate in the modified edge weight the *mutual interest* of the two vertices to link to each other. A good spanning-tree edge is one that provides both endpoints with several second neighbors: v wants $\mathcal{D}_w$ to be high, and w wants $\mathcal{D}_v$ to be high, but for the tree as a whole it suffices for one of the participants to be a hub for benefit to be gained in the average hop count. Hence we choose to introduce a factor

$$d = \mathcal{D}_v + \mathcal{D}_w \tag{5.17}$$

in the modified edge weight, deriving the equation from the above example of the effect of the degree (denoting $\tau = \mathcal{T} + \mathcal{R}$):

$$\begin{aligned} \omega(\{v,w\}) &= \tfrac{2\Delta(v)\tau}{\alpha} + \alpha \operatorname{dist}_{\mathrm{Eucl}}(v,w)^3 + \tfrac{d(d-1)\tau}{\alpha(d+1)} \\ &= \tfrac{\tau}{\alpha}\left(2\Delta(v) - \tfrac{d(d-1)}{d+1}\right) + \alpha \operatorname{dist}_{\mathrm{Eucl}}(v,w)^3, \end{aligned} \tag{5.18}$$

where $\Delta(v)$ is the maximum value of $\mathcal{D}_u$ over $u \in \Gamma(v)$ and ensures that all resulting edge weights are positive. Note that the above function is just one possible modification function, but as will be shown later through experiments, it serves for this purpose quite well. The key elements are the presence of the initial costs, degree-dependence, the exponent on the distance, and the possibility of adjusting through parameter selection whether the distance-based element should dominate over the degree-dependent element or vice versa.

When the edge weights are not determined online in a distributed environment but are needed beforehand for centralized tree formation, we substitute $\deg(v) - 1$ in place of $\mathcal{D}_v$, as any other neighbor of v than w itself is a potential benefit for w. This essentially corresponds to having all the vertices evaluate the weights when each vertex forms a singleton component in the formation forest and no edges have yet been added. Also, in a centralized setting, $\Delta(v)$ can be replaced by the maximum degree of the graph minus one, $\Delta - 1$.

We choose this approach of combining the two goals in a straightforward weight-based manner, because a closely related problem of constructing a spanning tree that has upper bounds for both the total weight and the diameter is known to be **NP**-complete [227] and hence we believe the combination at hand, bounding both the total weight and the average distance, to be computationally demanding to solve optimally. We would like to point out the similarity of this approach to *multi-objective optimization*, where a (linear) combination of several fitness functions is optimized in order to find a solution that is "agreeable" with respect to at least two usually somewhat contradictory criteria.

In our experiments in Section 5.2.4 we examine the behavior of these precomputed edge weights with different values of $\mathcal{T}$, $\mathcal{R}$, and α. We assume, based on Equation 5.8, that $\mathcal{R}$ is smaller than $\mathcal{T}$.

5.2.3 Distributed tree-construction algorithms

Many important applications do not allow for a centralized control and hence the methods discussed above are infeasible. One such application are *sensor networks*, which are collections of *sensor nodes* spread around an area in which a certain phenomenon of interest is expected to take place [8, 9]. An example could be a national park, where forest fires are to be avoided — sensor nodes that are capable of detecting e.g. smoke or high temperatures are scattered in the park and the network needs to propagate an alarm to the forest guard in case of a fire. In most cases, the sensor placement is not a carefully designed process but more of a random scattering. Each sensor node is composed of essentially four components [8]:

- the *sensing* unit that makes observations of the environment,
- the *processing* unit that determines what actions need to be taken; this is commonly a limited computational device with little memory,
- the *transceiver* unit that receives and broadcasts signals enabling nearby sensor nodes to communicate; usually the range of the broadcast is somewhat limited,
- the *power* unit — essentially a battery — that supplies energy for the other components; the battery life of the nodes governs the life-time of the network [111, 172, 317].

There exist algorithms to construct minimum spanning trees in a distributed manner [107, 113, 127, 129], having each vertex execute a certain algorithm that contains a communication protocol to exchange information with neighboring vertices. Usually the tree is grown by connecting single vertices to pairs and pairs to triplets and so forth until the spanning tree is complete. The growth is achieved by detecting and adding minimum-weight edges to connect currently disconnected tree fragments.

A classical algorithm by Gallager *et al.* [127] uses at most $2m + 5n \log_2 n$ messages and $\mathcal{O}(n \log n)$ time for the tree construction assuming that all edge weights are distinct (or alternatively that the vertices have unique identifiers for which an ordering exists). An algorithm by Garay *et al.* [129] operates in $\mathcal{O}(\operatorname{diam}(G) + n^{\epsilon} \log^* n)$ and combines the aforementioned growth approach with elimination of candidate edges. In most of these algorithms each vertex v is assumed to have a unique identifier $\mathrm{id}(v)$; an ordering exists for these identifiers. For example any network adapter in a computer has a MAC address that could be used as such an identifier.

If the network undergoes frequent topology changes, one possibility is to use a *self-stabilizing algorithm* [97, 100], where the idea is that a distributed system, starting from an arbitrary initial state, performs some steps that will eventually bring the system into a legal configuration, after which the system will stay within legal configurations until a fault occurs. If the system is composed of communicating nodes that are choosing links to use to communicate, a configuration is any subgraph of the full-connectivity graph of the nodes and a legal configuration with respect to the spanning tree construction would be any spanning tree. Such algorithms have high fault tolerance, as they provide full and automatic protection against transient process failures (i.e., data corruption) [297].

Such algorithms do not necessarily provide accurate results when starting or recovering from a fault or a topology change, and the processes have no way of detecting when to stop computing as new faults may occur at any time. The computational complexity is often worse and also the performance inferior to rigid algorithms. For spanning tree construction, one can employ a distributed BFS variant where one vertex is be selected as the root[2] and all vertices are assumed to be aware of their neighbors [100]. The knowledge of $\Gamma(v)$ is easily achieved with some simple HELLO messages [76]. The idea of the algorithm is that each vertex attempts to compute its distance from the root, reporting the outcome of each iteration to its neighbors, keeping record of their currently chosen parent vertices. When the system reaches a legal configuration, the tree can be read from the parent fields maintained by the vertices. There are also several other self-stabilizing distributed variants of the spanning tree problem [126, 281], including MST algorithms [16] as well as shortest-path problems [300].

What comes to achieving a low average hop count, we simply modify basically any existing distributed MST algorithm. Our goal is to build in a distributed manner a LHT spanning tree T that has low average path length. We achieve this by changing the edge selection criterion from choosing one with minimum weight to choosing an edge that links the component to the highest-degree neighbor, not counting links to the component that is choosing the neighbor to link to. The combined weight measure of Equation 5.18 can also be directly applied in an existing MST algorithm.

5.2.4 Experiments on spanning trees

For experiments on the qualities of the trees produced by the proposed method, we generated a set of random graphs and constructed several spanning trees for each instance. We used a graph generator that places n vertices in a unit square. The basic generator generates the coordinates $x, y \in [0, 1]$ independently, uniformly at random. We choose to use such random uniform networks to ensure that the underlying graph topology is *not* scale-free, so that we may observe whether the LHT method of modified edge weights is capable of constructing such a topology in the spanning tree instead of merely reproducing or amplifying a property inherent in the input graph. Note that we are not addressing radio interference or other such considerations in our experiments, but merely testing how the LHT spanning trees differ from the "traditional" MSTs.

The expected degree of a vertex v with communication range $r > 0$ is directly proportional to the area of the unit square that its communication range covers. If the entire communication range falls within the unit square, which is a reasonable assumption for small values of r, the area covered is πr^2, and as unit square has unit area, the probability that another vertex falls

[2] Distributed leader-election algorithms [252] solve the problem of selecting such a root without centralized control; they are however time-consuming [19] (even reducible to spanning-tree construction) and hence any approach relying on the existence of a leader suffers from the election cost.

within this range is $\min\{\pi r^2, 1\}$ and therefore

$$\mathrm{E}[\deg(v)] = (n-1)\min\{\pi r^2, 1\}. \tag{5.19}$$

We may use this value as an estimate of the expected average degree $\mathrm{E}\left[\bar{k}\right]$ of the graph, although in reality vertices close to the border of the unit square have a lower degree; the average degree is simply $\bar{k} = 2m/n$. Doing so, we obtain an upper bound for the expected density of a graph with n vertices as

$$\begin{aligned} \mathrm{E}[\delta(G)] &= \frac{2\,\mathrm{E}[m]}{n(n-1)} = \frac{2\,\mathrm{E}[m]}{n} \cdot \frac{1}{n-1} = \frac{\mathrm{E}\left[\bar{k}\right]}{n-1} \\ &< \frac{\min\{\pi r^2, 1\}(n-1)}{n-1} = \min\{\pi r^2, 1\}. \end{aligned} \tag{5.20}$$

We ran a set of experiments generating 30 graphs with $n = 1{,}000$ using ranges that give expected density $\mathrm{E}[\delta(G)] \in \{0.1, 0.2, 0.4\}$ using Equation 5.20,

$$r = \sqrt{\frac{\delta(G)}{\pi}}. \tag{5.21}$$

We wanted to ensure that the graphs would be connected in order to produce comparable spanning trees; with density 0.05, for example, some of generated graphs would remain disconnected. For each of the graphs, we generated spanning trees with the above algorithms and computed the total weight of the tree together with the average path length and the average hop count (i.e., the average path length of the unweighted tree). We also computed load measures such as the evenness of the degree distribution of the tree.

As r increases, the average weight of the edges present in the graph grows and hence the hop-count minimizing methods are likely to start using heavier and heavier edges, growing the difference in total tree weight when compared with the MST, but also enabling a lower hop count to be realized.

In our experiments, the minimum-weight spanning trees were computed with Prim-Jarnik algorithm[3] for simplicity, hence using global pre-computation of the edge weights. We varied the values of $\mathcal{T}$, $\mathcal{R}$, and α to study under which circumstances the savings are the highest. For each parameter combination, we used a set of 30 independently generated unit-square graphs of order $n = 1{,}000$. For each graph we also computed the minimum spanning trees based purely on Euclidean distances; these trees are denoted by the abbreviation MST in reporting the results.

The values of the parameters used to globally modify the edge weights according to Equation 5.18 we selected according to the following observations: when varying $\mathcal{T}$ and $\mathcal{R}$, only their sum affects the modified edge weights, which decreases the number of informative parameter combinations and allows us to use a fixed value $\mathcal{R} = 1$. Because the average Euclidean distances in the unit square are small, large values of α are needed to obtain dominance of the distance-dependent term over the degree-dependent one. We ran experiments using all combinations of α from 1,000 to 10,000 in increments of 1,000 combined with $\mathcal{T} \in \{1, 5, 10, 50\}$.

[3] A simple centralized greedy algorithm that picks edges in the order of increasing weight while avoiding cycle-formation [162, 260].

The left plots of Figure 5.9 demonstrate the maximum degrees present in the generated spanning trees. As the degree-dependent component gets stronger (with lower values of α and/or higher values of $\mathcal{T}$), the maximum degree begins to grow as hubs are being introduced into the spanning tree. With even higher values than those displayed in the plot, the method would start producing trees that approximate a star topology by choosing an edge that links vertices to high-degree neighbors regardless of the Euclidean-distance edge weights.

The total weights of the spanning trees are plotted on the right in Figure 5.9. The same phenomenon can be seen here: as α decreases and/or $\mathcal{T}$ grows, heavier edges are being selected. Hence the parameter adjustment clearly allows for different topologies to be chosen. In order to study what we gain for what we lose by increasing the total edge weight, we show in Figure 5.10 the average hop counts together with the average lengths of the paths (the latter use the Euclidean distances). The hop counts behave as desired, with the MSTs having the worst values and the LHTs getting closer to those of the input graphs as $\mathcal{T}$ is increased or α is decreased. The difference is favorable and hence we may conclude that the modified weights function as desired. The effect remains even in the average path lengths that take into account the inter-vertex Euclidean distance, as shown in Figure 5.10.

We wanted further insight into the distances, and plotted the diameters, unweighted and weighted, in Figure 5.11. As expected, in the hop-based diameter, the LHTs have lower diameter than the MSTs; the LHTs are able to reduce the diameter of the tree, although only the MHT-approximate had diameter close to the diameter of the graph. The MSTs are more path-like in structure with large diameter; notice the logarithmic scale on the y-axis of the diameter plot. The differences become smoother when the edge weights are taken into account in calculating the maximum distance, but the ordering of the curves stays the same: the gain of the LHTs over the MSTs remains, but is small, whereas the graph has a much smaller weighted diameter than any of the trees, which is to be expected for uniform random networks of densities as high as the ones used in the experiments.

In conclusion, utilizing a degree-dependent modification on the edge weights of a graph in Euclidean space helps to construct trees that have on average fewer intermediate vertices on communication paths, empowering savings in total transmission costs. The method is tunable to take into account the expenses of transmitting over different ranges. The experiments on uniform random graphs are a “baseline setup” where the graph itself does not have an underlying scale-free structure, and hence seeing the method work for such graphs indicates that the advantages would be even stronger in applications where the network topology itself contains hubs. As this is the case in numerous natural setups, we hope the method to be applicable as one component in designing routing solutions for practical problems involving radio communication or other setups where the communication cost has a initial component and a distance-based component.

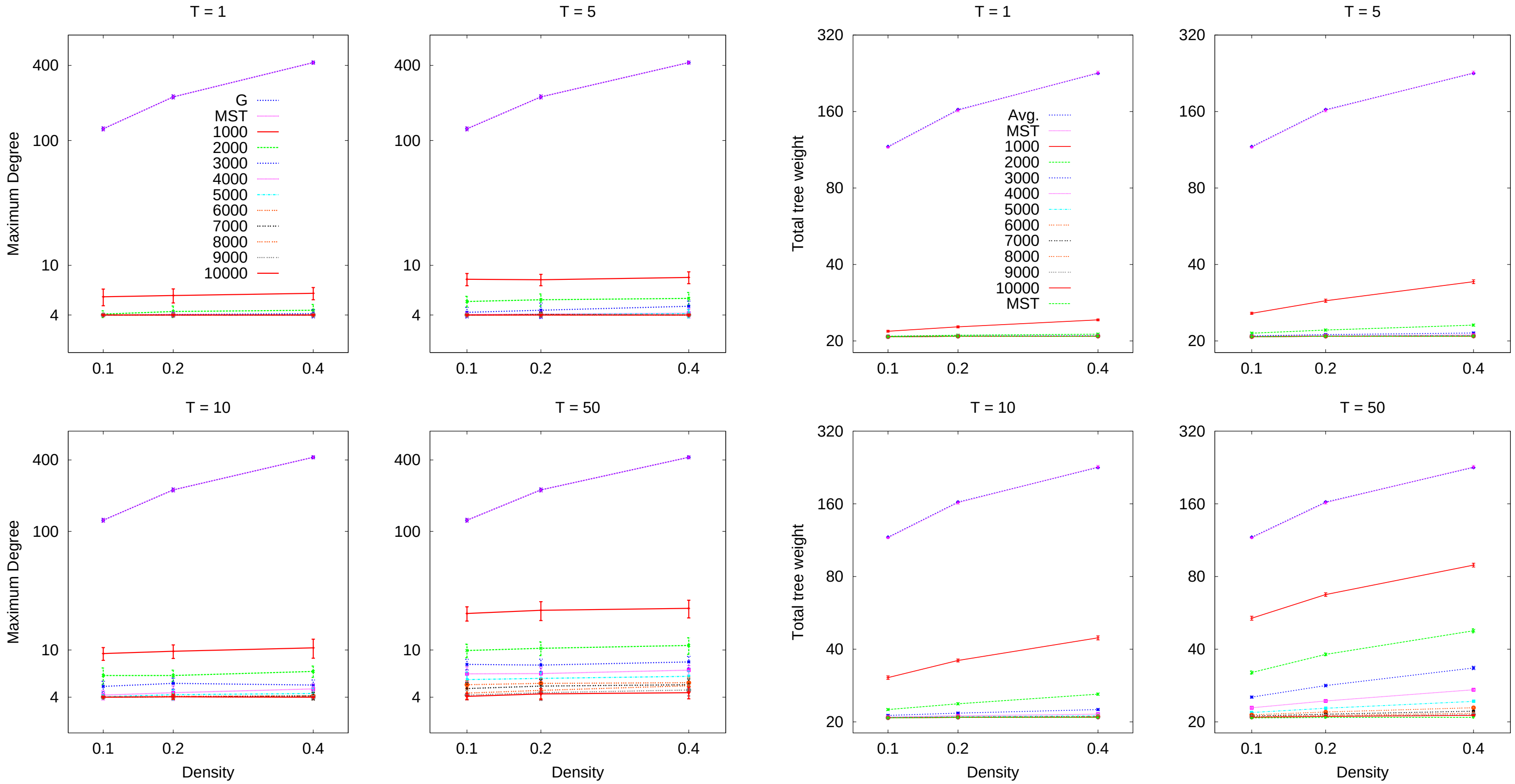


Figure 5.9: On the left, the maximum degree Δ in the generated graphs and the low-hop trees (LHT); note that MHT and the graph have the same Δ by construction. All MSTs had maximum degree of four. On the right, the total weights of the spanning trees — note that as MST optimizes this measure, the value given for those trees is the absolute minimum achievable for any spanning tree. All values shown are averages over the 30-graph sets with the same order and density; the standard deviation (small) is shown as error bars on the y-axis. For comparison, we calculated for each graph instance the average edge weight, estimated from that the average tree weight simply by multiplying the average edge weight by $n-1$; averages of this quantity over the 30 instances are shown in the curve titled “Avg.” — the curve for MHT practically overlaps with the average weight.

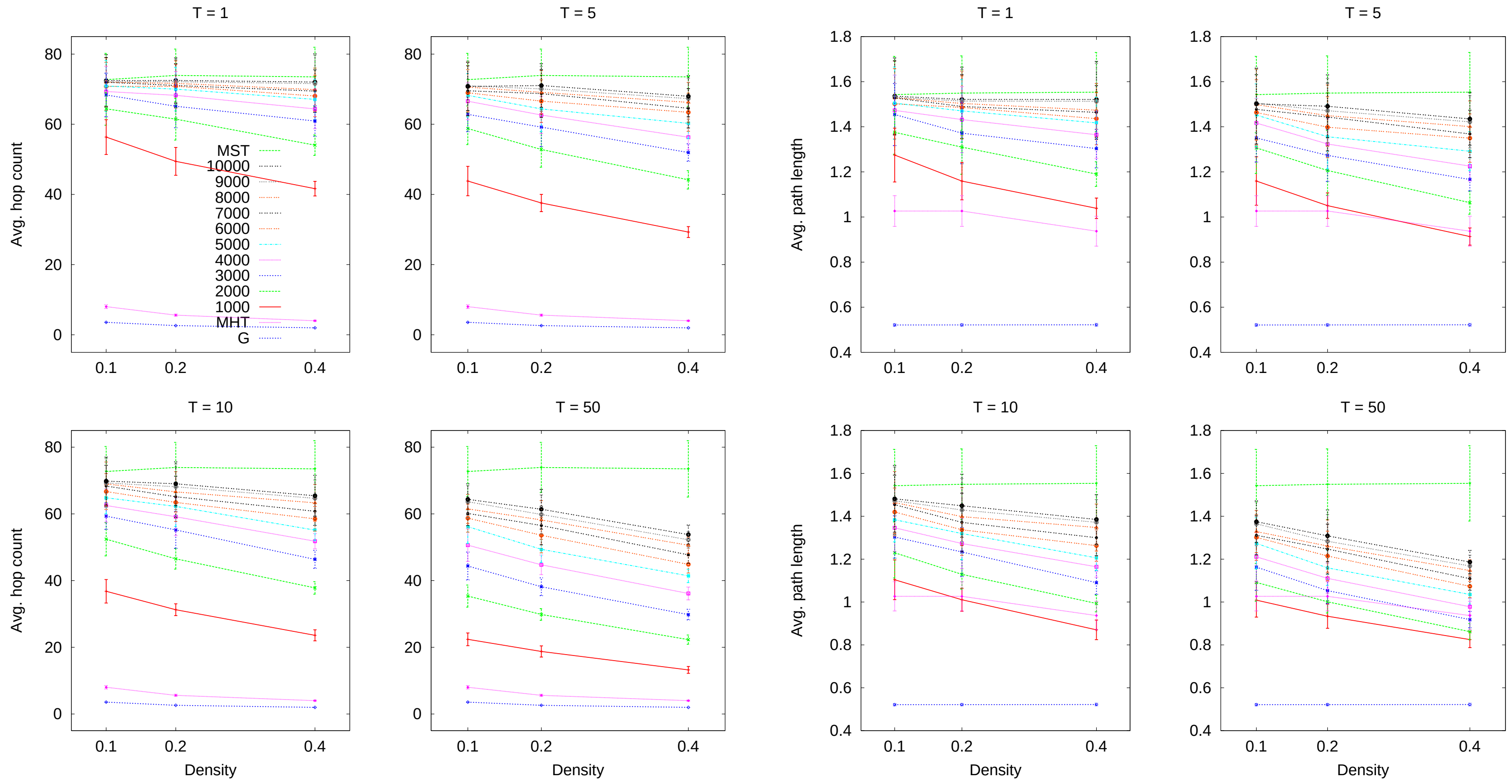


Figure 5.10: On the left, the average hop-length for the MST set and the different-parameter LHT sets, as well as the original graphs. On the right, the average path length for the MST set and the different-parameter LHT sets, as well as the approximate MHT set and the original graphs. The values are averages over the 30-graphs sets with the same order and density; the standard deviation is shown as error bars on the y-axis. The legend is the same for all plots, with the MST curve being the topmost one and the graph curve the lowest.

