Graph-Based Natural Language Processing and Information Retrieval

Rada Mihalcea
Dragomir Radev
This page intentionally left blank
Graph theory and the fields of natural language processing and information retrieval are well-studied disciplines. Traditionally, these areas have been perceived as distinct, with different algorithms, different applications, and different potential end-users. However, recent research has shown that these disciplines are intimately connected, with much variety in the way that natural language processing and information retrieval applications find efficient solutions within graph-theoretical frameworks.

This book is a comprehensive description of the use of graph-based algorithms for natural language processing and information retrieval. It brings together topics as diverse as lexical semantics, text summarization, text mining, ontology construction, text classification, and text retrieval, which are connected by the common underlying theme of the use of graph-theoretical methods for text- and information-processing tasks. Readers will gain a firm understanding of the major methods and applications in natural language processing and information retrieval that rely on graph-based representations and algorithms.

Rada Mihalcea is an Associate Professor in the Department of Computer Science and Engineering at the University of North Texas, where she leads the Language and Information Technologies research group. In 2009, she received the Presidential Early Career Award for Scientists and Engineers, awarded by President Barack Obama. She served on the editorial board of several journals, including *Computational Linguistics, Journal of Natural Language Engineering*, and *Language Resources and Evaluations*, and she cochaired the Empirical Methods in Natural Language Processing Conference in 2009 and the Association for Computational Linguistics Conference in 2011. She has been published in *IEEE Intelligent Systems, Journal of Natural Language Engineering, Journal of Machine Translation, Computational Intelligence, International Journal of Semantic Computing*, and *Artificial Intelligence Magazine*.

Dragomir Radev is a Professor in the School of Information, the Department of Electrical Engineering and Computer Science, and the Department of Linguistics at the University of Michigan, where he is the leader of the Computational Linguistics and Information Retrieval (CLAIR) research group. He has had more than 100 publications in conferences and journals such as *Communications of the Association for Computing Machinery* (ACM), *Journal of Artificial Intelligence Research*, *Bioinformatics, Computational Linguistics, Information Processing and Management*, and *American Journal of Political Science*, among others. He is on the editorial boards of *Information Retrieval, Journal of Natural Language Engineering*, and *Journal of Artificial Intelligence Research*. Professor Radev is an ACM distinguished scientist as well as coach of the U.S. high school team in computational linguistics. He is also an Adjunct Professor in computer science at Columbia University.
GRAPH-BASED NATURAL LANGUAGE PROCESSING AND INFORMATION RETRIEVAL

RADA MIHALCEA
University of North Texas,
Department of Computer Science and Engineering

DRAGOMIR RADEV
University of Michigan,
School of Information
Department of Electrical Engineering and Computer Science
Department of Linguistics
Contents

Introduction
  0.1 Background 3
  0.2 Book Organization 4
  0.3 Acknowledgments 7

Part I. Introduction to Graph Theory
  1 Notations, Properties, and Representations 11
    1.1 Graph Terminology and Notations 11
    1.2 Graph Properties 13
    1.3 Graph Types 14
    1.4 Representing Graphs as Matrices 15
    1.5 Using Matrices to Compute Graph Properties 16
    1.6 Representing Graphs as Linked Lists 17
    1.7 Eigenvalues and Eigenvectors 18

  2 Graph-Based Algorithms 20
    2.1 Depth-First Graph Traversal 20
    2.2 Breadth-First Graph Traversal 22
    2.3 Minimum Spanning Trees 23
    2.4 Shortest-Path Algorithms 26
    2.5 Cuts and Flows 29
    2.6 Graph Matching 31
    2.7 Dimensionality Reduction 32
    2.8 Stochastic Processes on Graphs 34
    2.9 Harmonic Functions 38
    2.10 Random Walks 40
    2.11 Spreading Activation 41
    2.12 Electrical Interpretation of Random Walks 42
    2.13 Power Method 44
2.14 Linear Algebra Methods for Computing Harmonic Functions 45
2.15 Method of Relaxations 46
2.16 Monte Carlo Methods 47