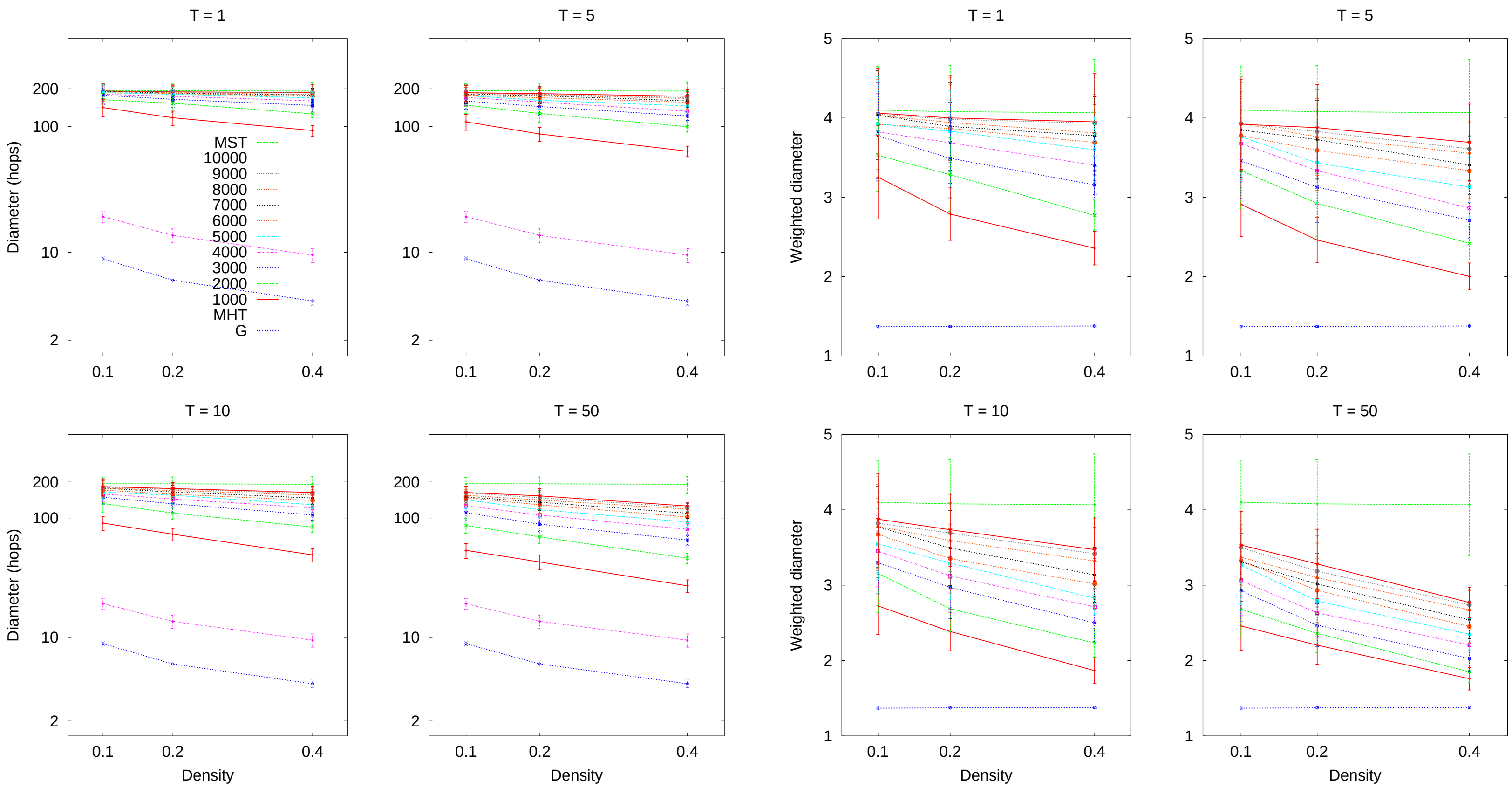


Figure 5.11: On the left, the diameter (i.e., the maximum hop-length) for the MST set and the different-parameter LHT sets, as well as the original graphs. The values are averages over the 30-graphs sets with the same order and density; the standard deviation is shown as error bars on the y-axis. The MST diameter is naturally the largest and the graph diameter the lowest. On the right, the weighted diameter (i.e., the the maximum weighted path length) for the MST set and the different-parameter LHT sets, as well as the original graphs. The same legend applies to all plots.

6 GRAPH DATA MINING

It has been observed that in natural networks, certain subgraphs occur with a much higher frequency than they do in uniform random graphs [225, 240]. In *graph data mining* [312], one fundamental problem is the identification of such *frequent* patterns, i.e., subgraphs that occur repeatedly in the structure of an input graph. Basic graph data mining algorithms enumerate all subgraphs and list the most frequent ones [160, 199, 200, 244].

Aiming for efficient and localizable graph data mining for massive natural graphs, we propose a method for finding (approximate) matches to a given pattern graph P in an input graph G instead of obtaining a complete listing of all frequent subgraphs. This approach is fitted to scenarios where it is of interest to know whether some substructure appears or not and in what form, such as hypothesis verification or examining whether a substructure that has been found to be a frequent pattern in one setting is present in another data set.

The common approach for mining frequent subgraphs is to start with a complete enumeration of small subgraphs, calculate how many times each subgraph-pattern occurs, and then iteratively grow subgraphs that have *minimum support*. In data mining, the term *support* refers to the number of times a pattern appears in a database, and in graph data mining, it is the number of times a subgraph appears in a given graph or a set of graphs. Hence minimum support means that the frequency of appearances of a pattern is above a predetermined threshold.

Apriori-based algorithms generate subgraphs starting from very small "seed subgraphs" (usually formed by single edges and the end-point vertices) and iteratively combine currently stored frequent subgraphs to construct larger patterns, keeping track of the frequency of each pattern [143]. Such algorithms include Apriori-Based Graph Mining (AGM) [160] and FSG [199]. Approaches designed to avoid generation of duplicates in the subgraph enumeration process include GSPAN [323] and CLOSEGRAPH [324] that employ *canonical labelings*[1] (namely a DFS code [323]) and lexicographic ordering of subgraphs. The CLOSEGRAPH algorithm mines for closed patterns, that is, subgraphs for which no superpattern exists with the same support.

Heuristic, greedy algorithms include SUBDUE [153] that aims to compress a given input graph using the minimum description length principle combined with subgraph frequency. Heuristic graph data mining methods do not in general examine all subgraphs in order to save computation time, but instead greedily choose to examine only those subgraphs that appear to be promising patterns in some sense.

Our focus is on studying whether a given graph G contains subgraphs *similar* to a given *pattern* graph P; this problem includes as a subproblem defining graph similarity. The strictest form of similarity is *graph isomorphism*,

[1] In settings where combinatorial objects are generated and the same structure may be generated multiple times, duplicates can be eliminated by computing a *canonical label* for each item, such that two objects are assigned the same label if and only if they are structurally identical.

already discussed in Chapter 2. There are two important "constructional" problems related to (sub-)graph isomorphism problems:

(i) the *maximum common subgraph* problem [53, 115] (given two graphs G and P, construct a graph G' of *maximum* order and size such that both G and P contain a subgraph isomorphic to G') and

(ii) the *minimum common supergraph* problem [54, 115] (given two graphs G and P, construct a graph G' of *minimum* order and size s.t. G' contains a subgraph isomorphic to G and a subgraph isomorphic to P).

For the latter problem, the two subgraphs are likely to overlap in case that G and P are similar, and in the former problem, the order and size of G' should be close to those of G and P in the case that the two input graphs have similar structure. Similarity metrics can be defined based on both of these constructions, maximum common subgraph appearing to be more widely used [55, 151]. Both problems are evidently in **NP**; also, solving either one solves the graph isomorphism problem, as (only) for isomorphic graphs the construction is of equal order and size as the graphs themselves.

The most popular variants of graph similarity measures include the *edit distance* (also widely known as *script distance* and *Levenshtein distance*) [142], which is the minimum number of *operations* needed to transform G into P, closely related to the above two transformation problems — originally edit distance was defined for strings, but as graphs have fluent representations as strings, the definition easily generalizes for graphs.

Defining edit distance is in many ways problematic, as one has to define the set of operations (i.e., whether to only allow vertex and edge additions and removals or also operations such as splitting vertices, joining vertices, contracting edges, and rewiring edge endpoints). Furthermore, in practical applications different operations may have different "costs"; for example the lack of a vertex may have more impact on practical similarity than the absence of an edge. For a specific application area for which a training data set exists, it is possible to automatically infer the operation costs [233], but such a setup is not always feasible. In addition, the similarity values obtained are highly sensitive to the selection of a cost function [151]. Edit distance is an **NP** complete problem, as it solves the graph isomorphism problem — the edit distance problem is known to be equivalent to the maximum common subgraph problem [51].

Another related problem is that of *graph matching* [52, 84, 152], where the goal is to find a mapping that might not necessarily be an isomorphism and may not even be bijective, but that in some sense relates the vertices of one graph to those of another to maximize some similarity measure. Less formal approaches to graph similarity include proposals of similarity evaluation in terms of measures of network nonuniformity [98], such as the measures discussed in Chapter 2.

As we are trying to avoid having to solve an **NP** complete problem each time we wish to evaluate the similarity of a candidate subgraph with a user-defined pattern, we will be returning to the graph matching scenario, but will not experiment with variants of the common sub- and supergraph problems.

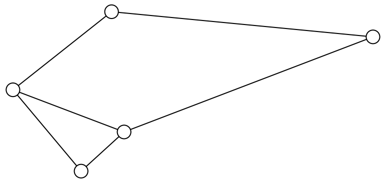
We begin by exploring computationally feasible approaches to determining subgraph similarity. A *bit-string representation* $\mathcal{B}(\mathbf{A})$ of a $n \times n$ adjacency matrix $\mathbf{A}$ is a bit string of length $N = n(n+1)/2$ that is composed of a concatenation of the *upper triangular matrix* of $\mathbf{A}$, i.e., the complete first row, the last $(n-1)$ elements of the second row, and so forth, as illustrated in Figure 6.1. If the graphs were known to be non-reflexive, the diagonal could be ignored, saving n bits. This encoding is essentially equivalent to the column-wise graph code used in AGM [160]. We do not use edge labels in the basic definition, but the encoding could easily be extended to hold edge label identifiers instead of binary values.

A *relabeling* of a graph $G = (V, E)$ is a bijection $\varphi : V \to V$. Each relabeling corresponds to a reordering of the adjacency matrix $\mathbf{A} = \mathbf{A}_G$. Each reordering of the matrix further defines a bit-string representation $\mathcal{B}(\mathbf{A})$.

Let $\mathcal{B}_{\varphi(G)}$ be the bit-string representation given relabeling φ. Given all possible relabelings of G, the *canonical* bit-string representation $\mathcal{B}(G)$ of G is the bit-string representation $\mathcal{B}$ is the *lexicographically largest* one over the set of possible relabelings, i.e., corresponds to the largest number when the bit string is interpreted as a binary number:

$$\mathcal{B}(G) = \max_{\varphi : V \to V} \left\{ \mathcal{B}_{\varphi(G)} \right\}. \tag{6.1}$$

If two graphs G^S and P have the same canonical label, they are *isomorphic*. Therefore computing the canonical label has the same complexity as graph isomorphism and is in **NP**. Our goal is to find a way to avoid having to compute canonical labels for all the subgraphs that are compared to the pattern during the search.



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

$\mathcal{B}(\mathbf{A}) = 011100100001010$

Figure 6.1: A graph G of order five, an adjacency matrix $\mathbf{A} = \mathbf{A}_G$ with the entries that form the bit string in boldface, and the corresponding bit-string representation $\mathcal{B}(\mathbf{A})$.

The *weight* $\omega(\mathcal{B})$ of a bit string $\mathcal{B}$ is the number of one bits it has. For a graph $G = (V, E)$ and $\mathbf{A} = \mathbf{A}_G$, the weight is the edge count of the graph, $\omega(\mathcal{B}(\mathbf{A})) = |E|$, as each edge corresponds to two one-bits of the adjacency matrix, only one of which lies in the upper triangular matrix.

The *Hamming distance* of two bit strings is the number of positions at which the two strings differ. Similarly, for binary data, the *exclusive or* (XOR) $\mathcal{B}_1 \oplus \mathcal{B}_2$ of two bit strings $\mathcal{B}_1$ and $\mathcal{B}_2$ of the same length N is a bit string $\mathcal{B}_3$ in which the bit $b_{\mathcal{B}_3}(p)$ at position p is one *if and only if* the bits at position p in $\mathcal{B}_1$ and $\mathcal{B}_2$ *differ*:

$$b_{\mathcal{B}_3}(p) = \begin{cases} 1, & \text{if } b_{\mathcal{B}_1}(p) \neq b_{\mathcal{B}_2}(p), \\ 0, & \text{otherwise.} \end{cases} \tag{6.2}$$

Hence, $\omega(\mathcal{B}_1 \oplus \mathcal{B}_2)$ is the number of bit positions in which the two bit strings differ. The XOR-*difference* $\delta_\oplus(\mathcal{B}_1, \mathcal{B}_2) \in [0, 1]$ of two bit strings $\mathcal{B}_1$ and $\mathcal{B}_2$ of length N is defined as

$$\delta_\oplus(\mathcal{B}_1, \mathcal{B}_2) = \frac{\omega(\mathcal{B}_1 \oplus \mathcal{B}_2)}{N}. \tag{6.3}$$

The measure, known as normalized Hamming distance in general form, assigns value zero for identical strings and one for complement strings, as desirable. The definition does not take into account the *positions* at which the two strings differ. In some cases, the most significant bits[2] are truly more important, and other difference measures work better in estimating how drastically two given bit strings differ. For the purpose of defining difference functions sensitive to bit positions $b_i(\mathcal{B})$, we define an indicator function over the bit positions of two bit strings $\mathcal{B}_1$ and $\mathcal{B}_2$ as

$$\mathcal{I}(i) = \begin{cases} 1, & \text{if } b_i(\mathcal{B}_1) = b_i(\mathcal{B}_2), \\ 0, & \text{if } b_i(\mathcal{B}_1) \neq b_i(\mathcal{B}_2). \end{cases} \tag{6.4}$$

Using this definition, we denote the first position in which two bit strings differ, starting from the most significant bit at position N going down to the lowest bit at position 1, by

$$b_{\text{pos}}(\mathcal{B}_1, \mathcal{B}_2) = \max_{i \in [N]} \{\mathcal{I}(i) = 0\} \tag{6.5}$$

and set $b_{\text{pos}}(\mathcal{B}_1, \mathcal{B}_2) = 0$ for $\mathcal{B}_1 = \mathcal{B}_2$. We define a *position-sensitive* difference measure

$$\delta_{\text{pos}}(\mathcal{B}_1, \mathcal{B}_2) = \frac{b_{\text{pos}}(\mathcal{B}_1, \mathcal{B}_2)}{N} \tag{6.6}$$

that assigns the value zero for equal strings and one for pairs that differ in the first position. This measure heavily punishes a difference in an early position even if all the remaining bits are equal. A smoother possibility for emphasizing the importance of the most significant bits is to define a *weighted* difference measure, such as

$$\delta_w(\mathcal{B}_1, \mathcal{B}_2) = 1 - \frac{\sum_{i=1}^{N} (\mathcal{I}(i) \cdot i)}{\sum_{i=1}^{N} i} = 1 - \frac{2\sum_{i=1}^{N} (\mathcal{I}(i) \cdot i)}{N(N+1)}. \tag{6.7}$$

The subtraction from one is included so that a zero difference would correspond to equal strings, as in the above two definitions.

Yet another option is to calculate the *longest common subsequence* (LCS) [142] of the two strings, which is a sequence of characters of maximum length over all sequences that appear as subsequences in both of the strings under comparison. We define a similarity measure by computing the length of the longest common subsequence and normalizing it to the interval [0, 1] by dividing by the string length (in case of unequal-length strings, the minimum length, which is an upper bound to the length of any common subsequence).

[2] In binary numbers, the left-most bit that corresponds to the highest power of two is known as the *most significant bit* as its effect on the numerical value is higher than that of the other bits.

If and only if the two strings are equal, the similarity value is one. In a general setting also a zero value would be possible, but for bit strings it only occurs when one strings has only zeroes and the other only ones. We refer to this similarity measure as LCS in this chapter. The computation of LCS is worst-case quadratic in the string length and can be done efficiently by a dynamic programming algorithm.

6.1 SUBGRAPH SIMILARITY

Given a *connected* pattern graph[3] $P = (V_P, E_P)$ and a subgraph $G^S = (S, F)$ of $G = (V, E)$ such that $|V_P| = |S| = k$, we define the *similarity* of a subgraph to the pattern through their bit-string representations. We want the similarity to be one when the two graphs are isomorphic and zero when the two graphs disagree completely (that is, one is a clique and another is an independent set).

Over all possible relabelings φ of G^S, the one that minimizes a difference measure $\delta_* \left(\mathcal{B}_{\varphi(G^S)}, \mathcal{B}(P)\right)$ gives the similarity of the subgraph G^S and the pattern P. Note that for zero difference under any of the proposed difference measures, $\mathcal{B}_{\varphi(G^S)}$ corresponds exactly to $\mathbf{A}_P$ and hence the two graphs G^S and P are isomorphic. Hence we define the exact similarity of a subgraph and a pattern to be determined by the lowest difference over all possible relabelings of the subgraph,

$$\rho\left(G^S, P\right) = 1 - \min_{\varphi: S \to S} \left\{ \delta_* \left(\mathcal{B}_{\varphi(G^S)}, \mathcal{B}(P)\right) \right\}. \tag{6.8}$$

This is evidently computationally demanding (as it solves the isomorphism problem), although all the difference measures themselves are simple to compute for a given pair of bit strings. Trying out all possible relabelings of vertices for a graph G is exponential in n as there are $n!$ orderings. A naïve algorithm to obtain the k lexicographically largest bit-string representations $\mathcal{B}(G)$ for a given graph $G = (V, E)$ simply generates all these permutations of the adjacency matrix. Such a search can be pruned by comparing the bit string to the largest one found thus far *during* its generation and discarding it as soon as a bit is appended that makes it lexicographically smaller.

With such a procedure, we may also store k of the lexicographically largest bit strings instead of storing just the canonical one. This is clearly a computationally infeasible approach for large graphs or for frequent repetition, but in the pattern-matching scenario, the pattern can be assumed to be of much smaller order than the graph G in which it is to be matched. Therefore obtaining a such a set of lexicographically large bit strings for the pattern is not a dominating factor in the runtime of a matching procedure. Hence we assume that it is feasible to compute k largest bit strings for $\mathcal{B}(P)_1, \ldots, \mathcal{B}(P)_k$ for the pattern graph P. Let $\mathcal{B}(P) = \mathcal{B}(P)_1$.

In order to avoid having to directly minimize Equation 6.8, we concentrate on finding a lexicographically large bit string for G^S, as $\mathcal{B}(P)$ is assumed to be known. We cannot compute the canonical labels for the subgraphs, as

[3] Should one wish to mine for *disconnected* patterns, the procedure presented here could be repeated for each connected component separately.

numerous subgraphs need to be examined while matching a pattern in a large graph. We use the following notation. Given a subset $S \subseteq V$, we denote the number of neighbors a vertex v has in S by

$$\deg_S(v) = |S \cap \Gamma(v)|. \tag{6.9}$$

For constructing a bit-string representation for a subgraph encountered during the pattern matching, we propose a greedy, heuristic algorithm (given as pseudo code in Table 6.1) that outputs the vertices of an input graph in such an order that produces a lexicographically high bit string when the adjacency matrix of the graph is sorted into that order from the initial ordering based on vertex labels $1, 2, \ldots$.

The algorithm prioritizes outputting reflexive vertices over non-reflexive ones. This is useful as the diagonal elements of the adjacency matrix are the leftmost elements of each row included in the bit string. The reflexive neighbors of an already output vertex are always output before reflexive non-neighbors and the non-reflexive neighbors are output before the non-reflexive non-neighbors. The algorithm attempts to order the vertices such that as many ones as possible would be placed as early as possible in the bit-string representation, preferably avoiding intermediate zeroes.

We assume the input to be given in the form of adjacency lists; the computational complexity of each step is given in the rightmost column of Table 6.1; assignments involving sets are considered to be worst-case linear in the number of elements. The algorithm necessarily terminates: it outputs one vertex at step 5 of each repetition of the loop between steps 5 and 14, and each vertex is output exactly once, implying that the loop is processed exactly n times.

In order to compute an approximate similarity for a given subgraph G^S in G with a pattern P, we compute for the pattern k lexicographically first bit strings $\mathcal{B}(P)_1, \ldots, \mathcal{B}(P)_k$ and calculate the similarities using each $\mathcal{B}(P)_i$ and the *approximate* representative bit string of the subgraph G^S obtained by the algorithm of Table 6.1. This gives us a lower bound of $\rho\left(G^S, P\right)$ as

$$\rho\left(G^S, P\right) \geq \max_{j \in [k]} \left\{1 - \delta_*\left(\mathcal{B}\left(G^S\right), \mathcal{B}(P)_j\right)\right\}. \tag{6.10}$$

If the subgraph and pattern are identical, using more than one representative bit string for the pattern increases the likelihood that the greedily constructed representative of the subgraph matches one of the pattern's representatives.

We conducted a small-scale experiment to study how similar the greedily constructed bit strings are to the top k bit strings of the same input graph found by an exhaustive search. The limited scope of the experiment follows from the combinatorial explosion in the running time of the exhaustive search: it becomes slow already for fewer than a dozen vertices. The greedy algorithm, however, performs well for graphs of much larger order.

For an illustration on the run times, averaged over fifty graphs output by different graph generators, see Figure 6.2. We used the three graph generation models: the $\mathcal{G}_{n,p}$ model [133], a variation of the Watts-Strogatz model that overlays a $\mathcal{G}_{n,p}$-graph on the circulant lattice [242, 243], and the Barabási-Albert generation method for scale-free graphs [25]; implementational

Table 6.1: A greedy, heuristic algorithm that outputs vertices of a given *connected* graph $G^S = (S, F)$ in the order that gives a lexicographically large bit-string representation when used to sort the columns of an adjacency matrix. L is the set of adjacency lists of the vertices.

	Main procedure obtaining as input $L = \{\Gamma(v)\}_{v \in S}$	$\mathcal{O}\left(n^5\right)$
1	$v :=$ maximum-degree reflexive vertex $\in S$, non-reflexive if none.	$\mathcal{O}\left(n^2\right)$
2	$\ell := 0$.	$\mathcal{O}(1)$
3	$W(v) := \ell$.	$\mathcal{O}(1)$
4	$\ell := \ell + 1$.	$\mathcal{O}(1)$
5	**Output** v.	$\mathcal{O}(1)$
6	$c := W(v)$.	$\mathcal{O}(1)$
7	$W(v) :=$ *invalid*.	$\mathcal{O}(1)$
8	**For each** $w \in \Gamma(v)$	$\times\mathcal{O}(n)$
9	**If** $W(w)$ is undefined,	$\mathcal{O}(1)$
10	**then** $W(w) := \ell$.	$\mathcal{O}(1)$
11	**If** at least for one vertex w, $W(w)$ was set,	$\mathcal{O}(1)$
12	**then** $\ell := \ell + 1$.	$\mathcal{O}(1)$
13	$v :=$ SelectVertex $(c, W(\cdot), \ell, L)$.	$\mathcal{O}\left(n^4\right)$
14	**If** v is defined, **then go to** line 5.	
15	**Exit**.	
	Subroutine SelectVertex *to pick* v *s.t.* $W(v) \geq c$, *given* L *and* ℓ.	$\mathcal{O}\left(n^4\right)$
1	$C := \emptyset$.	$\mathcal{O}(1)$
2	$j := c$.	$\mathcal{O}(1)$
3	**While** $\left((C = \emptyset) \wedge (j \leq \ell)\right)$:	$\times\mathcal{O}(n)$
4	$C := \{w \mid W(w) = j\}$.	$\mathcal{O}(n)$
5	If $C \neq \emptyset$, **then break**.	$\mathcal{O}(1)$
6	**Else** $j := j + 1$.	$\mathcal{O}(1)$
7	**If** $(C = \emptyset)$, **then return** *undefined*.	$\mathcal{O}(1)$
8	**If** $(\lvert C\rvert = 1)$, **then return** the included vertex.	$\mathcal{O}(1)$
9	$C' := \{w \in C \mid w \in \Gamma(w)\}$.	$\mathcal{O}\left(n^2\right)$
10	**If** $(\lvert C'\rvert = 1)$, **then return** the included vertex.	$\mathcal{O}(1)$
11	**If** $(C' \neq \emptyset)$, **then** $C := C'$	$\mathcal{O}(n)$
12	**Repeat** until a *break* occurs:	$\times\mathcal{O}(n)$
13	**If** $(j > \ell)$, $j :=$ *stop*.	$\mathcal{O}(1)$
14	$S := \{u \mid W(u) = j\}$.	$\mathcal{O}(n)$
15	$C' := \{w \in C \mid \deg_S(w)$ is maximal$\}$.	$\mathcal{O}\left(n^2\right)$
16	$C := C'$.	$\mathcal{O}(n)$
17	**If** $(\lvert C\rvert = 1)$, **then return** the included vertex.	$\mathcal{O}(1)$
18	**If** $((j = \mathit{stop}) \vee (j \neq \mathit{invalid}))$, **then break**.	$\mathcal{O}(1)$
19	**Return** the first element in C.	$\mathcal{O}(1)$

details of the constructions are available in our previous work [306]. After generating the graphs, we added reflexive edges independently for each vertex with probability p. This was done because none of the model implementations incorporate reflexive edges, which however are often found in application areas of subgraph data mining. The model parameters and resulting average edge counts are given in Table 6.2.

Table 6.2: The parameters used in generation of the test graphs for the two test sets and the resulting average edge counts of the graphs. The probability p for added reflexive edges was 0.3 for smaller graphs and 0.1 for the larger ones; the expected number of reflexive edges $p \cdot n$ is included in all the expected values of the table.

Small graphs, $n \in [5,\ 11]$		
Model	Parameters	Expected edge count
$\mathcal{G}_{n,p}$	$p_e = 0.7$	$\mathrm{E}[m] = 0.7\binom{n}{2} + 0.3n$
WS	$k = 1$, $p_e = 0.3$	$\mathrm{E}[m] = n + 0.3\binom{n}{2}$
BA	$n_0 = 3$, $\deg_0(v) = 2$	$\mathrm{E}[m] \approx 2.3n$
Larger graphs, $n \in [15, 200]$		
Model	Parameters	Expected edge count
$\mathcal{G}_{n,p}$	$p_e = 0.4$	$\mathrm{E}[m] = 0.4\binom{n}{2} + 0.1n$
WS	$k = 3$, $p_e = 0.2$	$\mathrm{E}[m] = 3n + 0.2\big(\binom{n}{2} - 3n\big) + 0.1n$
BA	$n_0 = 5$, $\deg_0(v) = 3$	$\mathrm{E}[m] \approx 3.1n$

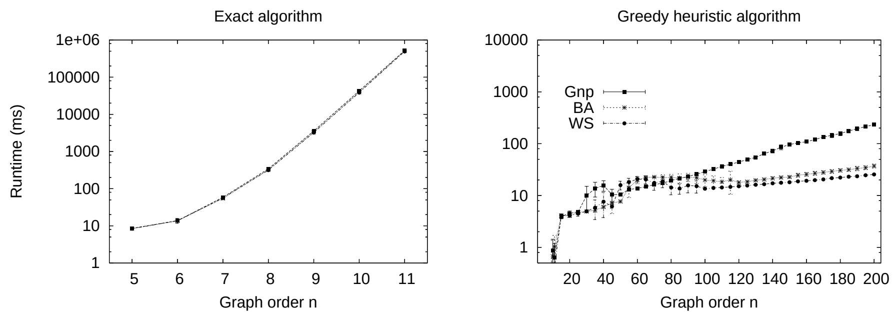


Figure 6.2: On the left, the average running times of exact (left plot) and approximate (right plot) computation of canonical bit-strings for input graphs of growing order. The algorithms were ran for three different graph generation models and the running times were averaged over 30 independently generated instances for graphs of order $n \in [5,\ 11]$. Note the change of magnitude on the y-axis of the plots. The standard deviation is plotted with error bars for all data points, but these are so small that they only show in some of the running times of the greedy algorithm.

As expected, the running time of the naïve exhaustive search does not differ for the models as graph structure plays no part in the way the algorithm proceeds; hence all three curves overlap perfectly with no deviations. However, the different models behave differently for the greedy heuristic both with respect to running time and accuracy. For graphs of order seven, the exact algorithm took approximately 40 seconds, whereas those of order eight already took more than eight minutes per repetition, and hence the complete test set of three models with 30 repetitions each required more than twelve hours. The full enumeration of the permutations *alone* (using a standard algorithm to advance from a given permutation to the next [193]) takes for about a second for eleven elements on the same machine, but for twelve elements it needs almost 20 seconds, and for 13 elements almost four minutes. Therefore the algorithm is clearly not appropriate for frequent repetition for graphs with as few as a dozen vertices and will not be at all applicable for larger graphs.

We used test sets for larger orders from 15 up to 200 with increments of five for the approximation algorithm, in addition to those on which the exact algorithm was tested. For these graphs, the exhaustive search is evidently impractically slow and hence was not attempted. The approximation however scales well for graphs of considerable order. Therefore, if the canonical label of a pattern of interest can be obtained, our method would allow for mining rather large subgraphs. We conjecture that a method in which the approximate bit strings are used for both the pattern and the subgraphs could prove useful for some applications and hope to study this in future work.

In order to examine how well the greedy heuristic performs, we checked if the greedy bit string was among the $k = 2n$ lexicographically first bit strings for those graphs for which the exact algorithm was ran, and if so, at which index. Figure 6.6 on page 132 shows the results of this study in the form of a histogram. The results are not surprising; for small graphs, we recognize isomorphism easily, but as the graphs grow, the greedy string hardly ever appears within the k largest bit strings. The $\mathcal{G}_{n,p}$ graphs are the hardest, which is not surprising, as there is no structure or nonuniformity to take advantage of. Similarly the most regular of the graph models, the Watts-Strogatz lattice with additional random edges, has the greedy string most frequently included in the set of exact strings.

It is worth remembering that there are in total $n!$ bit strings in the complete lexicographic ordering of a given graph of order n, some of which may be identical if the graph structure has symmetries. For example the symmetry of the circulant lattice of the Watts-Strogatz small-world model causes a high probability for different orderings of the vertices to produce identical and/or highly similar bit strings. The bit-string length $n(n+1)/2$ grows quadratically with n, making the relative importance of a single-bit difference inversely quadratically smaller in calculating the similarities.

In addition we calculated the similarity values between the greedy string and all k lexicographically first strings; Figure 6.3 shows the average similarities and standard deviations over the test sets. It can be easily seen that the first different bit position arrives early, and hence $\delta_{\text{pos}}(\mathcal{B}_1, \mathcal{B}_2)$ of Equation 6.6 gets low values. The LCS similarity maintains on average the highest values, but it is a little slower to compute than the others, as the other measures are linear and LCS is quadratic in the length of the string.

In order to interpret Figure 6.3, it is of interest to know what is the expected value of each similarity measure given two *random* bit strings of length $N = n(n+1)/2$ both corresponding to a graph with order n and size m. Denote the probability that two bit strings $\mathcal{B}_1$ and $\mathcal{B}_2$, both of length N with exactly m ones, *first* differ at position k by $q(m, N, k)$; the corresponding formula is given along with a derivation in an appendix starting on page 175 (Equation 7). The expected values of the similarity under the different difference measures are shown in Table 6.3 — it is noteworthy that the XOR-difference and the weighted difference have exactly the same expectation for two random strings with the same number of ones, as the symmetry in computing the expectation over all possible strings effectively "cancels out" the effect of the weights.

Table 6.3: The expected value and the minimum value of the similarity $\mathrm{E}[\rho(\mathcal{B}_1, \mathcal{B}_2)]$ for two bit strings that both correspond to graphs with m edges and n vertices (yielding bit strings with $N = n(n+1)/2$ bits, out of which m are ones) when using the three difference measures $\mathrm{E}[\rho(\mathcal{B}_1, \mathcal{B}_2)]$. Note that $\delta_\oplus(\mathcal{B}_1, \mathcal{B}_2)$ and $\delta_w(\mathcal{B}_1, \mathcal{B}_2)$ give the same expected similarity. We denote $u = \max\{m,\ N-m\}$ and $\ell = \min\{m,\ N-m\}$. For the LCS similarity, the lower bound for similarity using this notation is simply $\frac{u}{N}$.

Measure	$\mathrm{E}[\rho(\mathcal{B}_1, \mathcal{B}_2)]$	Lower bound
$\delta_\oplus(\mathcal{B}_1, \mathcal{B}_2)$	$\dfrac{\mathrm{E}[\omega(\mathcal{B}_1 \oplus \mathcal{B}_2)]}{N} = \dfrac{2m(N-m)}{N^2}$	$\frac{u-\ell}{N}$
$b_{\mathrm{pos}}(\mathcal{B}_1, \mathcal{B}_2)$	$1 - \dfrac{\mathrm{E}\left[b_{\mathrm{pos}}(\mathcal{B}_1, \mathcal{B}_2)\right]}{N} = 1 - \dfrac{\sum_{i=1}^{N}(N-i+1) \cdot \prod_{j=0}^{i-1} q(m, N, j)}{N}$	$\frac{u-\ell}{N}$
$\delta_w(\mathcal{B}_1, \mathcal{B}_2)$	$1 - \mathrm{E}[\delta_w(\mathcal{B}_1, \mathcal{B}_2)] = \frac{2m(N-m)}{N^2}$	$\frac{(u-\ell)(u-\ell+1)}{N}$

The plots of the expected similarities in Figure 6.4 show that for dense or sparse graphs, even a random pair achieves high similarity. For random pairs with density not at an extreme value, the XOR and weighted similarities have expected values lower than those shown in Figures 6.3 achieved by comparing a greedy string of an exact match. Hence the measure exhibits capability of distinguishing true matches from non-matching pairs. For the LCS similarity on a pair of binary strings with the same number of ones and zeroes in both strings, the value is *never* below 0.5; also, the average values for random strings are rather high, which means that in thresholding the LCS similarity, a rather high value may still imply dissimilarity of the input graphs — this could be partially resolved by scaling the minimum from 0.5 to zero, hence amplifying the differences between similarities that should be considered matches and those that should not be considered matches. With this observation in mind, the weighted similarity measure is easier to use.

We also wanted to see what values would partial matches give, i.e., graphs where some edges are absent or where additional edges are included. For this purpose, we generated another test set using the same parameters than in the accuracy tests (given in Table 6.2), but for each graph, also calculating a greedy string for modified graphs with one to five edges missing and with one to five additional edges. We ran the experiment for graphs of orders in $\{8, 9, 10\}$ and with $2n$ exact strings, but only the results for $n = 9$ using the maximum similarity among the set of k strings (the approximation of Equation 6.10) are shown.

For visualization of how the three difference functions and the LCS similarity function behave when computing the maximum similarities over the set of k strings of original graphs of order $n = 9$ and greedy strings generated for modified graphs, we give box-whiskers plots[4] of the values in Figure 6.5. It can be seen that the simple exclusive-or difference $\delta_{\oplus}(\mathcal{B}_1, \mathcal{B}_2)$ does not drop very clearly when the graph is modified, even though that would be desirable. The bit-position difference measure $\delta_{\text{pos}}(\mathcal{B}_1, \mathcal{B}_2)$ suffers from high variability, but the difference of the good matches to the modified ones is more clearly visible. All in all, the weighted difference and the LCS difference appear to be the most useful ones.

The weighted measure $\delta_w(\mathcal{B}_1, \mathcal{B}_2)$ succeeds in giving on average higher values to graphs that are more similar to the pattern with respect to their connection patterns, and the similarity values decrease as more differences are introduced. The LCS similarity measure also shows the decrease as modifications are introduced, but it falls somewhat slower as modifications are made and hence may be harder to adjust when choosing which threshold to use to filter the matches. In conclusion, we choose to use the combination of the weighted measure and the greedy strategy for determining whether or not a given subgraph resembles a given pattern or not, expecting values near 0.8 or above for matches and near 0.5 or below for non-matches as seen from the experiments on similar versus random pairs; for cases where only small deviation from the input pattern is tolerated, we recommend using the LCS similarity.

Generalization to directed graphs only requires the use of the full adjacency matrix instead of the upper-diagonal part in defining a bit-string representation. For a weighted graph, instead of binary values the weights may be put in an array to produce a string-like representation, but as searching for the exact weights would be restrictive, we recommend defining a table of prototype weights such that an edge weight in a certain range gets mapped into a certain prototype weight, and the differences of the weights are used in computing the difference, such that the normalization is done with the string length multiplied by the maximum weight-difference possible.

Given a graph $G = (V, E)$ of order n and a pattern graph $P = (V_P, E_P)$ of order $k \leq n$, the task of interest is to find a set of subgraphs $G_1^S, \ldots, G_s^S$ in G such that $\rho\left(G_i^S, P\right) \geq \xi$ for a given threshold $\xi \in [0,\ 1]$.

If the search is not restricted to *perfect matches* only, modifications such as single or multiple edge removals or additions can be done on the *pattern*, with the search repeated for each modified pattern. This is necessary as having even a single edge, especially a reflexive one, present in a subgraph and not in the pattern, can have drastic effects on what are the lexicographically first bit-string representations. Also, it allows for a user interface where the user first defines a pattern for exact search and then, if the set of matches found is unsatisfactory, the user may either modify the pattern manually or run an automated search for mutations until a sufficient number of satisfying

[4] A *box-whiskers plot* is composed of a box that captures the samples that fall between the first and the third quartile with a horizontal line inside the box representing the median value, and a vertical line indicating the full range of the set of observations: If the set contains some obvious outliers, they are usually excluded from the calculation of the box-whiskers plot and drawn as dots further outside the vertical line representing the range [266].

matches is found or the user concludes that nothing similar to the pattern currently exists in the graph.

Enumerating and comparing all subgraphs of order k in G is time-consuming and inefficient. Alternatives are to either prune the search space or use a stochastic search method. Several algorithms exist for generating all maximal induced subgraphs with a given property, if the property itself meets some requirements, such as being hereditarity [80]. Methods using complete search are time consuming, especially for a graph where the structure changes often and no intermediate results of previous searches help in serving future pattern queries. For such scenarios, we propose instead an approximate search. For a stochastic, *local* search, a *neighborhood* over the subgraphs of order k and with a fixed number of edges in $G = (V, E)$ needs to be defined. A possible neighborhood definition is considering G_1^S and G_2^S neighbors if either their vertex sets differ by one or if their edge sets differ by one while the vertex set remains static. The field of bioinformatics could benefit from such an approach for studying massive data sets; we hope to experiment on local pattern mining on real-world data set in the future.

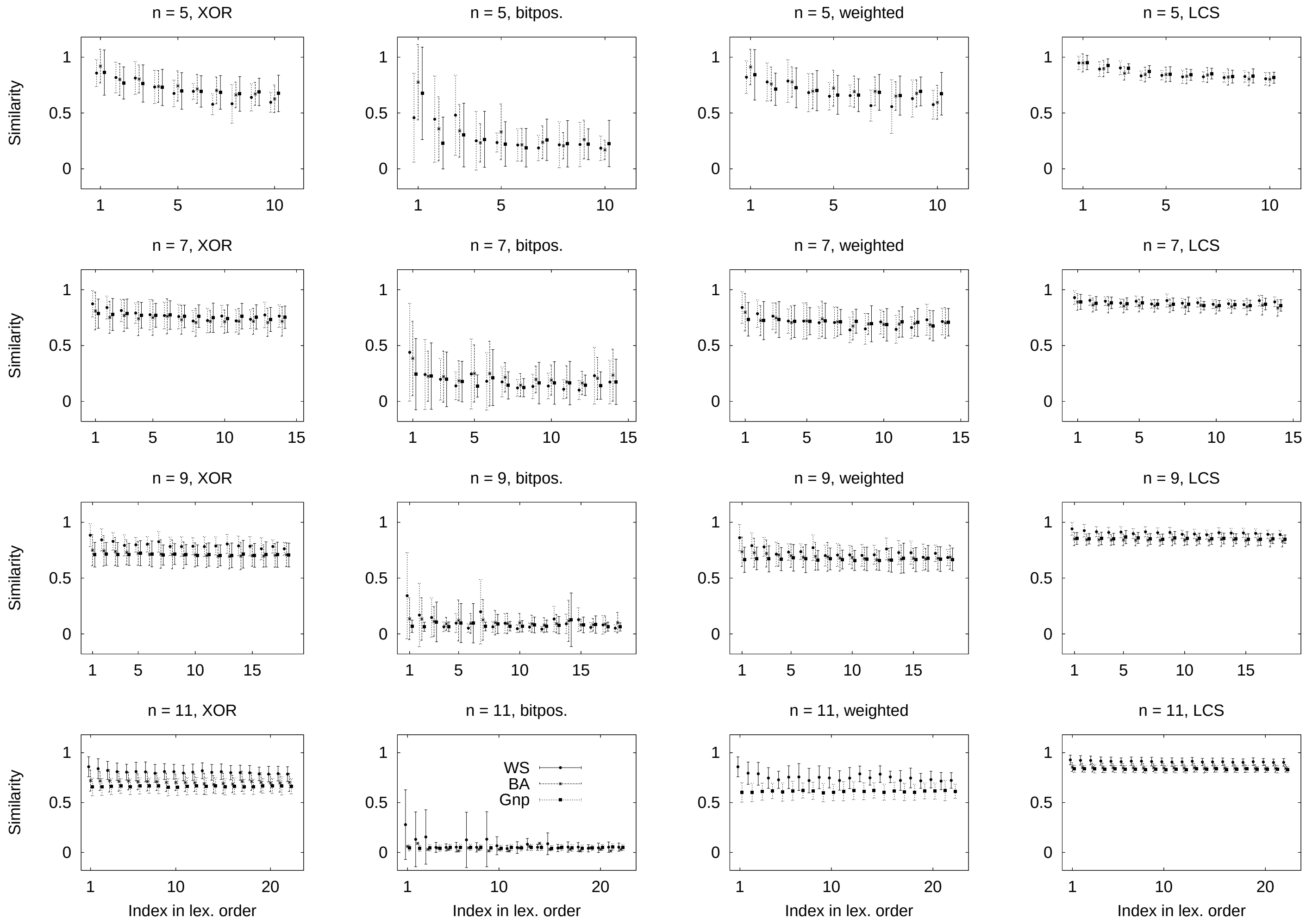


Figure 6.3: The average and standard deviation of the similarities between the greedy bit string and the $k = 2n$ lexicographically first bit strings for graphs of order $n \in \{5, 7, 9, 11\}$ for the three generation models used in the study. The first column shows the results for $\delta_{\oplus}(\mathcal{B}_1, \mathcal{B}_2)$ of Equation 6.3, the second column those of $\delta_{\text{pos}}(\mathcal{B}_1, \mathcal{B}_2)$ of Equation 6.6, the third column those of $\delta_w(\mathcal{B}_1, \mathcal{B}_2)$ of Equation 6.7, and the right-most column the LCS similarity.

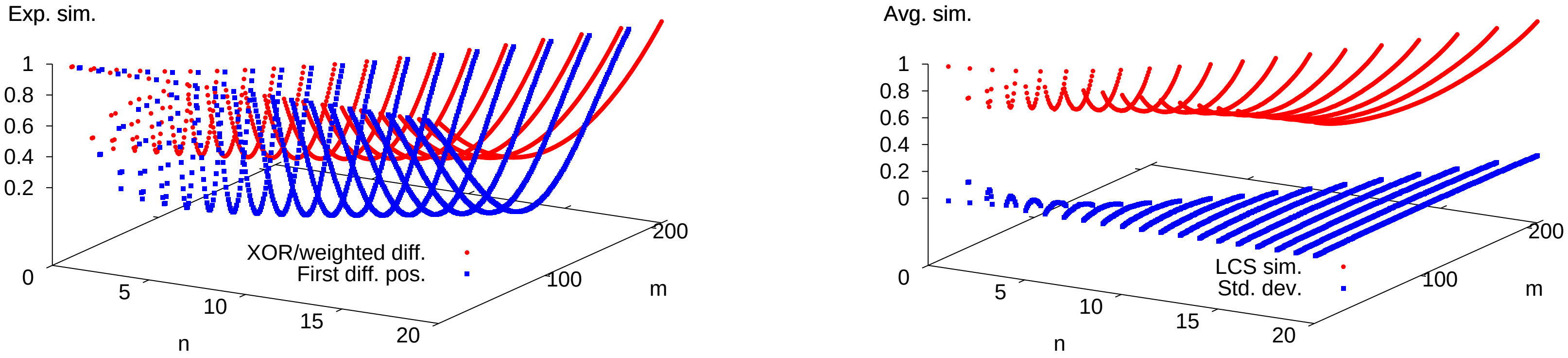


Figure 6.4: On the left, the expected value of Equation 6.8 for $\delta_{\oplus}(\mathcal{B}_1, \mathcal{B}_2)$ and $\delta_{\text{pos}}(\mathcal{B}_1, \mathcal{B}_2)$ (top curves), and $b_{\text{pos}}(\mathcal{B}_1, \mathcal{B}_2)$ (bottom curves) for two random bit strings $\mathcal{B}_1$ and $\mathcal{B}_2$ with m ones and $(n(n+1)/2 - m)$ zeroes (corresponding to two random reflexive graphs with n vertices and m edges; the corresponding formulas are given in Table 6.3. On the right, the average LCS similarity (top curves) over 1,000 pairs of random bit strings for each (n, m) pair, together with the standard deviation for each data point (bottom curves).