Part II. Networks
3 Random Networks 53
  3.1 Networks and Graphs 53
  3.2 Random Graphs 54
  3.3 Degree Distributions 54
  3.4 Power Laws 57
  3.5 Zipf’s Law 58
  3.6 Preferential Attachment 61
  3.7 Giant Component 62
  3.8 Clustering Coefficient 62
  3.9 Small Worlds 63
  3.10 Assortativity 65
  3.11 Centrality 67
  3.12 Degree Centrality 67
  3.13 Closeness Centrality 68
  3.14 Betweenness Centrality 69
  3.15 Network Example 70
  3.16 Dynamic Processes: Percolation 72
  3.17 Strong and Weak Ties 74
  3.18 Assortative Mixing 76
  3.19 Structural Holes 76

4 Language Networks 78
  4.1 Co-Occurrence Networks 78
  4.2 Syntactic Dependency Networks 80
  4.3 Semantic Networks 81
  4.4 Similarity Networks 85

Part III. Graph-Based Information Retrieval
5 Link Analysis for the World Wide Web 91
  5.1 The Web as a Graph 91
  5.2 PageRank 92
  5.3 Undirected Graphs 95
  5.4 Weighted Graphs 95
9.8 Cross-Language Information Retrieval 166
9.9 Information Extraction 169
9.10 Question Answering 171
9.11 Term Weighting 174

Bibliography 179
Index 191
Graph theory is a well-studied discipline as are the fields of natural language processing and information retrieval. Traditionally, these areas of study have been perceived as distinct, with different algorithms, different applications, and different potential end-users. However, as recent research work has shown, these disciplines in fact are intimately connected, with much variety in the way that natural language processing and information retrieval applications find efficient solutions within graph-theoretical frameworks.

In a cohesive text, **language units** – whether they are words, phrases, or entire sentences – are connected through various relationships, which contribute to the overall meaning and maintain the cohesive structure and discourse unity of the text. Since the early stages of artificial intelligence, associative or semantic networks have been proposed as representations that enable the storage of such language units and their interconnecting relationships, which allow for a variety of inference and reasoning processes that simulate functionalities of the human mind (Sowa 1983). The symbolic structures that emerge from these representations correspond naturally to graphs – in which text constituents are represented as vertices and their interconnecting relationships form the edges in the graph.

Many text-processing applications can be modeled by means of a graph. These data structures have the capability to encode naturally the meaning and structure of a cohesive text and to follow closely the associative or semantic memory representations. For instance, Figure 0.1 is an example of graph representations of language units and their interconnecting relationships: 0.1(a) (adapted from Collins and Loftus [1975]) shows a network of concepts related by **semantic relations** – simulating a fragment of human memory on which reasoning and inferences about various concepts represented in the network can be performed; 0.1(b) shows a network with a similar structure, this time automatically derived via **definitional links** in a dictionary; and 0.1(c) represents a graph of the cohesive structure of a text by encoding similarity relationships among textual units.
Introduction

APPLE, the fleshy pome fruit of a tree of the rose family

FLESHY, succulent or pulpy

FRUIT, the usually edible reproductive part of a seed plant

PLANT, a young tree, vine, shrub, or herb

TREE, a woody perennial plant

PEAR, the fleshy pome fruit that is borne by a tree of the rose family

S1: Apples are mentioned throughout most of recorded human history.

S2: The generic name Malus was derived from the word *malus*, or bad, referring to Eve picking an apple.

S3: However, some scholars think the fig, and not the apple, was the forbidden fruit.

S4: The fig is also the first tree mentioned in the story of Adam and Eve.

S5: There are some scholars who think the apricot is a more likely candidate because it was an abundant fruit.

Figure 0.1. Graph representations of textual units and their interconnecting relationships.

This book is a comprehensive description of the use of graph-based algorithms for natural language processing and information retrieval. It brings together areas as diverse as lexical semantics, text summarization, text mining, ontology construction, text classification, and information retrieval, which are connected by the common underlying theme of using graph-theoretical methods for text- and information-processing tasks.
The book complements existing textbooks in natural language processing and information retrieval such as Manning and Schütze (1999); Jurafsky and Martin (2009); and Manning, Raghavan, and Schütze (2008); recent related books also cover the topic of network theory, including Kleinberg and Easley (2010), Jackson (2009), and Newman (2010).