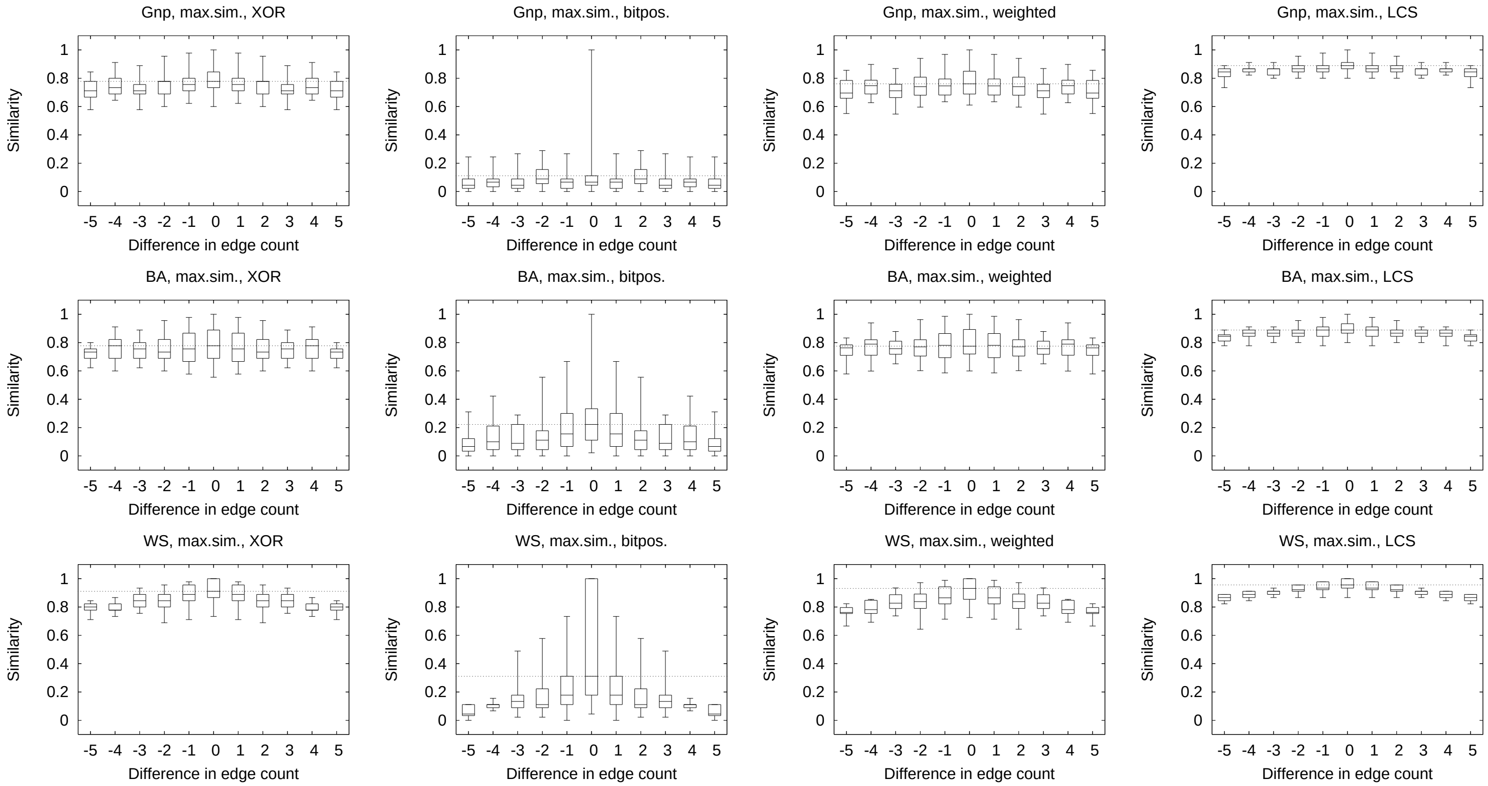


Figure 6.5: Box-whiskers plots that visualize the median and the deviations of the maximum similarity value over the set of k lexicographically first strings over a set of 30 graphs of order nine and their modified versions with one to five additional or missing edges. The first column corresponds to $\delta_{\oplus}(\mathcal{B}_1, \mathcal{B}_2)$ of Equation 6.3, the second column to $\delta_{\text{pos}}(\mathcal{B}_1, \mathcal{B}_2)$ of Equation 6.6, the third column to $\delta_w(\mathcal{B}_1, \mathcal{B}_2)$ of Equation 6.7, and the right-most column has the LCS similarities.

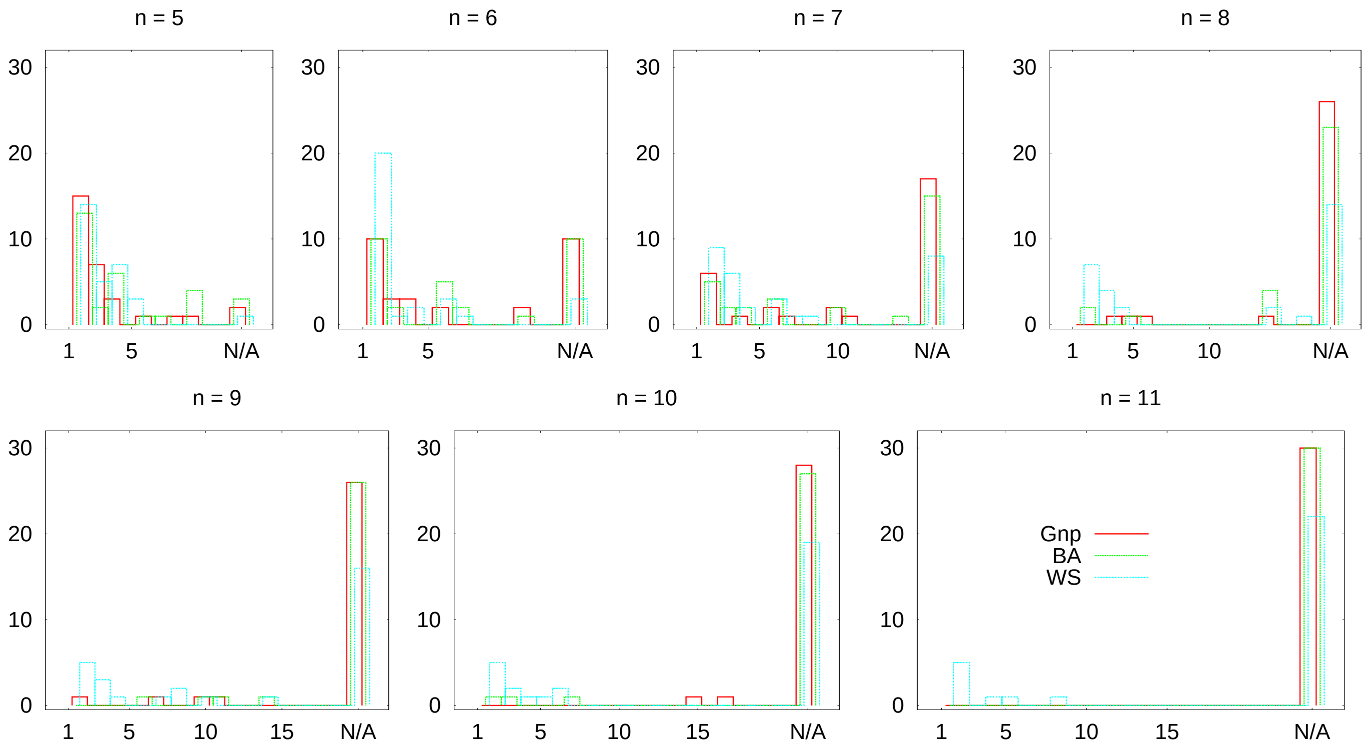


Figure 6.6: For the generation models used, the position of the list L of the lexicographically first $2n$ bit strings at which the greedy bit string $\mathcal{B}$ was found for the sets of 50 graphs of orders $[5, 10]$; the position $(2n+1)$ represents the cases in which the greedy string was not among the exact strings. The histograms for the $\mathcal{G}_{n,p}$ model were shifted 0.25 to the left and those of the WS model 0.25 to the right to avoid overlap of the histograms.

6.2 PARTIAL MATCHING IN LABELED GRAPHS

The *Semantic Web* [31] is an overlay of the current World-Wide Web with metadata annotations that aim to help in structuring, querying, and browsing information available on the web. With the current plenitude of languages, terminology, and different practises of people writing web pages, having a computer organize and interpret the information available is demanding if not impossible. For an introduction to the Semantic Web, we recommend an article by Berners-Lee *et al.* [31].

In the Semantic Web, each resource on the web has a *unique resource identifier* (URI) that enables a computer system to identify the resources referred to in the information it attempts to parse. *Ontologies* define classes of *objects*, inter-object *relations*, and *inference rules* that allow reasoning on objects and their relations. One of the goals of the use of ontologies in the Semantic Web is to resolve issues related to ambiguous, ill-defined, misused, or overlapping terminology.

The *Resource Description Framework* (RDF) is a representation of an ontology as a directed, labeled graph, where the vertices represent the resources (each of which has a URI) and the edges represent properties of the resources [46]. An RDF is formed by triples that contain a *subject*, a *predicate* and an *object*, where the subject and object map to vertices of the graph and the predicate to an edge from the subject vertex to the object vertex. An RDF schema is a set of definitions of classes, inheritance relations between classes, and class properties. Such schemas can be extended (with OWL) to define ontologies and related knowledge. Applications include social networks and semantic search engines in general; in the literature, common data sets for experiments are RDF versions of databases such as IMDB [159] and DBLP [206].

An RDF graph is a richer structure than the undirected, unweighted graphs we have mostly concentrated on. All of the vertices have *labels* $\omega \in \mathcal{V} \cup \perp$, where $\perp$ is a "dummy" label for those vertices that have no semantic label assigned. The graph definition includes a function $f_{\mathcal{V}} : V \to \mathcal{V} \cup \perp$. We denote the label of vertex v by $f_{\mathcal{V}}(v)$. Similarly the edges are labeled, in addition to being directed; an edge label $\epsilon \in \mathcal{E} \cup \perp$ is assigned to each edge by the function $f_{\mathcal{E}} : E \to \mathcal{E}$ and the label of edge $\langle v, w \rangle$ is denoted by $f_{\mathcal{E}}(\langle v, w \rangle)$. Note that $\mathcal{V}$ and $\mathcal{E}$ may and often in applications do have a non-empty intersection. The edge set allows multiple edges for a given pair of vertices, i.e., the graph may be a multigraph. Also reflexive edges are possible.

A special application to the similar-subgraph search is the distributed construction of a collaborative RDF ontology graph by several, independent users who use different terminology and hence can not easily determine by browsing the existing schema or by performing keyword-based searches if the concept they are about to introduce is already present. Such distributed annotation systems are becoming increasingly popular in bioinformatics applications, such as in annotation of genomes. The schema may either be directly edited by several users or the process may involve aggregation of independently written schema. A similar problem is that of combining database views from several independent servers [220].

Two descriptions of the same concept, such as a university course that has a lecturer, a lecture hall, a schedule, a course book, etc., are however likely to have similar *structure*. Hence, if such a concept is already included in the existing schema, searching the graph by using the newly proposed concept as a pattern will return parts of the schema that are structurally close to the pattern. The result of the search can then be presented to the user so that he or she may evaluate whether an already existing concept in the schema could be used instead of introducing duplicate or overlapping elements into the ontology. With large ontologies and big user populations, such as with the Semantic Web, such a mechanism could greatly assist in avoiding redundancy in the schema.

With this application in mind, it is easy to see that exact isomorphism between the pattern and the subgraphs encountered is not of interest, as two people independently designing exactly the same ontology is less likely than them designing largely overlapping ones. The application brings many new challenges in comparison to the simplified scenario discussed above, such as vertex and edge labels and the need to mine for disconnected patterns.

An application of pattern mining in an RDF is the realization of *queries*, where a partial pattern is given and (approximate) matches to the pattern from an RDF database are requested. Kanza *et al.* [176] use vertex mappings from the pattern to the subgraphs of a *rooted*, finitely branching "database graph", which imposes restrictions that we would like to avoid. First of all, we do not place any restrictions on the graph $G = (V, E)$, and secondly, we are also interested in matches where a single vertex in the user-defined query pattern can be captured by two or more vertices in a "matching" subgraph or the opposite scenario, where the pattern uses two or more vertices to represent a concept only captured by a single vertex in the graph G. The approach of Kanza *et al.* preserves edges of the pattern by aiming to satisfy edge constraints, whereas we would prefer a more flexible way of handling missing, additional or reversed-direction edges. Figure 6.7 illustrates some examples of structures and matches we aim to mine.

One work-around to the limitations of vertex-to-vertex mappings is to mod-

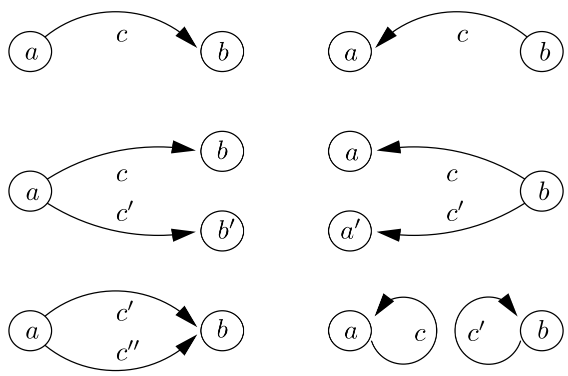


Figure 6.7: Some simple structures that we would like to be considered as possible matches when one is given as a pattern P and the others appear in the graph G as subgraphs. The label similarities are represented in a non-formal manner such that "similar" labels are denoted by primed versions of the same symbol $d, d', d'', \ldots$ and labels not similar to d-labels are represented by another symbol.

ify the user-defined query, i.e., to make *transformations* to the pattern P that produce a *set* of patterns $P_1, P_2, \ldots, P_k$, and then search the graph $G = (V, E)$ for occurrences of all these patterns P_i and combine the results to obtain subgraphs similar to P but with more flexibility within the order, size and connectivity structure of the matches. Another possibility is to define similarity measures that do not attempt to produce a one-to-one mapping of the pattern to the subgraph but evaluate the similarity by other means.

Algorithms to compute graph similarity and algorithms for inexact matching for directed, labeled graphs have been proposed [85, 86]. Blondel *et al.* [35] calculate similarity values for vertices with a matrix scheme that draws from Kleinberg's famous "hubs and authorities" method for ranking web pages [188], similarly calculated as an eigenvector of a matrix derived from the graph topology. A matrix approach to determining the similarity of two graphs is proposed by Blondel and Van Dooren [36].

A nice approach by Jeh and Widom [163]for computing vertex similarities for a given graph is based on the idea of the PageRank algorithm [47]. The vertex similarity values are based on the edge set: two vertices are similar if the neighbors these vertices point to are similar [163]. The formal definition is recursive in the spirit of PageRank and the algorithm itself is called SimRank. Jeh and Widom [164] present in later work a tool called F-Miner for answering similarity and relevance queries for labeled graphs.

In the following we assume that we have been given two similarity functions that evaluate how similar two labels are; one function for vertices and another for edges, both assigning similarity one to identical labels and any pair that contains at least one occurrence of $\perp$. We denote these functions respectively by $\rho_{\mathcal{V}} : \mathcal{V} \times \mathcal{V} \rightarrow [0, 1]$ and $\rho_{\mathcal{E}} : \mathcal{E} \times \mathcal{E} \rightarrow [0, 1]$. A *variable vertex* is a vertex v which has no label, i.e., $f_{\mathcal{V}}(v) = \perp$. This means that $\rho_{\mathcal{V}}(v, w) = 1$ for any vertex w.

We denote the multi-set of labels that are assigned to a set A by $\mathcal{L}(A)$. For example, in a subgraph $G^S = (S, F)$ where $S = \{v_1, v_2, \ldots, v_k\}$,

$$\mathcal{L}(S) = \{f_{\mathcal{V}}(v_1), f_{\mathcal{V}}(v_2), \ldots, f_{\mathcal{V}}(v_k)\}; \tag{6.11}$$

this is a multi-set because more than one vertex may have the same label. We use the same notation for labels of edge sets. Given two multi-sets of labels, $\mathcal{L}(A_1)$ and $\mathcal{L}(A_2)$ we calculate their similarity as

$$\rho(\mathcal{L}(A_1), \mathcal{L}(A_2)) = \frac{|\mathcal{L}(A_1) \cup \mathcal{L}(A_2)|}{\max\{|A_1|, |A_2|\}} \in [0, 1], \tag{6.12}$$

where the intersection of a multi-set is defined naturally to contain an element a b times if and only if it appears at least b times in both of the sets for which the intersection is computed and no more than b times in at least one of them.

Similarities between a pattern $P = (V_P, E_P)$ and a subgraph $G^S = (S, F)$ (assuming that the pattern and the subgraph are *independent* graphs in the sense that they share no vertices) inherently compose of several factors; we listed some basic ones in Table 6.4.

In addition to the measures listed in Table 6.4, we wish to study how well the structure of one graph maps to that of the other, without having to solve

Table 6.4: Some fundamental difference measures for two graphs $P = (V_P, E_P)$ and $G^S = (S, F)$. The measures all assign the low values for identical graphs; hence the subtraction from one in those that naturally map identical sets to one instead.

`v1` Normalized difference of the vertex label sets $1 - \frac{1}{|S||V_P|}\sum_{v\in S}\sum_{w\in V_P} \rho_{\mathcal{V}}\left(f_{\mathcal{V}}(v), f_{\mathcal{V}}(w)\right)$.

`e1` Normalized difference of the edge label sets $1 - \frac{1}{|F||E_P|}\sum_{\langle v_1,v_2\rangle\in F}\sum_{\langle v_3,v_4\rangle\in E_P} \rho_{\mathcal{E}}\left(f_{\mathcal{E}}(\langle v_1, v_2\rangle), f_{\mathcal{E}}(\langle v_3, v_4\rangle)\right)$.

`v2` The correspondence of the orders of the two graphs, $|S| = k_G^S$, $|V_P| = k_P$ as $\delta\left(k_G^S, k_P\right) = \frac{\left|k_G^S - k_P\right|}{\max\{k_G^S, k_P\}}$.

`e2` The correspondence of the size of the two graphs, defined as in `v2`: $\delta\left(|F|, |E_P|\right)$.

the computationally hard graph isomorphism problem. We resort to binary relations $\mathcal{R} : S \times V_P$ composed of $|\mathcal{R}| = k_{\mathcal{R}}$ vertex pairs $\{v, w\}$ such that $v \in S$ and $w \in V_P$. We assume that a relation is constructed by pairing vertices that hold a similar role in the graphs under comparison; for isomorphic graphs, a good relation would only contain a vertex pair if those two vertices map to each other in a graph isomorphism.

Similarity of two graphs $G^S = (S, F)$ and $P = (V_P, E_P)$ can be studied by determining how well such a binary relation $\mathcal{R} : S \times V_P$ can satisfy the following three criteria:

(i) `r1`: How low is the *deficit* of a *maximal matching* in a *bipartite graph* with vertex set $S \cup V_P$ with the pairs of the relation $\mathcal{R}$ as a set of undirected edges. A *matching* in a bipartite graph is a subgraph of a bipartite graph where $\deg(v) \geq 1$ for all vertices; a matching is *maximal* if it has maximal edge- and vertex-cardinalities. The *deficit* of a matching is the number of vertices in the bipartite graph that have degree zero in the matching, sometimes only perceived from the smaller-cardinality bipartite vertex set. We denote the number of edges in the maximal matching by $\mathcal{M}$ and normalize a deficit measure to be one for relations that contain a complete matching:

$$\rho_1^{\mathcal{R}}(\mathcal{R}, P) = \frac{\mathcal{M}}{\min\{|S|, |V_P|\}}. \tag{6.13}$$

We choose to compute a maximal matching instead of simply checking whether the relation itself is a bijection, as structural symmetries will introduce redundancies in relations constructed to pair vertices that have a similar role in the two graphs being compared. The weakness of this measure is that simply adding all vertex pairs to the relation maximizes it; hence we define two more measures that punish the presence of extra pairs in the relation unless the vertex labels and corresponding connectivity patterns are highly similar.

(ii) `r2`: How well the relation $\mathcal{R}$ preserves the vertex labels:

$$\rho_2^{\mathcal{R}}(\mathcal{R}, P) = \frac{\sum_{\{v,w\}\in\mathcal{R}} \rho_{\mathcal{V}}(f_{\mathcal{V}}(v), f_{\mathcal{V}}(w))}{k_{\mathcal{R}}} \in [0,\ 1]. \tag{6.14}$$

(iii) `r3`: How well the relation $\mathcal{R}$ preserves the edges of the graphs, with respect to their presence, direction, and labels. We use an indicator function

$$\mathcal{I}(\{v,w\}) = \begin{cases} 1, & \text{if } \{v,w\} \in \mathcal{R}, \\ 0, & \text{otherwise} \end{cases} \tag{6.15}$$

and the abbreviations

$$\begin{aligned} i &= \mathcal{I}(\{v,v'\}) \cdot \mathcal{I}(\{w,w'\}) \\ j &= \mathcal{I}(\{v,w'\}) \cdot \mathcal{I}(\{w,v'\}) \\ a &= \rho_{\mathcal{E}}(f_{\mathcal{E}}(\langle v,w\rangle), f_{\mathcal{E}}(\langle v',w'\rangle)) \\ b &= \rho_{\mathcal{E}}(f_{\mathcal{E}}(\langle v,w\rangle), f_{\mathcal{E}}(\langle w',v'\rangle)) \end{aligned} \tag{6.16}$$

to define a similarity measure in [0, 1]:

$$\rho_3^{\mathcal{R}}(\mathcal{R}, P) = \frac{1}{2\,|F|\,|E_P|} \sum_{\langle v,w\rangle\in F} \sum_{\langle v',w'\rangle\in E_P} (i \cdot a + \beta \cdot j \cdot b) \tag{6.17}$$

where $\beta \in [0,\ 1]$ is a parameter that allows rewarding for the presence of an otherwise appropriate but reverse-direction edge; we use $\beta = 0.5$.

In light of the above measure and those of Table 6.4, in order to evaluate the similarity of two graphs G^S and P, we would need to find a binary relation of the vertex sets that simultaneously optimizes the above criteria. Note that any relation that preserves all edges in both directions defines a graph isomorphism which suggests that instead of optimizing the relation, constructing a relation that can be assumed to be approximately optimal is a feasible approach to construct a similarity-evaluation subroutine that needs to be invoked frequently during the subgraph scan.

We propose obtaining an approximate relation by iteratively constructing a weighted bipartite graph $B = (S \cup V_P, E_B)$ and then pruning the edges to improve the similarity rating. Such a construction should at least account for edge-presence, edge and vertex labels, and degree differences in some way. First we build an auxiliary bipartite graph using the *edges* as vertices: the vertex set is $F \cup E_P$ and an edge is placed between two vertices if the corresponding edge labels have nonzero similarity, storing the similarity as the weight of that edge. At this point also thresholding based on application-specific *a priori* information could be done.

We then switch to using the intended vertex set $S \cup V_P$ by splitting each vertex of the auxiliary graph into the endpoints of the edge it represents. Each edge of the auxiliary bipartite graph is multiplied into four edges: an edge connecting $\{v,w\}$ $(v,w \in S)$ and $\{v',w'\}$ $(v',w' \in V_P)$ in the auxiliary graph produces edges $\{v,v'\}$, $\{w,v'\}$, $\{v,w'\}$, and $\{w,w'\}$. Multiple occurrences of each vertex are merged into one, and the edge weights of the resulting bipartite graph are multiplied by the vertex label similarity of the endpoints, pruning out zero-weight edges.

Next we compute for each vertex pair $v \in S$, $w \in V_P$ such that $\{v, w\}$ is included in the bipartite graph

$$\frac{1}{|\deg(v) - \deg(w)| + 1} \tag{6.18}$$

and multiply the weight of the corresponding edge by this number, thus weakening the edges that connect vertices that have different degrees. After the weight computations, we compute the minimum of the maximum weights in the neighborhoods of the bipartite graph:

$$\xi = \min_{v \in S \cup V_P} \left\{ \max_{w \in \Gamma(v)} \{\omega(\{v, w\})\} \right\} \tag{6.19}$$

and prune out any edge with weight lower than ξ in order to limit the number of pairs in the resulting relation. The relation $\mathcal{R}$ is formed by the vertex pairs that are connected by an edge in the remaining bipartite graph. The procedure has polynomial worst-case complexity, albeit high. Considering that the worst-case complexity for all known general algorithms are exponential (the naïve solution scales as a factorial of the graph orders), this is not a bad complexity.

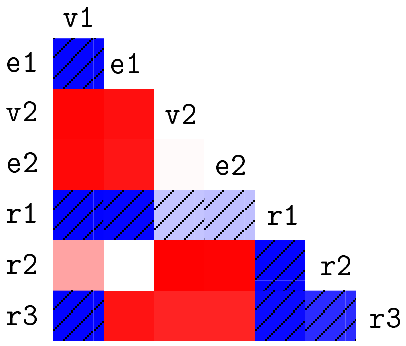


Figure 6.8: Spearman rank correlations among the difference measures proposed in Table 6.4 and the similarity obtained from the construction of a weighted bipartite graph. Red corresponds to one, blue to minus one, and lighter colors to values gradually from one to zero (shades of red for positive correlations, shades of blue for negative correlations). The cells corresponding to negative correlations are striped diagonally. Note that as the difference measures assign low values to similar graphs, a negative correlation between the first four and the last three implies agreement.

We generated a set of 100 small uniform random graphs of order $n \in [5, 10]$ and size $m \in [2n, n(n-1)]$ using the $\mathcal{G}_{n,m}$ model, and then added labels $\mathcal{L}$ of length $|\mathcal{L}| \in [3, 6]$ for each vertex and edge using an alphabet $\Sigma = \{a, b, c\}$ such that each symbol in Σ was equally likely to appear at each position of the label string. For label similarities, we used the *normalized edit distance*

$$\mathrm{dist}_{\mathrm{edit}}^{\mathrm{norm}}(\mathcal{L}_1, \mathcal{L}_2) = \frac{\mathrm{dist}_{\mathrm{edit}}(\mathcal{L}_1, \mathcal{L}_2)}{\max\{|\mathcal{L}_1|, |\mathcal{L}_2|\}} \tag{6.20}$$

where $\mathrm{dist}_{\mathrm{edit}}(\mathcal{L}_1, \mathcal{L}_2)$ is the minimum number of insertions, deletions, and replacements required to transform $\mathcal{L}_1$ into $\mathcal{L}_2$ (computable by a standard dynamic-programming algorithm [142]). We then computed the similarities using all four difference measures of Table 6.4 and the bipartite-graph method proposed; Table 6.8 shows the rank correlations of the similarity factors together with the similarity value obtained by the bipartite construction; Pearson correlations for the values themselves were practically identical in visualization, although small numerical differences are present.

The relation-derived measure `r1` correlates with the four difference measures negatively as desirable, but the three relation-derived measures are not all positively correlated themselves; although the first and the second are correlated, their correlations with the other measures differ. This gives us a reason to believe that none of the relation-dependent measures alone will suffice to characterize graph similarity, as they cannot be explained by each other. We experimented further with a combined measure

$$\rho^{\mathcal{R}}(\mathcal{R}, P) = \rho_1^{\mathcal{R}}(\mathcal{R}, P) \cdot \frac{\rho_2^{\mathcal{R}}(\mathcal{R}, P) + \rho_3^{\mathcal{R}}(\mathcal{R}, P)}{2} \tag{6.21}$$

using the above construction for the relation $\mathcal{R}$ given P and G^S over the set of 100 random graphs to see which pairs of graphs it points out as similar and which as dissimilar. Using a simple product of the three terms punishes too severely graphs that have high similarity over vertex labels but low over edge labels and vice versa, and hence we use their sum instead of allowing either one to "pull down" the total similarity.

Some of these pairs are listed in Table 6.5. For each of the pairs shown, we ignored the labels and computed the edit distance using the set of all possible adjacency matrices and having operation cost one for edge-deletions and removals, first embedding both of the graphs to the order of the larger one. Only nine of the twenty edit-distance computations finished in less than a few hours; the rest were allowed to run for several hours (some well over a month) and then terminated, recording the current known minimum distance, which serves as an upper bound to the edit distance. The largest graphs take the longest to compute, and the bound improves as the computation proceed, and hence we can assume that the bounds obtained are the crudest for the pair where at least one of the instances is of order ten.

It is easy to see from Table 6.5 that the measure of Equation 6.21 gives high similarity to graphs of similar order and size, without putting too much emphasis on the similarity, as it assigns highest similarities to pairs where structural and labeling similarities go beyond such simple means.

Similarity of the edge structure, as captured by the heavy-computation edit distance, is well-preserved in the computationally lighter measure: in general, the graphs of low similarity have high edit distance (average bound 11.2) and those of high similarity have low edit distance (average bound 5.9), although a few exceptions are included. The relatively high bounds on the last two pairs can be explained by the order of the graphs: the computation has not obtained a tight bound before being cut off. Such edit distances provide a more established but computationally heavily obtained information on graph similarity for directed graphs.

The two low-similarity instances with low edit distance have differing edge labelings, the pair (13, 51) only has `r3` ≈ 0.01 and the pair (15, 39) ≈ 0.02. The desired effect is observed in the experiment: the differences in vertex

Table 6.5: A few pairs of graphs that obtained the lowest or highest similarities by Equation 6.21; the order and size of the graphs and the unlabeled edit distance $\mathrm{dist}_{\mathrm{edit}}\left(G^S, P\right)$ are shown (in the last column, denoted by d) for each pair. The identifiers of the graphs in the test set of 100 graphs are shown in the first two columns.

G^S	P	$\rho^{\mathcal{R}}$	$\lvert S\rvert$	$\lvert F\rvert$	$\lvert V_P\rvert$	$\lvert E_P\rvert$	d
13	51	0.0774	7	14	5	10	5
47	87	0.1089	5	10	10	21	≤ 12
18	44	0.1090	5	10	10	21	≤ 14
17	77	0.1120	5	10	9	18	≤ 10
44	47	0.1178	10	21	5	10	≤ 15
18	56	0.1209	5	10	10	22	≤ 16
15	39	0.1270	7	14	6	12	4
2	17	0.1292	10	21	5	10	≤ 14
26	47	0.1301	10	20	5	10	≤ 14
17	93	0.1305	5	10	10	20	≤ 13
8	61	0.4973	7	14	8	16	5
75	81	0.4997	9	18	8	16	≤ 8
11	38	0.5004	6	12	7	14	5
16	95	0.5031	6	12	5	10	3
50	60	0.5063	8	16	7	14	4
30	46	0.5070	6	12	8	16	5
14	81	0.5170	7	14	8	16	5
67	78	0.5538	7	14	8	16	5
22	98	0.5665	10	20	10	22	≤ 8
31	72	0.5696	10	22	10	21	≤ 11

or edge labels bring down the similarity value of a structurally close pair, whereas agreement in labels in the structural parts that do overlap increases the similarity. Hence we believe that an approach based on computing a relation between the vertex sets of two graphs considering vertex and edge similarities is a promising way to evaluate similarity of labeled directed graphs in polynomial time.

7 CONCLUSIONS

We opened the thesis by explaining the structure and motivation of the work in Chapter 1, also explaining the collaborations and contributions. In Chapter 2 we summarized recent advances on natural networks, through the concept of nonuniform networks. We reviewed essential properties and analytical models of such networks and discussed the implications that structural properties of graph instances have for the design on graph algorithms. For a more concrete view on the topic at hand, we studied as examples a data set on graph coloring algorithms and a graph representing the words of the English language called WordNet.

After establishing the structural complexity inherent in natural networks, we began the development of algorithms for studying and handling such complex networks, accompanied by reviews of existing methods. First in Chapter 3 we studied graph sampling, proposing a generalizable Markov-chain combination for uniform sampling.

This was followed by a broad discussion on graph clustering in Chapter 4, including proposals for local methods that efficiently locate the clusters of given seed vertices in large nonuniform networks. The applications explored in the chapter include ad hoc networks, storing massive graphs, and evaluation of graph generators.

In Chapter 5 we addressed problems related to search and spanning tree construction, with emphasis on constructing spanning trees for Euclidean-plane communication networks that are not too far from minimizing the distances of communicating nodes but that also take into account the number of hops each communication packet must traverse. We also discussed the effect of graph structure on search strategies and random walks, studying the effect of storing limited-capacity lookahead buffers in a network.

In the rest of the thesis we concentrated on mining subgraphs that are similar to a given pattern in Chapter 6. We first proposed a method of estimating the similarity of undirected reflexive graphs, and then studied similarity measures for labeled directed graphs.

For further work, of special interest are extensions to directed and weighted graphs on all of the topics studied in this work. Applying the presented methods on large real-world problem instances would give more insight on their flexibility and provide valuable feedback on how to improve the efficiency of the algorithms as well as on the structural properties present in the application-specific data.

In the area of sampling, we hope to study applications of sampling for property-discovery and network classification. We would like to work with practical applications of local clustering to gain insight on how the application area affects the choice of the fitness function. Also energy-efficient routing in mobile radio networks closer to the grass-level with knowledge of the physical capabilities of the nodes and the environment in which the network is to operate and under what conditions is a motivating area of future investigation. We are also interested in implementing information retrieval systems that use the clustering and pattern-matching methods to serve the data mining needs of people with little or no expertise in computer science.

BIBLIOGRAPHY

[1] Emile Aarts and Jan Karel Lenstra, editors. *Local Search in Combinatorial Optimization.* John Wiley & Sons, Inc., Chichester, UK, 1997.

[2] Alf-Christian Achilles. The collection of computer science bibliographies. At `http://liinwww.ira.uka.de/bibliography/`, accessed Dec. 2, 2002.

[3] Dimitris Achlioptas, Aaron Clauset, David Kempe, and Cristopher Moore. On the bias of traceroute sampling (or: Why almost every network looks like it has a power law). In Harold N. Gabow and Ronald Fagin, editors, *Proceedings of the Thirty-seventh Annual* ACM *Symposium on Theory of Computing* (STOC), pages 694–703, New York, NY, USA, 2005. ACM Press.

[4] Lada A. Adamic. The small world web. In S. Abiteboul and A.-M. Vercoustre, editors, *Proceedings of* ECDL'99, volume 1696 of *Lecture Notes in Computer Science*, pages 443–452, Berlin, Germany, 1999. Springer-Verlag GmbH.

[5] Lada A. Adamic, Rajan M. Lukose, Amit R. Puniyani, and Bernardo A. Huberman. Search in power-law networks. *Physical Review E*, 64(4):046135, October 2001.

[6] Pankaj K. Agarwal and Cecilia M. Procopiuc. Exact and approximation algorithms for clustering. In *Proceedings of the Ninth Annual* ACM-SIAM *Symposium on Discrete Algorithms*, pages 658–667, Philadelphia, PA, USA, 1998. Society for Industrial and Applied Mathematics.

[7] Rakesh Agrawal and H. V. Jagadish. Algorithms for searching massive graphs. IEEE *Transactions on Knowledge and Data Engineering*, 6(2):225–238, April 1994.

[8] Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. A survey on sensor networks. IEEE *Communication Magazine*, 40(8):102–114, 2002.

[9] Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38:393–422, 2002.

[10] Réka Albert, Hawoong Jeong, and Albert-László Barabási. Diameter of the World Wide Web. *Nature*, 401:130–131, September 1999.

[11] Réka Albert, Hawoong Jeong, and Albert-László Barabási. Error and attack tolerance of complex networks. *Nature*, 406:378–382, July 2000.

[12] David J. Aldous and James Allen Fill. Reversible Markov chains and random walks on graphs, Book in preparation. Some chapters are available at `http://www.stat.berkeley.edu/users/aldous/RWG/book.html`.

[13] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A survey of peer-to-peer content distribution technologies. ACM *Computing Surveys*, 36(4):335–371, December 2004.

[14] Fred S. Annexstein, Kenneth A. Berman, and Mijhalo A. Jovanovic. Broadcasting in unstructured peer-to-peer overlay networks. *Theoretical Computer Science*, 355(1):25–36, April 2006.

[15] S. V. Annibaldi and K. I. Hopcraft. Random walks with power-law fluctuations in the number of steps. *Journal of Physics A*, 35(41):8635–8645, October 2002.

[16] Gheorghe Antonoiu and Pradip K. Srimani. Distributed self-stabilizing algorithm for minimum spanning tree construction. In *Proceedings of the Third International Euro-Par Conference on Parallel Processing*, volume 1300 of *Lecture Notes in Computer Science*, pages 480–487, London, UK, 1997. Springer-Verlag GmbH.

[17] Alex Arenas, Antonio Cabrales, Albert Díaz-Guilera, Roger Guimerà, and Fernando Vega-Redondo. Search and congestion in complex networks. In Romualdo Pastor-Satorras, Miguel Rubi, and Albert Diaz-Guilera, editors, *Statistical Mechanics of Complex Networks*, volume 625 of *Lecture Notes in Physics*, pages 175–194, Berlin, Germany, 2003. Springer-Verlag GmbH.

[18] Franz Aurenhammer. Voronoi diagrams – a survey of a fundamental geometric data structure. ACM *Computing Surveys*, 23(3):345–405, 1991.

[19] Baruch Awerbuch. Optimal distributed algorithms for minimum weight spanning tree, counting, leader election and related problems (detailed summary). In *Proceedings of the Nineteenth Annual* ACM *Symposium on Theory of Computing (*STOC*)*, pages 230–240, New York, NY, USA, 1987. ACM Press.

[20] Ricardo Baeza-Yates, Barbara J. Poblete, and Felipe Saint-Jean. Evolución de la web chilena 2001–2002. Technical report, Centro de Investigación de la Web, Deptartamento de Ciencias de la Computación, Universidad de Chile, Santiago, Chile, January 2003.

[21] Pierre Baldi, Paolo Frasconi, and Padhraic Smyth. *Modeling the Internet and the Web: Probabilistic Methods and Algorithms.* John Wiley & Sons, Inc., Chichester, UK, 2003.

[22] Farnoush Banaei-Kashani and Cyrus Shahabi. Criticality-based analysis and design of unstructured peer-to-peer networks as "complex systems". In *Third International Workshop on Global and Peer-to-Peer Computing*, 2003. CCGRID.

[23] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. In *Proceedings of the Fourty-third Annual* IEEE *Symposium on Foundations of Computer Science (*FOCS*)*, pages 238–247, Washington, DC, USA, 2002. IEEE Computer Society Press.

[24] Ziv Bar-Yossef, S. Ravi Kumar, and D. Sivakumar. Sampling algorithms: lower bounds and applications. In *Proceedings of the Thirty-third Annual* ACM *Symposium on Theory of Computing* (STOC), pages 266–275, New York, NY, USA, 2001. ACM Press.

[25] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286:509–512, October 1999.

[26] Albert-László Barabási, Réka Albert, and Hawoong Jeong. Mean-field theory for scale-free random networks. *Physica A*, 272:173–187, 1999.

[27] Albert-László Barabási, Erzsébet Ravasz, and Zoltán N. Oltvai. Hierarchical organization of modularity in complex networks. In Romualdo Pastor-Satorras, Miguel Rubi, and Albert Diaz-Guilera, editors, *Statistical Mechanics of Complex Networks*, volume 625 of *Lecture Notes in Physics*, pages 46–65, Berlin, Germany, 2003. Springer-Verlag GmbH.

[28] Albert-László Barabási, Erzsébet Ravasz, and Tamás Vicsek. Deterministic scale-free networks. *Physica A*, 299(3–4):559–564, October 2001.

[29] Wolfgang Barthel, Alexander K. Hartmann, and Martin Weigt. Phase transition and finite-size scaling in the vertex-cover problem. *Computer Physics Communications*, 169(1–3):234–237, July 2005.

[30] Ehrhard Behrends. *Introduction to Markov Chains, with Special Emphasis on Rapid Mixing.* Vieweg & Sohn, Braunschweig/Wiesbaden, Germany, 2000.

[31] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, pages 35–43, May 2001.

[32] Krishna Bharat and Andrei Z. Broder. A technique for measuring the relative size and overlap of public Web search engines. In *Proceedings of Seventh International World-Wide Web Conference*, pages 379–388, Amsterdam, The Neatherlands, 1998. Elsevier Science Publishers.

[33] Daniel K. Blandford, Guy E. Blellch, and Ian A. Kash. An experimental analysis of a compact graph representation. In Lars Arge, Giuseppe F. Italiano, and Robert Sedgewick, editors, *Proceedings of the Sixth Workshop on Algorithm Engineering and Experiments and the First Workshop on Analytic Algorithmics and Combinatorics*, pages 49–61, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics.

[34] Daniel K. Blandford, Guy E. Blelloch, and Ian A. Kash. Compact representations of separable graphs. In *Proceedings of the Fourteenth Annual* ACM–SIAM *Symposium on Discrete Algorithms* (SODA), pages 579–688, New York, NY, USA, 2003. ACM Press.

[35] Vincent D. Blondel, Anahí Gajardo, Maureen Heymans, Pierre Senellart, and Paul Van Dooren. A measure of similarity between graph vertices. with applications to synonym extraction and web searching. SIAM *Review*, 46(4):647–666, 2004.

[36] Vincent D. Blondel and Paul Van Dooren. Similarity matrices for pairs of graphs. In Jos C. M. Baeten, Jan Karel Lenstra, Joachim Parrow, and Gerhard J. Woeginger, editors, *Automata, Languages and Programming (The Thirtieth International Colloquium,* ICALP*)*, volume 2719 of *Lecture Notes in Computer Science*, pages 739–750, Heidelberg, Germany, 2003. Springer-Verlag GmbH.

[37] Marián Boguñá and Romualdo Pastor-Satorras. Class of correlated random networks with hidden variables. *Physical Review E*, 68(3):036112, September 2003.

[38] Marián Boguñá, Romualdo Pastor-Satorras, and Alessandro Vespignani. Epidemic spreading in complex networks with degree correlations. In Romualdo Pastor-Satorras, Miguel Rubi, and Albert Diaz-Guilera, editors, *Statistical Mechanics of Complex Networks*, volume 625 of *Lecture Notes in Physics*, pages 127–147, Berlin, Germany, 2003. Springer-Verlag GmbH.

[39] Béla Bollobás. *Random Graphs*. Number 73 in Cambridge Studies in Advanced Mathematics. Cambridge University Press, Cambridge, UK, second edition, 2001.

[40] Béla Bollobás and Oliver Riordan. Robustness and vulnerability of scale-free random graphs. *Internet Mathematics*, 1(1):1–35, 2004.

[41] Béla Bollobás and Oliver M. Riordan. The diameter of a scale-free random graph. *Combinatorica*, 24(1):5–34, January 2004.

[42] James G. Booth, George Casella, and James P. Hobert. Clustering using objective functions and stochastic search, August 2004. Submitted to Biometrika, available at `http://www.bscb.cornell.edu/Homepages/Jim_Booth/papers.html`.

[43] Stephen P. Boyd, Persi Diaconis, and Lin Xiao. Fastest mixing Markov chain on a graph. SIAM *Review*, 46(4):667–689, December 2004.

[44] P. Bradley, U. Fayyad, and C. Reina. Scaling clustering algorithms to large databases. In *Proceedings of the Fourth* ACM *International Conference on Knowledge Discovery and Data Mining*, pages 9–15, New York, NY, USA, 1998. ACM.

[45] Ulrik Brandes, Marco Gaertler, and Dorothea Wagner. Experiments on graph clustering algorithms. In Giuseppe Di Battista and Uri Zwick, editors, *Proceedings of the Eleventh European Symposium on Algorithms*, volume 2832 of *Lecture Notes in Computer Science*, pages 568–579, Heidelberg, Germany, September 2003. Springer-Verlag GmbH.

[46] Dan Brickley and R. D. Guha. RDF vocabulary description language 1.0: RDF schema. Technical Report W3C Recommendation 10, The World-Wide Web Consortium, `http://www.w3.org/`, February 2004. `http://www.w3.org/TR/rdf-schema/`.

[47] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and* ISDN *Systems*, 30(1–7):107–117, 1998.

[48] Andrei Z. Broder, S. Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet Wiener. Graph structure in the web. *Computer Networks*, 33(1–6):309–320, June 2000.

[49] Steve P. Brooks, Petros Dellaportas, and Gareth O. Roberts. An approach to diagnosing total variation convergence of mCMC algorithms. *Journal of Computational and Graphical Statistics*, 6:251–265, 1997.

[50] Thang Nguyen Bui, F. Thomson Leighton, Soma Chaudhuri, and Michael Sipser. Graph bisection algorithms with good average case behavior. *Combinatorica*, 7(2):171–191, 1987.

[51] Horst Bunke. Error correcting graph matching: on the influence of the underlying cost function. IEEE *Transactions on Pattern Analysis and Machine Intelligence*, 21(9):917–922, 1999.