Readers of this book will gain a firm understanding of the major methods and applications in natural language processing and information retrieval that rely on graph-based representations and algorithms. Our objective is that readers will have sufficient understanding to make informed decisions about using graph-based algorithms in text-processing applications in the future and to recognize opportunities for advancing the state of the art in natural language processing and information retrieval problems by the application of graph-theoretical algorithms.

0.1. Background

The idea of networks (or graphs\(^1\)) as mental representations for language units and the relationships among them originates with the early work in psychology (Freud 1901) and psycholinguistics (Schvaneveldt 1989; Spitzer 1999). Theories of semantic or associative memory (Quillian 1968) and connectionism (Fodor and Pylyshyn 1988) initially emerged in cognitive psychology as models for human language representation and reasoning; since then, they have been applied to various computer-based applications.

The semantic memory developed by Quillian (1968) was proposed as a representational format that enables the storage of the meaning of words or other cognitive units and then makes the use of such meaning possible for other applications. The memory model proposed by Quillian consists of a mass of nodes connected by associative links, which can be either simple associations or more complex attributes of the cognitive units. The model was introduced as a representation of “the memory that a person calls upon in his everyday language behavior” (Quillian 1968), which allows for associations and configurations as varied and rich as ideas expressed by humans in natural language.

Using his semantic-memory model, Quillian demonstrated how a properly constructed semantic network that encodes word types and tokens, as well as relationships among them derived via dictionary definitions, can

---

\(^1\) Except when noted otherwise, throughout this book, the notions of graph and network are used interchangeably.
be used successfully to identify the meaning of words, to find nontrivial semantic paths between words and ideas, and even to generate natural language texts starting with a network of concepts and relationships. His early work in semantic-memory models and computer-feasible representations established the foundation for more general theories of text understanding.

Starting with the seminal work of Quillian (1968) on theories of semantic memory, associative or semantic networks have been used as the underlying model for several applications in language processing, including word-sense disambiguation, automated reasoning, text generation, information retrieval, and text summarization.

A major impediment faced by early researchers in this area was the complexity of the resulting structures, which was not always supported by the underlying computer hardware. Consequently, limitations on the size of applications attempted with these approaches resulted. For similar reasons, the evaluation of such algorithms was mostly performed on toy-size problems (e.g., Quillian [1968] evaluated his proposed word-sense disambiguation algorithm on nineteen ambiguous words), and the scalability of the models was rarely evaluated, if ever.

The situation has changed in recent years, however, and the area of graph-based representations and algorithms applied to natural language processing and information retrieval has experienced tremendous growth. Graph-ranking algorithms (e.g., HITS [Kleinberg 1999] and PageRank [Brin and Page 1998]) have been used successfully on a large scale for analysis of the link structure of the World Wide Web (Brin and Page 1998). A series of successful workshops that began in 2006, centered on the theme of graph-based natural language processing, reached its fifth edition, as follows: Mihalcea and Radev 2006, New York; Biemann et al. 2007, Rochester; Matveeva et al. 2008, Manchester; Chowdhury et al. 2009, Singapore; and Banea et al. 2010, Uppsala. A growing number of publications on the use of graphs for text processing (i.e., more than four thousand papers that address the concept of graph in the Association for Computational Linguistics Anthology2) is also indicative of the increased interest in this area.

0.2. Book Organization

The material covered in this book is divided into four parts: the introductory part explains relevant graph representations and algorithms; the second part

---

2 Available at http://acl.ldc.upenn.edu.
discusses networks, including random networks and language networks; the third part presents the application of graphs to information retrieval; and the fourth part addresses the use of graphs in natural language processing.\(^3\)

0.2.1. Part I: Introduction to Graph Theory

The first part of the book is an overview of graph-based representations and algorithms, focusing on those methods that have been successful in the fields of natural language processing and information retrieval.

Notations, Properties, and Representations

Chapter 1 introduces the most commonly used graph notations, which are used as a reference throughout the book. The chapter also discusses the most important properties of graphs: diameter; clustering coefficient; degree distribution and average shortest path; and graph types and classifications, including random graphs, regular graphs, small-world graphs, and scale-free graphs. Finally, Chapter 1 describes representations that can be used in practical implementations, including adjacency lists and matrices.