[52] Horst Bunke. Developments in graph matching. In *Proceedings of the Fifteenth International Conference on Pattern Recognition*, volume 2, pages 117–124, Washington, DC, USA, 2000. IEEE Computer Society Press.

[53] Horst Bunke, Pasquale Foggia, C. Guidobaldi, Carlo Sansone, and Mario Vento. A comparison of algorithms for maximum common subgraph on randomly connected graphs. In Terry Caelli, Adnan Amin, Robert P. W. Duin, Mohamed S. Kamel, and Dick de Ridder, editors, *Structural, Syntactic, and Statistical Pattern Recognition in Proceedings of Joint* IAPR *International Workshops* SSPR *2002 and* SPR *2002*, volume 2396 of *Lecture Notes in Computer Science*, pages 123–132, New York, NY, USA, 2002. Springer-Verlag GmbH.

[54] Horst Bunke, Xiaoyi Jiang, and Abraham Kandel. On the minimum common supergraph of two graphs. *Computing*, 65(1):13–25, July 2000.

[55] Horst Bunke and Kim Shearer. A graph distance metric based on the maximal common subgraph. *Pattern Recognition Letters*, 19(3–4):255–259, May 1998.

[56] Diego Calvanese, Giuseppe de Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Answering regular path queries using views. In

Proceedings of the Sixteenth International Conference on Data Engineering (ICDE'00), pages 389–398, Washington, DC, USA, 2000. IEEE Computer Society Press.

[57] Tracy Camp, Jeff Boleng, and Vanessa A. Davies. A survey of mobility models for ad hoc network research. *Wireless Communication and Mobile Computing*, 2(5):483–502, September 2002.

[58] George Casella, Michael Lavine, and Christian P. Robert. Explaining the Gibbs sampler. *The American Statistician*, 55(4):299–305, 2001.

[59] Deepayan Chakrabarti, Yiping Zhan, Daniel Blandford, Christos Faloutsos, and Guy Blelloch. Netmine: New mining tools for large graphs. In *Proceedings of the SIAM International Conference on Data Mining 2004 Workshop on Link Analysis, Counter-terrorism and Privacy*, 2004.

[60] Anantha P. Chandrakasan, Rex K. Min, Manish Bhardwaj, Seong-Hwan Cho, and Alice Wang. Power aware wireless microsensor systems. In *Proceedings of the European Conference on Solid-State Circuit Design (ESSCIRC 2002)*, pages 47–54, New York, NY, USA, 2002. IEEE.

[61] Moses Charikar, Chandra Chekuri, Tomás Feder, and Rajeev Motwani. Incremental clustering and dynamic information retrieval. In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing (STOC)*, pages 626–635, New York, NY, USA, 1997. ACM Press.

[62] David Cheng, Ravi Kannan, Santosh Vempala, and Grant Wang. On a recursive spectral algorithm for clustering from pairwise similarities. Technical Report MIT-LCS-TR-906, Laboratory of Computer Science, Massachusetts Institute of Technology, Boston, MA, USA, 2003.

[63] Marco Chiarandini. *Stochastic local search methods for highly constrained combinatorial optimisation problems: Graph coloring, generalisations, and applications*. PhD thesis, Technische Universität Darmstadt, Darmstadt, Germany, In preparation.

[64] Marco Chiarandini and Thomas Stützle. An application of iterated local search to graph coloring. In D. S. Johnson, A. Mehrotra, and M. Trick, editors, *Proceedings of the Computational Symposium on Graph Coloring and its Generalizations*, pages 112–125, September 2002.

[65] Siddhartha Chib and Ivan Jeliazkov. Accept-reject Metropolis-Hastings sampling and marginal likelihood estimation. *Statistica Neerlandica*, 59(1):30–44, 2005.

[66] Paul-Alexandru Chirita, Wolfgang Nejdl, and Oana Scurtu. Knowing where to search: Personalized search strategies for peers in P2P

networks. In Jamie Callan, Norbert Fuhr, and Wolfgang Nejdl, editors, *Proceedings of the* SIGIR *Workshop on Peer-to-Peer Information Retrieval, The Twenty-seventh Annual International* ACM SIGIR *Conference*. Electronic publication, July 2004.

[67] Tham Yoke Chun. World Wide Web robots: an overview. *Online Information Review*, 23(3):135–142, 1999.

[68] Fan R. K. Chung. *Spectral Graph Theory*. American Mathematical Society, Providence, RI, USA, 1997.

[69] Fan R. K. Chung and R. B. Ellis. A chip-firing game and Dirichlet eigenvalues. *Discrete Mathematics*, 257:341–355, 2002.

[70] Fan R. K. Chung and Linyuan Lu. The average distances in a random graph with given expected degrees. *Internet Mathematics*, 1(1):91–114, 2005.

[71] Fan R. K. Chung, Linyuan Lu, and Van Vu. The spectra of random graphs with given expected degrees. *Internet Mathematics*, 1(3):257–275, 2004.

[72] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Designing Privacy Enhancing Technologies: Proceedings of the International Workshop on Design Issues in Anonymity and Unobservability 2000*, volume 2009 of *Lecture Notes in Computer Science*, pages 46–66. Springer-Verlag GmbH, 2001.

[73] I. Vaughan L. Clarkson, Edwin D. El-Mahassni, and Stephen D. Howard. Sensor scheduling in electronic support using Markov chains. IEE *Proceedings: Radar, Sonar, and Navigation*, June 2005. Submitted for publication.

[74] Kenneth L. Clarkson. Further applications of random sampling to computational geometry. In *Proceedings of the Eighteenth Annual* ACM *symposium on Theory of Computing (*STOC*)*, pages 414–423, New York, NY, USA, 1986. ACM Press.

[75] Kenneth L. Clarkson. Applications of random sampling in computational geometry, II. In *Proceedings of the Fourth Annual Symposium on Computational Geometry*, pages 1–11, New York, NY, USA, 1998. ACM Press.

[76] Thomas H. Clausen and Philippe Jacquet. Optimized link state routing protocol (OLSR). Technical Report RFC 3626, Internet Engineering Task Force, Reston, VA, USA, 2003.

[77] Aaron Clauset and Cristopher Moore. Accuracy and scaling phenomena in Internet mapping. *Physical Review Letters*, 94(1):018701, January 2005.

[78] Aaron Clauset, Mark E. J. Newman, and Cristopher Moore. Finding community structure in very large networks. *Physical Review E*, 70(6):066111, 2004.

[79] Reuven Cohen, Shlomo Havlin, and Daniel ben Avraham. Efficient immunization strategies for computer networks and populations. *Physical Review Letters*, 91(24):247901, December 2003.

[80] Sara Cohen and Yehoshua Sagiv. Generating all maximal induced subgraphs for hereditary, connected-hereditary and rooted-hereditary properties. Technical Report cs.DS/0410039, arXiv.org e-Print archive, `http://arxiv.org/`, 2004.

[81] Brian F. Cooper. Quickly routing searches without having to move content. In *Proceedings of the Fourth International Workshop on Peer-to-Peer Systems (*IPTPS*)*, 2005.

[82] Colin Cooper and Alan Frieze. Crawling on simple models of web graphs. *Internet Mathematics*, 1(1):57–90, 2004.

[83] Colin Cooper, Alan Frieze, and Juan Vera. Random deletion in a scale-free random graph process. *Internet Mathematics*, 1(4):463–483, 2004.

[84] Luigi P. Cordella, Pasquale Foggia, Carlo Sansone, and Francesco Tortorella. Graph matching: a fast algorithm and its evaluation. In Anil K. Jain, Svetha Venkatesh, and Brian C. Lovell, editors, *Proceedings of the Fourteenth International Conference on Pattern Recognition*, volume 2, pages 1582–1584, Brisbane, CA, USA, 1998. IEEE Computer Society Press.

[85] Luigi P. Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. Subgraph transformations for the inexact matching of attributed relational graphs. *Computing*, 12:43–52, 1998.

[86] Luigi P. Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. An improved algorithm for matching large graphs. In *Proceedings of the Third* IAPR *TC-15 International Workshop on Graph-based Representation in Pattern Recognition*, pages 149–159, 2001.

[87] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. McGraw-Hill Book Co., Boston, MA, USA, second edition, 2001.

[88] M. Scott Corson and Anthony Ephremides. A distributed routing algorithm for mobile wireless networks. *Wireless Networks*, 1(1):61–81, February 1995.

[89] G. A. Croes. A method for solving traveling salesman problems. *Operations Research*, 6:791–812, 1958.

[90] Luciano da F. Costa, Francisco A. Rodrigues, Gonzalo Travieso, and P. R. Villas Boas. Characterization of complex networks: A survey of measurements. Technical Report cond-mat/0505185, arXiv.org e-Print archive, `http://arxiv.org/`, 2005.

[91] Elias Dahlhaus, Peter Dankelmann, Wayne Goddard, and Henda C. Swart. MAD trees and distance-hereditary graphs. *Discrete Applied Mathematics*, 131(1):151–167, September 2003.

[92] Luca Dall'Asta, Ignacio Alvarez-Hamelin, Alain Barrat, Alexei Vázquez, and Alessandro Vespignani. Exploring networks with traceroute-like probes: Theory and simulations. *Theoretical Computer Science*, 355(1):6–24, April 2006.

[93] David L. Davies and Donald W. Bouldin. A cluster separation measure. IEEE *Transactions on Pattern Analysis and Machine Intelligence*, 1(4):224–227, 1979.

[94] Alessandro P. S. de Moura, Adilson E. Motter, and Celso Grebogi. Searching in small-world networks. *Physical Review E*, 68(3):036106, September 2003.

[95] Narsingh Deo and Pankaj Gupta. Sampling the Web graph with random walks. *Congressus Numerantium*, 149:65–73, December 2001.

[96] Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag GmbH, New York, NY, USA, third edition, July 2005.

[97] Edsger W. Dijkstra. Self stabilizing systems in spite of distributed control. *Communications of the Association of the Computing Machinery*, 17:643–644, 1974.

[98] Stephen Dill, S. Ravi Kumar, Kevin S. McCurley, Sridhar Rajagopalan, D. Sivakumar, and Andrew Tomkins. Self-similarity in the web. ACM *Transactions on Internet Technology*, 2(3):205–223, August 2002.

[99] Chris Ding and Xiaofeng He. Linearized cluster assignment via spectral ordering. In *Proceedings of the Twenty-First International Conference on Machine Learning*, pages 30–37, New York, NY, USA, 2004. ACM Press.

[100] Shlomi Dolev. *Self-Stabilization*. The MIT Press, Cambridge, MA, USA, 2000.

[101] Sergey N. Dorogovtsev, A. V. Goltsev, and José Ferreira F. Mendes. Pseudofractal scale-free web. *Physical Review E*, 65(6):066122, June 2002.

[102] Sergey N. Dorogovtsev and José Ferreira F. Mendes. Evolution of networks. *Advances in Physics*, 51(4):1079–1187, 2002.

[103] Sergey N. Dorogovtsev and José Ferreira F. Mendes. *Evolution of Networks: From Biological Nets to the Internet and* WWW. Oxford University Press, Oxford, UK, January 2003.

[104] Peter G. Doyle and J. Laurie Snell. *Random Walks and Electric Networks*. Mathematical Association of America, Washington, DC, USA, 1984.

[105] Philippe Duchon, Nicolas Hanusse, Emmanuelle Lebhar, and Nicolas Schabanel. Could any graph be turned into a small-world? *Theoretical Computer Science*, 355(1):96–103, April 2006.

[106] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. John Wiley & Sons, Inc., New York, NY, USA, second edition, 2001.

[107] Michael Elkin. A faster distributed protocol for constructing a minimum spanning tree. In J. Ian Munro, editor, *Proceedings of the Fifteenth Annual* ACM-SIAM *Symposium on Discrete Algorithms (*SODA*)*, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics.

[108] David Eppstein. Subgraph isomorphism in planar graphs and related problems. *Journal of Graph Algorithms and Applications*, 3(3):1–27, 1999.

[109] Pál Erdős and Alfréd Rényi. On random graphs i. In *Selected Papers of Alfréd Rényi*, volume 2, pages 308–315. Akadémiai Kiadó, Budapest, Hungary, 1976. First publication in Publ. Math. Debrecen 1959.

[110] Pál Erdős and Alfréd Rényi. On the evolution of random graphs. In *Selected Papers of Alfréd Rényi*, volume 2, pages 482–525. Akadémiai Kiadó, Budapest, Hungary, 1976. First publication in MTA Mat. Kut. Int. Közl. 1960.

[111] Deborah Estrin, Ramesh Govindan, John Heidemann, and Satish Kumar. Next century challenges: Scalable coordination in sensor networks. In Harel Kodesh, Victor Bahl, Tomasz Imielinski, and Martha Steenstrup, editors, MOBICOM*'99: Proceedings of the Fifth Annual* ACM/IEEE *International Conference on Mobile Computing and Networking*, pages 263–270, New York, NY, USA, August 1999. ACM Press.

[112] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the Internet topology. In *Proceedings of the* ACM SIGCOMM*'99 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 251–262, New York, NY, USA, 1999. ACM Press.

[113] Michalis Faloutsos and Mart Molle. What features really make distributed minimum spanning tree algorithms efficient? In *International Conference on Parallel and Distributed Systems*, pages 106–114, Washington, DC, USA, 1996. IEEE.

[114] Laura Marie Feeney and Martin Nilsson. Investigating the energy consumption of a wireless network interface in ad hoc networking environment. In *Proceedings of the Twentieth Annual Joint Conference of the* IEEE *Computer and Communications Societies (*INFOCOM*)*, pages 1548–1557, 2001.

[115] Mirtha-Lina Fernández and Gabriel Valiente. A graph distance measure combining maximum common subgraph and minimum common supergraph. *Pattern Recognition Letters*, 22(6–7):753–758, 2001.

[116] Ramon Ferrer i Cancho and Ricard V. Solé. Optimization in complex networks. In Romualdo Pastor-Satorras, Miguel Rubi, and Albert Diaz-Guilera, editors, *Statistical Mechanics of Complex Networks*, volume 625 of *Lecture Notes in Physics*, pages 114–126, Berlin, Germany, 2003. Springer-Verlag GmbH.

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

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

[119] James Allen Fill. An interruptible algorithm for perfect sampling via Markov chains. *Annals of Applied Probability*, 8:131–162, 1998.

[120] Gary William Flake, Steve Lawrence, and C. Lee Giles. Efficient identification of Web communities. In *Proceedings of the Sixth* ACM SIGKDD *International Conference on Knowledge Discovery and Data Mining*, pages 150–160, New York, NY, USA, 2000. ACM Press.

[121] Gary William Flake, Steve Lawrence, C. Lee Giles, and Frans M. Coetzee. Self-organization and identification of Web communities. IEEE *Computer*, 35(3):66–71, 2002.

[122] Pasquale Foggia, Carlo Sansone, and Mario Vento. A performance comparison of five algorithms for graph isomorphism. In *Proceedings of the Third* IAPR *TC-15 International Workshop on Graph-based Representation in Pattern Recognition*, pages 188–199, 2001.

[123] Chris Fraley and Adrian E. Raftery. How many clusters? Which clustering method? Answers via model-based cluster analysis. *The Computer Journal*, 41(8):578–588, 1998.

[124] Jeremy Frank, Ian P. Gent, and Toby Walsh. Asymptotic and finite size parameters for phase transitions: Hamiltonian circuit as a case study. *Information Processing Letters*, 65(5), 1998.

[125] Jeremy Frank and Charles U. Martel. Phase transitions in the properties of random graphs. In *Studying and Solving Really Hard Problems Workshop, in association with the First International Conference on Principles and Practice of Constraint Programming*, 1995.

[126] Felix C. Gaertner. A survey of self-stabilizing spanning-tree construction algorithms. Technical Report 200338, School of Computer and Communication Sciences, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, June 2003.

[127] Robert G. Gallager, Pierre A. Humblet, and Philip M. Spira. A distributed algorithm for minimum-weight spanning trees. ACM *Transactions Programming Languages and Systems*, 5:66–77, January 1983.

[128] Niloy Ganguly, Geo Canright, and Andreas Deutsch. Design of an efficient search algorithm for P2P networks using concepts from natural immune systems. In Özalp Babaoglu, Márk Jelasity, Alberto Montresor, Christof Fetzer, Stefano Leonardi, Aad P. A. van Moorsel, and Maarten van Steen, editors, *Self-star Properties in Complex Information Systems, Conceptual and Practical Foundations*, volume 3460 of *Lecture Notes in Computer Science*, pages 358–372, Berlin, Germany, 2005. Springer-Verlag GmbH.

[129] Juan A. Garay, Shay Kutten, and David Peleg. A sublinear time distributed algorithm for minimum-weight spanning trees. SIAM *Journal on Computing*, 27(1):302–316, February 1998.

[130] Michael R. Garey and David S. Johnson. *Computers and Intractability A Guide to the Theory of NP-Completeness.* W. H. Freeman, San Francisco, CA, USA, 1979.

[131] Bernd Gärtner and Emo Welzl. Random sampling in geometric optimization: new insights and applications. In *Proceedings of the Sixteenth Annual Symposium on Computational Geometry*, pages 91–99, New York, NY, USA, 2000. ACM Press.

[132] Ian P. Gent, Holger H. Hoos, Patrick Prosser, and Toby Walsh. Morphing: Combining structure and randomness. In AAAI/IAAI *99: Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on on Innovative Applications of Artificial Intelligence*, pages 654–660, Menlo Park, CA, USA, 1999. AAAI Press/ The MIT Press.

[133] E. N. Gilbert. Random graphs. *Annals of Mathematical Statistics*, 30(4):1141–1144, December 1959.

[134] Christos Gkantsidis, Milena Mihail, and Amin Saberi. Hybrid search schemes for unstructured peer-to-peer networks. In *Proceedings of the Twenty-fourth Annual Joint Conference of the* IEEE *Computer and Communications Societies (*INFOCOM*)*, volume 3, pages 1526–1537, Los Alamitos, CA, USA, 2005. IEEE Computer Society Press.

[135] Christos Gkantsidis, Milena Mihail, and Ellen Zegura. Spectral analysis of Internet topologies. In *Proceedings of the Twenty-second Annual Joint Conference of the* IEEE *Computer and Communications Societies (*INFOCOM*)*, pages 364–374, New York, NY, USA, 2003. IEEE.

[136] Fred Glover. Tabu search — part I. ORSA *Journal on Computing*, 1(3):190–206, 1989.

[137] Kwang-Il Goh, B. Kahng, and D. Kim. Spectra and eigenvectors of scale-free networks. *Physical Review E*, 64(5):051903, 2001.

[138] Geoffrey R. Grimmett and David R. Stirzaker. *Probability and Random Processes*. Oxford University Press, Oxford, UK, third edition, June 2001.

[139] Stephen Guattery and Gary L. Miller. On the quality of spectral separators. SIAM *Journal on Matrix Analysis and Applications*, 19(3):701–719, 1998.

[140] Sudipto Guha, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. Clustering data streams. In *Proceedings of the Fourty-first Annual Symposium on Foundations of Computer Science (FOCS)*, pages 359–366, Los Alamitos, CA, USA, 2000. IEEE Computer Society Press.

[141] Gaurav Gupta and Mohamed Younis. Performance evaluation of load-balanced clustering in wireless sensor networks. In *Proceedings of Tenth International Conference on Telecommunications*. IEEE, February 2003.

[142] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, Cambridge, UK, 1997.

[143] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann Publishers, 2000.

[144] J. A. Hartigan and M. A. Wong. Algorithm AS 136: A k-means clustering algorithm. *Applied Statistics*, 28:100–108, 1979.

[145] Erez Hartuv and Ron Shamir. A clustering algorithm based on graph connectivity. *Information Processing Letters*, 76(4–6):175–181, December 2000.

[146] Refael Hassin and Arie Tamir. On the minimum diameter spanning tree problem. *Information Processing Letters*, 53(2):109–111, 1995.

[147] Xiaofeng He, Hongyuan Zha, Chris H. Q. Ding, and Horst D. Simon. Web document clustering using hyperlink structures. *Computational Statistics & Data Analysis*, 41(1):19–45, 2002.

[148] Monika R. Henzinger. Algorithmic challenges in web search engines. *Internet Mathematics*, 1(1):115–126, 2004.

[149] Monika R. Henzinger, Allan Heydon, Michael Mitzenmacher, and Marc Najork. On near-uniform URL sampling. In *Proceedings of the Ninth International World-Wide Web Conference*, pages 295–308. Elsevier Science, 2000.

[150] Carlos P. Herrero. Self-avoiding walks on scale-free networks. *Physical Review E*, 71(1):016103, 2005.

[151] Džena Hidović and Marcello Pelillo. Metrics for attributed graphs based on the maximal similarity common subgraph. *International Journal on Pattern Recognition and Artificial Intelligence*, 18(3):299–313, 2004.

[152] Adel Hlaoui and Shengrui Wang. A new algorithm for inexact graph matching. In *Proceedings of the Sixteenth International Conference on Pattern Recognition*, volume 4, pages 180–183, Washington, DC, USA, 2002. IEEE Computer Society Press.

[153] Lawrence B. Holder, Diane J. Cook, and Surnjani Djoko. Substructure discovery in the SUBDUE system. In Usama M. Fayyad and Ramasamy Uthurusamy, editors, *Knowledge Discovery in Databases: Papers from the 1994* AAAI *Workshop, Seattle, Washington, July 1994*, pages 169–180, Menlo Park, CA, USA, 1994. AAAI Press.

[154] Petter Holme and Beom Jun Kim. Growing scale-free networks with tunable clustering. *Physical Review E*, 65(2):026107, February 2002.

[155] Klaus Holzapfel, Sven Kosub, Moritz G. Maaß, and Hanjo Täubig. The complexity of detecting fixed-density clusters. In R. Petreschi, G. Persiano, and R. Silvestri, editors, *Proceedings of the Fifth Italian Conference on Algorithms and Complexity (*CIAC*'03)*, volume 2653 of *Lecture Notes in Computer Science*, pages 201–212, Berlin, Germany, May 2003. Springer-Verlag GmbH.

[156] John E. Hopcroft, Omar Khan, Brian Kulis, and Bart Selman. Natural communities in large linked networks. In *Proceedings of the ninth* ACM SIGKDD *International Conference on Knowledge Discovery and Data Mining*, pages 541–546, New York, NY, USA, 2003. ACM.

[157] John E. Hopcroft and Robert Endre Tarjan. Algorithm 447: efficient algorithms for graph manipulation. *Communications of the Association of the Computing Machinery*, 16(6):372–378, June 1973.

[158] M. Impett, M. S. Corson, and V. Park. A receiver-oriented approach to reliable broadcast ad hoc networks. In *Proceedings of Wireless Communications and Networking Conference (*WCNC *2000)*, volume 1, pages 117–122, September 2000.

[159] Internet Movie Databse Inc. The Internet movie database. `http://www.imdb.com/`.

[160] Akihiro Inokuchi, Takashi Washio, and Hiroshi Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *Proceedings of the Fourth European Conference on Principles of Data Mining and Knowledge Discovery*, volume 1910 of *Lecture Notes in Computer Science*, pages 13–23, London, UK, 2000. Springer-Verlag GmbH.

[161] Anil K. Jain, M. Narasimha Murty, and Patrick J. Flynn. Data clustering: a review. ACM *Computing Surveys*, 31(3):264–323, September 1999.

[162] Vojtech Jarník. O jistem problemu minimalnim. *Praca Moravske Prirodovedecke Spolecnosti*, 6:57–63, 1930.

[163] Glen Jeh and Jennifer Widom. SimRank: a measure of structural-context similarity. In *Proceedings of the Eighth* ACM SIGKDD *International Conference*, pages 538–543, New York, NY, USA, 2002. ACM Press.

[164] Glen Jeh and Jennifer Widom. Mining the space of graph properties. In *Proceedings of the Tenth* ACM SIGKDD *International Conference*, pages 187–196, New York, NY, USA, 2004. ACM Press.

[165] Birgit Jenner, Johannes Köbler, Pierre McKenzie, and Jacobo Torán. Completeness results for graph isomorphism. *Journal of Computer and System Sciences*, 66(3):549–566, May 2003.

[166] Minseok Jeong and Bomson Lee. Comparison between path-loss prediction models for wireless telecommunication system design. In *Proceedings of the* IEEE *International Symposium of Antennas & Propagation Society*, volume 2, pages 186–189, New York, NY, USA, 2001. IEEE.

[167] Mark Jerrum and Alistair Sinclair. The Markov chain Monte Carlo method: An approach to approximate counting and integration. In Dorit Hochbaum, editor, *Approximations for NP-hard Problems*, chapter 12, pages 482–520. PWS Publishing, Boston, MA, USA, 1996.

[168] David Johnson, Anuj Mehrotra, and Michael Trick. COLOR02/03/04: Graph coloring and its generalizations. At `http://mat.gsia.cmu.edu/COLOR02/`, accessed May 11, 2005.

[169] David S. Johnson, Jan Karel Lenstra, and A. H. G. Rinnooy Kan. The complexity of the network design problem. *Networks*, 8:279–285, 1978.

[170] Ellis J. L. Johnson, Anuj Mehrotra, and George L. Nemhauser. Min-cut clustering. *Mathematical Programming*, 62(1):133–151, October 1993.

[171] Valen E. Johnson. Studying convergence of Markov Chain Monte Carlo algorithms using coupled sample paths. *Journal of the American Statistical Association*, 91(433):154–166, 1996.

[172] Christine E. Jones, Krishna M. Sivalingam, Prathima Agrawal, and Jyh Cheng Chen. A survey of energy efficient network protocols for wireless networks. *Wireless Networks*, 7(4):343–358, August 2001.

[173] Nabil Kahale. A semidefinite bound for mixing rates of Markov chains. *Random Structures and Algorithms*, 11(4):299–313, 1998.

[174] Farouk Kamoun and Leonard Kleinrock. Stochastic performance evaluation of hierarchical routing for large networks. *Computer Networks*, 3:337–353, November 1979.

[175] Ravi Kannan, Santosh Vempala, and Adrian Vetta. On clusterings — good, bad and spectral. *Journal of the* ACM, 51(3):497–515, 2004.

[176] Yaron Kanza, Werner Nutt, and Yehoshua Sagiv. Querying incomplete information in semistructured data. *Journal of Computer and System Sciences*, 64(3):655–693, 2002.

[177] David R. Karger. Random sampling in cut, flow, and network design problems. In *Proceedings of the Twenty-sixth Annual* ACM *Symposium on Theory of Computing (*STOC*)*, pages 648–657, New York, NY, USA, 1994. ACM Press.

[178] David R. Karger. *Random Sampling in Graph Optimization Problems*. PhD thesis, Stanford University, Stanford, CA, USA, 1995.

[179] David R. Karger, Philip N. Klein, and Robert E. Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *Journal of the* ACM, 42(2):321–328, March 1995.

[180] Ismo Kärkkäinen and Pasi Fränti. Minimization of the value of Davies-Bouldin index. In Juan J. Villanueva, editor, *Proceedings of the* IASTED *International Conference on Signal Processing and Communications (*SPC*'00)*, pages 426–432, Calgary, AB, Canada, 2000. Acta Press.

[181] Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a Symposium on the Complexity of Computer Computations*, pages 85–103, New York, NY, USA, 1972. Plenum Press.

[182] Rajesh Kasturirangan. Multiple scales in small-world networks. Technical Report AIM-1663, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Cambridge, MA, USA, December 1999.

[183] David Kempe and F. McSherry. A decentralized algorithm for spectral analysis. In *Proceedings of the Thirty-sixth* ACM *Symposium on Theory of Computing (*STOC*)*, New York, NY, USA, 2004. ACM.

[184] Beom Jun Kim, Chang No Yoon, Seung Kee Han, and Hawoong Jeong. Path finding strategies in scale-free networks. *Physical Review E*, 65(2):027103, 2002.

[185] Sun Kim. Graph theoretic sequence clustering algorithms and their applications to genome comparison. In *Computational Biology and Genome Informatics*, chapter 4. World Scientific Publishing Company, Singapore, 2003.

[186] Scott Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[187] Raymond W. Klein and Richard C. Dubes. Experiments in projection and clustering by simulated annealing. *Pattern Recognition*, 22(2):213–220, 1989.

[188] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the* ACM, 46(5):604–632, September 1999.

[189] Jon M. Kleinberg. The small-world phenomenon: an algorithmic perspective. In *Proceedings of the Thirty-second Annual* ACM *Symposium on Theory of Computing* (STOC), pages 163–170, New York, NY, USA, 2000. ACM Press.

[190] Jon M. Kleinberg, S. Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew S. Tomkins. The Web as a graph: Measurements, models, and methods. In T. Asano, H. Imai, D.T. Lee, S. Nakano, and T. Tokuyama, editors, *Proceedings of the Fifth Annual International Conference on Computing and Combinatorics*, volume 1627 of *Lecture Notes in Computer Science*, Berlin, Germany, 1999. Springer-Verlag GmbH.

[191] Jon M. Kleinberg and Steve Lawrence. The structure of the web. *Science*, 294(5548):1849–1850, November 2001.

[192] Leonard Kleinrock and Farouk Kamoun. Hierarchical routing for large networks: Performance evaluation and optimization. *Computer Networks*, 1(3):155–174, 1977.

[193] Donald L. Kreher and Douglar R. Stinson. *Combinatorial Algorithms: Generation, Enumeration, and Search*. CRC Press, Boca Raton, FL, USA, 1998.

[194] P. Krishna, Nitin H. Vaidya, Mainak Chatterjee, and Dhiraj K. Pradhan. A cluster-based approach for routing in dynamic networks. ACM SIGCOMM *Computer Communication Review*, 27(2):49–64, April 1997.

[195] Balachander Krishnamurthy and Jia Wang. On network-aware clustering of web clients. In *Proceeings of* SIGCOMM, pages 97–110, 2000.

[196] Vaishnavi Krishnamurty, Junhong Sun, Michalis Faloutsos, and Sudhir Tauro. Sampling Internet topologies: How small can we go? In Hamid R. Arabnia and Youngsong Mun, editors, *International Conference on Internet Computing*, pages 577–580, Las Vegas, NV, USA, 2003. CSREA Press.

[197] Florent Krząkała, Andrea Pagnani, and Martin Weigt. Threshold values, stability analysis and high-q asymptotics for the coloring problem on random graphs. *Physical Review E*, 70(4):046705, October 2004.

[198] S. Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, D. Sivakumar, Andrew Tomkins, and Eli Upfal. Stochastic models for the Web graph. In *Proceedings of the Fourty-first Annual Symposium on Foundations of Computer Science* (FOCS), pages 57–65, Los Alamitos, CA, USA, 2000. IEEE Computer Society Press.

[199] Michihiro Kuramochi and George Karypis. Freqent subgraph discovery. In *Proceedings of the* IEEE *International Conference on Data Mining*, pages 313–320, Washington, DC, USA, 2001. IEEE Computer Society Press.

[200] Michihiro Kuramochi and George Karypis. Finding frequent patterns in a large sparse graph. Technical Report 03-039, Department of Computer Science and Engineering / Digital Technology Center / Army HPC Research Center, University of Minnesota, Minneapolis, MN, USA, September 2003.

[201] Kim S. Larsen. Amortized constant relaxed rebalancing using standard rotations. *Acta Informatica*, 35(10):859–874, 1998.

[202] Gregory S. Lauer. Hierarchical routing design for SURAN. In *Proceedings of the* IEEE *International Conference on Communications (*ICC*)*, pages 93–102, Los Alamitos, CA, USA, 1986. IEEE Computer Society Press.

[203] Steve Lawrence and C. Lee Giles. Accessibility of information on the web. *Nature*, 400:117–119, July 1999.

[204] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: Densification laws, shrinking diameters and possible explanations. In Robert Grossman, Roberto Bayardo, and Kristin Bennett, editors, *Proceedings of the Eleventh* ACM SIGKDD *International Conference on Knowledge Discovery and Data Mining*, pages 251–262, New York, NY, USA, 2005. ACM Press.

[205] Mark Levene, Trevor Fenner, George Loizou, and Richard Wheeldon. A stochastic model for the evolution of the web. *Computer Networks*, 39(3):277–287, 2002.

[206] Michael Ley. Computer science bibliography. Universität Trier, `http://www.informatik.uni-trier.de/~ley/db/`.

[207] Ning Li, Jennifer C. Hou, and Lui Sha. Design and analysis of an MST-based topology control algorithm. In *Proceedings of the Twenty-second Annual Joint Conference of the* IEEE *Computer and Communications Societies (*INFOCOM*)*, volume 3, pages 1702–1712, New York, NY, USA, April 2003. IEEE.

[208] Hyojun Lim and Chongkwon Kim. Flooding in wireless ad hoc networks. *Computer Communications Journal*, 24(3–4):353–363, 2001.

[209] Chunhung Richard Lin and Mario Gerla. Adaptive clustering for mobile wireless networks. IEEE *Journal on Selected Areas in Communications*, 15(7):1265–1275, September 1997.

[210] S. Lin. Computer solutions of the traveling salesman problem. *Bell Systems Technical Journal*, 44:2245–2269, 1965.

[211] Wei Lou and Jie Wu. On reducing broadcast redundancy in ad hoc wireless networks. IEEE *Transactions on Mobile Computing*, 1(2):111–123, April 2002.

[212] Wei Lou and Jie Wu. Double-covered broadcast (DCB): A simple reliable broadcast algorithm in manets. In *Proceedings of the Twenty-third Annual Joint Conference of the* IEEE *Communications Society* (INFOCOM), volume 3, pages 2084–2095, Los Alamitos, CA, USA, 2004. IEEE Computer Society Press.

[213] László Lovász and Ravi Kannan. Faster mixing via average conductance. In *Proceedings of the Thirty-first Annual* ACM *Symposium on Theory of Computing* (STOC), pages 282–287, New York, NY, USA, 1999. ACM Press.

[214] Ricard Lowry. *Concepts and Applications of Inferential Statistics*. Vassar College, Poughkeepsie, NY, USA, 1999–2005. Online publication; available at `http://faculty.vassar.edu/lowry/webtext.html`.

[215] Gin Lv, Pei Cao, Edith Cohan, Kai Li, and Scott Shenker. Search and replication in unstructured peer-to-peer networks. In *Proceedings of the Sixteenth Annual* ACM *International Conference on Supercomputing*, pages 84–95, New York, NY, USA, 2002. ACM.

[216] Gurmeet Singh Manku, Moni Naor, and Udi Wieder. Know thy neighbor's neighbor: The power of lookahead in randomized P2P networks. In *Proceedings of the Thirty-Sixth Annual* ACM *Symposium on Theory of Computing* (STOC), pages 54–63, New York, NY, USA, 2004. ACM Press.

[217] Olivier C. Martin, Rémi Monasson, and Riccardo Zecchina. Statistical mechanics methods and phase transitions in optimization problems. *Theoretical Computer Science*, 265(1–2):3–67, August 2001.

[218] Hideo Matsuda, Tatsuya Ishihara, and Akihiro Hashimoto. Classifying molecular sequences using a linkage graph with their pairwise similarities. *Theoretical Computer Science*, 210(2):305–325, January 1999.

[219] Steven McCanne, Sally Floyd, Kevin Fall, and Kannan Varadhan. The network simulator `ns-2`. The VINT project, available for download at `http://www.isi.edu/nsnam/ns/`.

[220] Sally McClean, Bryan Scotney, and Kieran Greer. A scalable approach to integrating heterogeneous aggregate views of distributed databases. IEEE *Transactions on Knowledge and Data Engineering*, 15(1):232–235, Jan./Feb. 2003.

[221] José Ferreira F. Mendes. Effect of accelerated growth on networks dynamics. In Romualdo Pastor-Satorras, Miguel Rubi, and Albert Diaz-Guilera, editors, *Statistical Mechanics of Complex Networks*, volume 625 of *Lecture Notes in Physics*, pages 88–113, Berlin, Germany, 2003. Springer-Verlag GmbH.

[222] Bruno T. Messmer and Horst Bunke. Subgraph isomorphism in polynomial time. Technical Report TR-IAM-95-003, Institute of Computer Science and Applied Mathematics, University of Bern, Bern, Switzerland, 1995.

[223] Milena Mihail, Christos Gkantsidis, Amin Saberi, and Ellen Zegura. On the semantics of Internet topologies. Technical Report GIT-CC-02-07, College of Computing, Georgia Institute of Technology, Atlanta, GA, USA, 2002.

[224] George A. Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine J. Miller. Introduction to wordnet: An on-line lexical database. *International Journal of Lexicography*, 3(4):235–244, 1990.

[225] Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. Network motifs: Simple building blocks of complex networks. *Science*, 298(5594):824–827, October 2002.

[226] Jan ming Ho, D. T. Lee, Chia-Hsiang Chang, and C. K. Wong. Bounded diameter minimum spanning trees and related problems. In *Proceedings of the Fifth Annual Symposium on Computational Geometry*, pages 276–282, New York, NY, USA, June 1989. ACM Press.

[227] Jan ming Ho, D. T. Lee, Chia-Hsiang Chang, and C. K. Wong. Minimum diameter spanning trees and related problems. SIAM *Journal on Computing*, 20(5):987–997, 1991.

[228] David Mitchell, Bart Selman, and Hector Levesque. Hard and easy distributions of SAT problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (*AAAI*-92)*, pages 459–465, Cambridge, MA, USA, 1992. AAAI Press / The MIT Press.

[229] Michael Mitzenmacher. A brief history of generative models for power law and lognormal distributions. *Internet Mathematics*, 1(2):226–251, 2004.

[230] Mike Molloy and Bruce Reed. A critical point for random graphs with a given degree sequence. *Random Structures and Algorithms*, 6:161–180, 1995.

[231] Cristopher Moore and Mark E. J. Newman. Epidemics and percolation in small-world networks. *Physical Review E*, 61(5):5678–5682, May 2000.

[232] Harsha Nagesh, Sanjay Goil, and Alok Choudhary. Parallel algorithms for clustering high-dimensional large-scale datasets. In Robert L. Grossman, Chandrika Kamath, W. Philip Kegelmeyer, Vipin Kumar, and Raju R. Namburu, editors, *Data Mining for Scientific and Engineering Applications*, volume 2 of *Massive Computing*, chapter 19. Kluwer Academic Publishers, Boston, MA, USA, September 2001.

[233] Michel Neuhaus and Horst Bunke. Self-organizing graph edit distance. In Edwin Hancock and Mario Vento, editors, *Proceedings of the Fourth* IAPR *International Workshop on Graph Based Representations in Pattern Recognition*, volume 2726 of *Lecture Notes in Computer Science*, pages 83–94, New York, NY, USA, 2003. Springer-Verlag GmbH.

[234] Mark E. J. Newman. The structure of scientific collaboration networks. *Proceedings of the National Academy of Sciences,* USA, 98(2):404–409, January 2001.

[235] Mark E. J. Newman. A measure of betweenness centrality based on random walks. Technical Report cond-mat/0309045, arXiv.org e-Print archive, `http://arxiv.org/`, 2003.

[236] Mark E. J. Newman. Properties of highly clustered networks. *Physical Review E*, 68(2):026121, August 2003.

[237] Mark E. J. Newman. The structure and function of complex networks. SIAM *Review*, 45(2):167–256, 2003.

[238] Mark E. J. Newman. Detecting community structure in networks. *European Physical Journal B*, 38(2):321–330, 2004.

[239] Mark E. J. Newman. Fast algorithm for detecting community structure in networks. *Physical Review E*, 69(6):066133, 2004.

[240] Mark E. J. Newman and Michelle Girvan. Finding and evaluating community structure in networks. Technical Report cond-mat/0308217, arXiv.org e-Print archive, `http://arxiv.org/`, August 2003.

[241] Mark E. J. Newman and Michelle Girvan. Mixing patterns and community structure in networks. In Romualdo Pastor-Satorras, Miguel Rubi, and Albert Diaz-Guilera, editors, *Statistical Mechanics of Complex Networks*, volume 625 of *Lecture Notes in Physics*, pages 66–87, Berlin, Germany, 2003. Springer-Verlag GmbH.

[242] Mark E. J. Newman, Cristopher Moore, and Duncan J. Watts. Mean-field solution of the small-world network model. *Physical Review Letters*, 84(14):3201–3204, April 2000.

[243] Mark E. J. Newman and Duncan J. Watts. Scaling and percolation in the small-world network model. *Physical Review E*, 60(6):7332–7342, 1999.

[244] Phu Chien Nguyen, Takashi Washio, Kouzou Ohara, and Hiroshi Motoda. Using a hash-based method for apriori-based graph mining. In Jean-François Boulicaut, Floriana Esposito, Fosca Giannotti, and Dino Pedreschi, editors, *Proceedings of the Eighth European Conference on Principles and Practice of Knowledge Discovery in Databases*, volume 3202 of *Lecture Notes in Computer Science*, pages 349–361, Berlin, Germany, 2004. Springer-Verlag GmbH.

[245] Sze-Yao Ni, Yu-Chee Tseng, Yuh-Shyan Chen, and Jang-Ping Sheuu. The broadcast storm problem in a mobile ad hoc network. In Harel Kodesh, Victor Bahl, Tomasz Imielinski, and Martha Steenstrup, editors, MOBICOM'99: *Proceedings of the Fifth Annual* ACM/IEEE *International Conference on Mobile Computing and Networking*, pages 151–162, New York, NY, USA, August 1999. ACM Press.

[246] Jorge Nuevo. Mobility generator program for NS-2, 2002. Available for download at `http://externe.inrs-emt.uquebec.ca/users/nuevo/NSmobgenerator.htm`.

[247] Liadan O'Callaghan, Nina Mishra, Adam Meyerson, Sudipto Guha, and Rajeev Motwani. Streaming-data algorithms for high-quality clustering. In *Proceedings of 18th* IEEE *International Conference on Data Engineering*, pages 685–694, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[248] Tomoyuki Ohta, Shinji Inoue, and Yoshiaki Kakuda. An adaptive multihop clustering scheme for highly mobile Ad Hoc networks. In *Proceedings of the Sixth International Symposium on Autonomous Decentralized Systems (*ISADS*)*, pages 293–302, Washington, DC, USA, 2003. IEEE Computer Society Press.

[249] Pekka Orponen and Satu Elisa Schaeffer. Efficient algorithms for sampling and clustering of large nonuniform networks. Technical Report cond-mat/0406048, arXiv.org e-Print archive, `http://arxiv.org/`, June 2004.

[250] Pekka Orponen and Satu Elisa Schaeffer. Local clustering of large graphs by approximate Fiedler vectors. In Sotiris Nikoletseas, editor, *Proceedings of the Fourth International Workshop on Efficient and Experimental Algorithms (*WEA'05*)*, volume 3505 of *Lecture Notes in Computer Science*, pages 524–533, Berlin/Heidelberg, Germany, 2005. Springer-Verlag GmbH.

[251] Sagar A. Pandit and Ravindra E. Amritkar. Characterization and control of small-world networks. *Physical Review E*, 60(2):R1119–R1122, August 1999.

[252] Vinayaka Pandit and Dhananjay M. Dhamdhere. Self-stabilizing maxima finding on general graphs. Technical report, IBM Research Division, April 2002.

[253] Gopal Pandurangan, Prabhakar Raghavan, and Eli Upfal. Building low-diameter P2P networks. In *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science (*FOCS*)*, pages 492–499, Los Alamitos, CA, USA, 2001. IEEE Computer Society Press.

[254] Romualdo Pastor-Satorras, Alexei Vazquez, and Alessandro Vespignani. Dynamical and correlation properties of the Internet. *Physical Review Letters*, 87(25):258701, December 2001.

[255] Romualdo Pastor-Satorras and Alessandro Vespignani. Epidemic spreading in scale-free networks. *Physical Review Letters*, 86(14):3200–3203, April 2001.

[256] Vern Paxson and Sally Floyd. Why we don't know how to simulate the Internet. In *Proceedings of the 1997 Winter Simulation Conference*, pages 1037–1044, New York, NY, USA, 1997. ACM Press.

[257] Wei Peng and Xicheng Lu. Efficient broadcast in mobile ad hoc networks using connected dominating sets. *Journal of Software*, 12(4):529–536, 2001.

[258] Charles E. Perkins, editor. *Ad Hoc Networking*. Addison Wesley, Reading, MA, USA, 2001.

[259] Alex Pothen, Horst D. Simon, and Kan-Pu Liou. Partitioning sparse matrices with eigenvectors of graphs. SIAM *Journal of Matrix Analysis and Applications*, 11(3):430–452, 1990.

[260] Robert C. Prim. Shortest connection networks and some generalizations. *Bell Systems Technical Journal*, 36(6):1389–1401, November 1957.

[261] James Gary Propp and David Bruce Wilson. How to get a perfectly random sample from a generic Markov chain and generate a random spanning tree of a directed graph. *Journal of Algorithms*, 27(2):170–217, 1988.

[262] James Gary Propp and David Bruce Wilson. Exact sampling with coupled Markov chains and applications to statistical mechanics. *Random Structures and Algorithms*, 9(1&2):223–252, 1996.

[263] Janez Puhan, Tadej Tuma, and Iztok Fajfar. SPICE for Windows 95/98/NT. *Elektrotehnišski vestnik*, 65(5):267–271, 1998. (Electrotechnical review, Ljubljana, Slovenia.).

[264] Jan M. Rabaey. The SPICE circuit simulator. EECS Department of the University of California at Berkeley, `http://bwrc.eecs.berkeley.edu/Classes/ICBook/SPICE/`.

[265] Siddheswar Ray and Rose H. Turi. Determination of number of clusters in k-means clustering and application in colour image segmentation. In N. R. Pal, A. K. De, and J. Das, editors, *Proceedings of the Fourth International Conference on Advances in Pattern Recognition and Digital Techniques (ICAPRDT'99)*, pages 137–143, New Delhi, India, 1999. Narosa Publishing House.

[266] D. G. Rees. *Essential Statistics*, volume 50 of *Texts in Statistical Science Series*. CRC Press, Boca Raton, FL, USA, fourth edition, 2000.

[267] Matei Ripeanu. Peer-to-peer architecture case study: Gnutella network. In *Proceedings of the First International Conference on Peer-to-Peer Computing (P2P'01)*, pages 99–100, Washington, DC, USA, 2001. IEEE Computer Society Press.

[268] Matei Ripeanu, Ian Foster, and Adriana Iamnitchi. Mapping the Gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. IEEE *Internet Computing Journal*, 6(1):50–57, 2002.

[269] Christian P. Robert and George Casella. *Monte Carlo Statistical Methods*. Springer-Verlag GmbH, Heidelberg, Germany, 2004.

[270] Gabriel Robins and Jeffrey S. Salowe. On the maximum degree of minimum spanning trees. In Kurt Mehlhorn, editor, *Proceedings of the Tenth Annual Symposium on Computational Geometry*, pages 250–258, New York, NY, USA, June 1994. ACM Press.

[271] Martin Rosvall, Ala Trusina, Petter Minnhagen, and Kim Sneppen. Networks and cities: An information perspective. *Physical Review Letters*, 94(2):028701, January 2005.

[272] Nick Roussopoulos, Stephen Kelley, and Frédéric Vincent. Nearest neighbor queries. ACM SIGMOD *Record*, 24(2):71–79, 1995.

[273] Cesar A. Santivanez, Ram Ramanathan, and Ioannis Stavrakakis. Making link-state routing scale for ad hoc networks. In *Proceedings of the Second* ACM *International Symposium on Mobile ad hoc Networking & Computing*, pages 22–32, Long Beach, CA, USA, 2001. ACM Press.

[274] Tapan K. Sarkar, Zhong Ji, Kyungjung Kim, Abdellatif Medouri, and Magdalena Salazar-Palma. A survey of various propagation models for mobile communication. IEEE *Antennas and Propagation Magazine*, 45(3):51–82, June 2003.

[275] Nima Sarshar, Oscar Boykin, and Vwani Roychowdhury. Scalable percolation search on complex networks. *Theoretical Computer Science*, 355(1):48–64, April 2006.

[276] Nima Sarshar, P. Oscar Boykin, and Vwani P. Roychowdhury. Percolation search in power law networks: Making unstructured peer-to-peer networks scalable. In *Proceedings of Fourth* IEEE *International Conference on Peer-to-Peer Computing*, pages 2–9, Washington, DC, USA, 2004. IEEE Computer Society Press.

[277] Satu Elisa Schaeffer. Stochastic local clustering for massive graphs. In T. B. Ho, D. Cheung, and H. Liu, editors, *Proceedings of the Ninth Pacific-Asia Conference on Knowledge Discovery and Data Mining (*PAKDD*-05)*, volume 3518 of *Lecture Notes in Computer Science*, pages 354–360, Berlin/Heidelberg, Germany, 2005. Springer-Verlag GmbH.

[278] Satu Elisa Schaeffer. Query-friendly storage for graphs, 2006. In preparation.

[279] Satu Elisa Schaeffer. Spanning trees with small average hop count, 2006. In preparation.

[280] Satu Elisa Schaeffer, Stefano Marinoni, Pekka Nikander, and Mikko Särelä. Dynamic local clustering for ad hoc networks, 2006. Submitted for publication.

[281] Frank J. Seinstra. Time optimal uniform self-stabilizing spanning tree. Technical report, Intelligent Sensory Information Systems, Faculty of Science, University of Amsterdam, Amsterdam, The Netherlands, March 2001.

[282] Bart Selman, Henry A. Kautz, and Bram Cohen. Local search strategies for satisfiability testing. In David S. Johnson and Michael A. Trick, editors, *Cliques, Coloring and Satisfiability: Second* DIMACS *Implementation Challenge*, volume 26 of DIMACS *Series in Discrete Mathematics and Theoretical Computer Science*, pages 521–532, Providence, RI, USA, 1996. American Mathematical Society Press.

[283] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423 & 623–656, July & Oct. 1948.

[284] Jiří Šíma and Satu Elisa Schaeffer. On the NP-completeness of some graph cluster measures. In Jiří Wiedermann, Gerard Tel, Jarslav Pokorný, Mária Bieliková, and Július Štuller, editors, *Proceedings of the Thirty-second International Conference on Current Trends in Theory and Practice of Computer Science (*SOFSEM *06)*, volume 3831 of *Lecture Notes in Computer Science*, pages 530–537, Berlin/Heidelberg, Germany, 2006. Springer-Verlag GmbH.

[285] Herbert A. Simon. On a class of skew distribution functions. *Biometrika*, 42(3/4):425–440, December 1955.

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

[287] Kim Sneppen, Ala Trusina, and Martin Rosvall. Hide-and-seek on complex networks. *Europhysics Letters*, 69(5):853–859, 2005.

[288] Daniel A. Spielman and Shang-Hua Teng. Spectral partitioning works: planar graphs and finite element meshes. In *Proceedings of the Thirty-seventh* IEEE *Symposium on Foundations of Computing (*FOCS*)*, pages 96–105, Los Alamitos, CA, USA, 1996. IEEE Computer Society Press.

[289] Alexandre O. Stauffer and Valmir C. Barbosa. Local heuristics and the emergence of spanning subgraphs in complex networks. *Theoretical Computer Science*, 355(1):80–95, April 2006.

[290] Martha Steenstrup. *Cluster-Based Networks*, chapter 4. Addison Wesley, Reading, MA, USA, 2001.

[291] Ivan Stojmenovic and Xu Lin. Power-aware localized routing in wireless networks. IEEE *Transactions on Parallel and Distributed Systems*, 12(10), 2001.

[292] John Sucec and Ivan 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*, volume 3, pages 1698–1706, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[293] Csaba Szepesvári. Shortest path discovery problems: A framework, algorithms and experimental results. In Deborah L. McGuinness and George Ferguson, editors, *Proceedings of the Nineteenth National Conference on Artificial Intelligence and Sixteenth Conference on Innovative Applications of Artificial Intelligence*, pages 550–555, Menlo Park, CA, USA, 2004. AAAI Press/The MIT Press.

[294] Bosiljka Tadić. Adaptive random walks on the class of Web graphs. *European Physical Journal B*, 23(2):221–228, 2001.

[295] Bosiljka Tadić. Dynamics of directed graphs: the World-Wide Web. *Physica A*, 293(1–2):273–284, 2001.

[296] Bosiljka Tadić. Growth and structure of the World-Wide Web: Towards realistic modeling. *Computer Physics Communications*, 147(1–2):586–589, August 2002.

[297] Gerard Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, Cambridge, UK, second edition, 2000.

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

[299] Luke Tierney. Markov chains for exploring posterior distributions. *Annals of Statistics*, 22:1701–1762, 1994.

[300] Ming-Shin Tsai and Shing-Tsaan Huang. A self-stabilizing algorithm for the shortest paths problem with a fully distributed daemon. *Parallel Processing Letters*, 4(1):65–72, 1994.

[301] J. R. Ullmann. An algorithm for subgraph isomorphism. *Journal of the* ACM, 23(1):31–42, 1976.

[302] Stijn Marinus van Dongen. *Graph Clustering by Flow Simulation*. PhD thesis, Universiteit Utrecht, Utrecht, The Netherlands, 2000.

[303] Basil Vandegriend and Joseph Culberson. The $\mathcal{G}_{n,m}$ phase transition is not hard for the hamiltonian cycle problem. *Journal of Artificial Intelligence Research*, 9:219–245, November 1998.

[304] Alexei Vázquez and Martin Weigt. Computational complexity arising from degree correlations in networks. *Physical Review E*, 67(2):027101, February 2003.

[305] Satu Elisa Virtanen. Clustering the Chilean web. In *Proceedings of the First Latin American Web Congress*, pages 229–231, Los Alamitos, CA, USA, November 2003. IEEE Computer Society.

[306] Satu Elisa Virtanen. Properties of nonuniform random graph models. Research Report A77, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Espoo, Finland, May 2003.

[307] Satu Elisa Virtanen and Pekka Nikander. Local clustering for hierarchical ad hoc networks. In *Proceedings of* WIOPT*'04: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, pages 404–405, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

[308] Danica Vukadinović, Polly Huang, and Thomas Erlebach. On the spectrum and structure of Internet topology graphs. In H. Unger, T Böhme, and A. Mikler, editors, *Proceedings of Second International Workshop on Innovative Internet Computing Systems (*IICS *2002)*, volume 2346 of *Lecture Notes in Computer Science*, pages 83–95, Berlin, Germany, 2002. Springer-Verlag GmbH.

[309] W3C. Wordnet in RDFS and OWL. Technical report, The World-Wide Web Consortium, `http://www.w3.org/`, August 2004. `http://www.w3.org/2001/sw/BestPractices/WNET/wordnet-sw-20040713.html`.

[310] Toby Walsh. Search in a small world. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (*IJCAI*)*, volume 2, pages 1172–1177, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers.

[311] Hsinping Wang, Tsungnan Lin, Chia Hung Chen, and Yennan Shen. Dynamic search in peer-to-peer networks. In *Proceedings of the Thirteenth International World-Wide Web Conference*, New York, NY, USA, 2004. ACM.

[312] Takashi Washio and Hiroshi Motoda. Multi relational data mining (MRDM): State of the art of graph-based data mining. ACM SIGKDD *Explorations Newsletter*, 5(1):59–68, July 2003.

[313] Duncan J. Watts. *Small Worlds*. Princeton University Press, Princeton, NJ, USA, 1999.

[314] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small world' networks. *Nature*, 393(6684):440–442, June 1998.

[315] Bernard M. Waxman. Routing of multipoint connections. IEEE *Journal on Selected Areas in Communications*, 6(9):1617–1622, 1988.

[316] Wei Wei, Jordan Erenrich, and Bart Selman. Towards efficient sampling: Exploiting random walk strategies. In Deborah L. McGuinness and George Ferguson, editors, *Proceedings of the Nineteenth National Conference on Artificial Intelligence and Sixteenth Conference on Innovative Applications of Artificial Intelligence*, pages 670–676, Menlo Park, CA, USA, 2004. AAAI Press/The MIT Press.

[317] Jeffrey E. Wieselthier, Gam D. Nguyen, and Anthony Ephremides. Algorithms for energy-efficient multicasting in static ad hoc wireless networks. *Mobile Networks and Applications*, 6(3):251–263, June 2001.

[318] Jeffrey E. Wieselthier, Gam D. Nguyen, and Anthony Ephremides. On the construction of energy-efficient broadcast and multicast trees in wireless networks. In *Proceedings of the Nineteenth Annual Joint Conference of the* IEEE *Computer and Communications Societies (*INFOCOM*)*, pages 585–594, March 2002.

[319] Wai-Chiu Wong and Ada Wai-chee Fu. Incremental document clustering for web page classification. In J. Qun, editor, *International Conference on Information Society in the 21st century: Emerging Technologies and New Challenges (IS2000)*, Aizu-Wakamatsu, Fukushima, Japan, 2000. The University of Aizu.

[320] Andrew Y. Wu, Michael Garland, and Jiawei Han. Mining scale-free networks using geodesic clustering. In Won Kim, Ron Kohavi, Johannes Gehrke, and William DuMouchel, editors, *Proceedings of the Tenth* ACM SIGKDD *International Conference on Knowledge Discovery and Data Mining*, New York, NY, USA, 2004. ACM Press.

[321] Bang Ye Wu, Giuseppe Lancia, Vineet Bafna, Kun-Mao Chao, R. Ravi, and Chuan Yi Tang. A polynomial time approximation scheme for minimum routing cost spanning trees. In *Proceedings of the Ninth Annual* ACM-SIAM *Symposium on Discrete Algorithms*, pages 21–32, Philadelphia, PA, USA, 1998. Society for Industrial and Applied Mathematics.

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

[323] Xifeng Yan and Jiawei Han. gSpan: Graph-based substructure pattern mining. In *Proceedings of the* IEEE *International Conference on Data Mining*, pages 721–724, Washington, DC, USA, 2002. IEEE Computer Society Press.

[324] Xifeng Yan and Jiawei Han. CloseGraph: mining closed frequent graph patterns. In Lise Getoor, Ted E. Senator, Pedro Domingos, and Christos Faloutsos, editors, *Proceedings of the Ninth* ACM SIGKDD *International Conference on Knowledge Discovery and Data Mining*, pages 286–295, New York, NY, USA, 2003. ACM Press.

[325] Shi-Jie Yang. Exploring complex networks by walking on them. *Physical Review E*, 71(1):016107, 2005.

[326] Wayne W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33:452–473, 1977.

[327] Hui Zhang, Ashish Goel, and Ramesh Govindan. Using the small world model to improve Freenet performance. In *Proceedings of the Twenty-first Annual Joint Conference of the* IEEE *Computer and Communications Societies (*INFOCOM*)*, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[328] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: an efficient data clustering method for very large databases. In *Proceedings of the* ACM SIGMOD *1996*, pages 103–114, New York, NY, USA, 1996. ACM Press.

[329] Han Zhu and Zhuang-Xiong Huang. Navigation in a small world with local information. *Physical Review E*, 70(3):036117, September 2004.

APPENDICES

MATLAB CODE FOR FIEDLER VECTOR CALCULATIONS

Exact Fiedler vectors

```
function fiedler = fiedler_exact(file, pos);

% data should be in the "u v 1" format
% with the vertex for which the calculation is made in pos
% (indexing from 1 to n)

fprintf('The Fiedler vector of vertex %d', pos);
fprintf(' for the graph ''%s''.\n', file);

% load the graph data
data = load(file);

% adjacency matrix
A = spconvert(data);

% vertex count
n = size(A, 1);

fprintf('The graph has %d vertices.\n', n);

% degree vector
deg = sum(A);

% degree matrix
T = sparse(diag(deg));

% the square-root degree matrix
Tsq = sqrt(T);

% identity matrix
I = speye(n);

% transition probabilities
P = I;
for i = (1:n)
  if (deg(i) > 0)
     P(i,1:n) = (1.0/deg(i))*A(i,1:n);
  end;
end
```

```
% making state pos absorbing
Pabs = P;
for i = 1:n
  % no probability for exiting the state
  Pabs(pos, i) = 0.0;
end
% probability 1 for staying in state pos
Pabs(pos, pos) = 1.0;

% 2 largest eigenvalues of Pabs
[vects, ev] = eigs(Pabs, 2, 'lm');

% eigenvector related to the second largest eigenvalue
f = vects(:,2);

% normalization
fv1 = abs(f);
fv1 = fv1/max(fv1);

% another method for achieving the same vector
L = Tsq*Pabs*inv(Tsq);

% 2nd smallest eigenvalue and the related eigenvector
[utilde, e] = eigs(L, 2, 'lm');
u = inv(Tsq)*utilde(:,2);

% normalization
fv2 = abs(u);
fv2 = fv2/max(fv2);

% ensuring that the outcome of the two methods is close
if (max(abs(fv1 - fv2)) > 0.001)
  fprintf('ERROR: significant difference vector.\n');
  fiedler = fv2;
  return;
end;

fiedler = fv1;
return;
```

Approximate Fiedler vectors

```
function fiedler = fiedler_vector(file, pos);

% data should be in the "u v 1" format
% with the vertex for which the calculation is made in pos
% (indexing from 1 to n)

fprintf('An approximate Fiedler vector of vertex %d', pos);
fprintf(' for the graph ''%s''.\n', file);

% load the graph data
data = load(file);

% adjacency matrix
A = spconvert(data);

% vertex count
n = size(A, 1);

% degrees
fprintf('The degrees are\n');
deg = sum(A)
T = sparse(diag(deg));

% the square-root degree matrix
Tsq = sqrt(T);

% transition probabilities
I = speye(n);
P = I;
for i = (1:n)
  if (deg(i) > 0)
     P(i,1:n) = (1.0/deg(i))*A(i,1:n);
  end;
end

% making state pos absorbing
Pabs = P;
for i = 1:n
  Pabs(pos, i) = 0.0;
end
Pabs(pos, pos) = 1.0;
```

```
% 2 largest eigenvalues of Pabs
[vects, ev] = eigs(Pabs, 2);

% eigenvector related to the second largest eigenvalue
fprintf('The fiedler vector is\n');
f = vects(:,2);
f = abs(f)/max(abs(f))

% other method
L = Tsq*Pabs*inv(Tsq);

% largest eigenvalue of L and the related eigenvector
[utilde, e] = eigs(L, 2);

% fiedler vector
fprintf('The fiedler vector obtained by another method is\n');
u = inv(Tsq)*utilde(:,2);
u = abs(u)/max(abs(u))

% comparison that the two are identical

if (max(abs(f - u)) > 0.001)
  fprintf('ERROR: significant difference vector.\n');
end

fiedler = f;

return;
```

URN MODEL FOR BIT-STRING SIMILARITY MEASURES

We consider the following *urn model* to derive the probability that two bit strings differ at a certain position. In urn $\mathcal{U}_1$ there are $\mathcal{W}$ white balls and $\mathcal{B}$ black balls. We denote the total number of balls by $\mathcal{T} = \mathcal{W} + \mathcal{B}$. Another urn $\mathcal{U}_2$ has equally $\mathcal{W}$ white balls and $\mathcal{B}$ black balls.

Assume that a sequence of k draws in pairs has been made from the two urns and all pairs have been of equal color. Denote the number of white balls drawn from urn $\mathcal{U}_i$ at time k by w_k, fixing the number of blacks drawn to $k - w_k$. There are in total $\binom{\mathcal{W}}{w_k}$ ways to choose those white balls and $\binom{\mathcal{B}}{k-w_k}$ ways to choose the black ones. Hence the probability of choosing a certain color-configuration with w_k white balls in a total of k balls is

$$\Pr\left[w_k = w\right] = \frac{\binom{\mathcal{W}}{w}\binom{\mathcal{B}}{k-w}}{\binom{\mathcal{W}+\mathcal{B}}{k}} = \frac{\binom{\mathcal{W}}{w}\binom{\mathcal{B}}{k-w}}{\binom{\mathcal{T}}{k}}, \tag{1}$$

and hence follows the *hypergeometric distribution*. The number of ways to order these balls in a unique color sequence[1] is

$$\binom{w_k + k - w_k + 1 - 1}{w_k} = \binom{k}{w_k}, \tag{2}$$

and hence the probability of a given k-sequence with x white balls is

$$p_{\text{eq}}\left(x, k\right) = \begin{cases} \dfrac{1}{\binom{k}{x}}, & k > 0 \\ 1 & k = 0. \end{cases} \tag{3}$$

where we additionally define that the probability of drawing an equal empty prefix is one. This is the probability of drawing a unique k-sequence with w_k whites from a single urn. The probability for drawing a pre-specified sequence from a single urn at random is the same than the probability of drawing the same sequence from two urns independently.

Using the identity for conditional probability, we have

$$\begin{aligned} &\Pr\left[\text{difference at time } k+1 \mid \text{equal sequences at time } k\right] \\ &\quad = \frac{\Pr\left[\text{first difference at time } k+1\right]}{\Pr\left[\text{equal sequences at time } k\right]}, \end{aligned} \tag{4}$$

which is actually the probability that a draw gives a different-colored pair when the number of blacks balls and white balls in the urns are still the same. The probability of a differing pair when $\mathcal{W}$ is known is

$$p\left(\mathcal{W}, \mathcal{T}\right) = \frac{2\mathcal{W}(\mathcal{T} - \mathcal{W})}{\mathcal{T}^2}, \tag{5}$$

[1] Consider the problem of placing x identical balls into y distinct urns, allowing some of the urns to remain empty. There are $\binom{x+y-1}{x}$ ways to distribute the balls. Now take all the black balls and place them in a row. Consider each slot before, after or between the black balls an urn and distribute the white balls in these slots, obtaining each possible color configuration exactly once.

and from Equation 1 we get the probability that a specific number of white balls *remain* in the two urns; it is the same as probability than that of removing the balls that in were removed. For two urns that hold the same configuration, the probability for drawing a pair of balls with different colors at draw number $(k+1)$ when w_k whites have been previously drawn from each urn is

$$\begin{aligned}&\Pr\left[\text{difference for } k+1 \mid \text{equal for } k \text{ draws, } w_k \text{ of which whites}\right]\\ &\qquad = \frac{2(\mathcal{W}-w_k)(\mathcal{B}-k+w_k)}{(\mathcal{T}-k)^2}.\end{aligned} \tag{6}$$

Applying total probability theorem over the color configurations, we get the probability that the two sequences *first* differ at position $k \in [1, N]$ as

$$\begin{aligned}&q\left(\mathcal{W},\mathcal{T},k\right) =\\ &\sum_{x=s}^{t}\left(p_{\text{eq}}\left(x,k-1\right)\cdot\frac{2(\mathcal{W}-x)(\mathcal{B}-k+x)}{(\mathcal{T}-k)^2}\cdot\frac{\binom{\mathcal{W}}{x}\binom{\mathcal{B}}{k-x}}{\binom{\mathcal{T}}{k}}\right),\end{aligned} \tag{7}$$

where $s = \max\{0, \mathcal{W}+k-\mathcal{T}\}$, $t = \min\{\mathcal{W}, k\}$, and $\mathcal{B} = \mathcal{T}-\mathcal{W}$. Treating the ones as the black balls and the zeroes as the white balls or vice versa, we now have an expression for the probability that two strings first differ at position k.

INDEX

HELSINKI UNIVERSITY OF TECHNOLOGY LABORATORY FOR THEORETICAL COMPUTER SCIENCE
RESEARCH REPORTS

ISBN 951-22-8118-X
ISSN 1457-7615