Graph-Based Algorithms

Chapter 2 discusses the most frequently used graph-theoretical algorithms, emphasizing those that are relevant to the text and information processing applications addressed in Parts III and IV. Included in the chapter are graph-traversal, including depth-first and breadth-first strategies, minimum path length, and minimum spanning trees; flow-on graphs, including algorithms for min-cut/max-flow; graph matching; random walks with harmonic functions and electrical networks; and linear algebra on graphs.

0.2.2. Part II: Networks

Part II addresses random networks and naturally occurring networks, focusing specifically on networks that occur in linguistic structures.

Random Networks

Chapter 3 discusses random networks and naturally occurring networks. Their properties as well as algorithms for computing them are described.

\(^3\) The book does not include Hidden Markov Models (HMM), Conditional Random Fields (CRF), graphical models, trees in general, or neural networks.
Chapter 4 discusses language networks from an empirical perspective. It covers co-occurrence networks, syntactic and semantic networks, similarity networks, and networks of summaries and machine translation. Language networks are used in a number of methods and applications in natural language processing and information retrieval, such as syntax, semantics, and summarization.

0.2.3. Part III: Graph-Based Information Retrieval

Part III addresses the application of graph-based algorithms to problems relevant to the field of information retrieval, focusing on search engines, document reranking, and text classification and clustering.

Link Analysis for the World Wide Web

Chapter 5 addresses link-analysis methods used by search engines (e.g., PageRank and HITS), and discusses topics relevant to their application, including method stability, the combination of link-based and content-based models, topic-sensitive ranking, and query-dependent link analysis.

Text Clustering

Chapter 6 describes text clustering using a variety of graph-based algorithms, including the Fiedler method, Kernighan–Lin method, text categorization with min-cuts, betweenness, and random walks.

0.2.4. Part IV: Graph-Based Natural Language Processing

Part IV covers the application of graph-based methods to natural language processing, addressing separately the areas of semantics, syntax, and language-processing applications.

Semantics

Chapter 7 addresses the application of graph-based algorithms to problems in the area of semantics. It covers synonym detection and automatic construction of semantic classes using measures of graph connectivity on graphs built from either raw text or user-contributed resources; measures of
0.3 Acknowledgments

We want to thank attendees of the tutorials covering material from this book that we presented at the conferences on Recent Advances in Natural Language Processing (RANLP 2005), Human Language Technologies—North American Chapter of the Association for Computational Linguistics (HLT–NAACL 2006), and Workshop on Spoken Language Technology (SLT 2006). The feedback received during these tutorials helped us give shape to the book in its current version. We also want to thank the students semantic distance on semantic networks, including simple path-length algorithms as well as more complex random-walk methods; textual entailment using graph-matching algorithms on syntactic or semantic graphs; word-sense disambiguation and name disambiguation, including random-walk algorithms and other structural approaches for knowledge-based word-sense disambiguation as well as semi-supervised methods using label propagation on graphs; and sentiment classification using either semi-supervised graph-based learning or prior subjectivity detection with min-cut/max-flow algorithms.

Syntax

Chapter 8 covers the use of graph-theoretical algorithms for tasks in syntax, including part-of-speech tagging using graphs that encode word and tag dependencies; dependency parsing using minimum spanning trees; prepositional attachment using word-dependency distributions induced from random-walk models; and co-reference resolution using graph clustering and min-cut algorithms.

Applications

Chapter 9 is concerned with graph-theoretical methods for text-processing applications. The chapter includes text summarization using graph-centrality methods; keyword extraction and topic identification using random-walk language models; text segmentation using normalized-cut criteria for graph partitioning; graph structure to encode discourse relationships; word graphs for decoding in machine translation and speech processing; random-walk algorithms for translation selection in cross-language information retrieval; and graph representations and patterns on graphs for information extraction and question answering.

0.3. Acknowledgments

We want to thank attendees of the tutorials covering material from this book that we presented at the conferences on Recent Advances in Natural Language Processing (RANLP 2005), Human Language Technologies—North American Chapter of the Association for Computational Linguistics (HLT–NAACL 2006), and Workshop on Spoken Language Technology (SLT 2006). The feedback received during these tutorials helped us give shape to the book in its current version. We also want to thank the students
of the course on Advanced NLP and IR that we taught in Winter 2006 at the University of Michigan, as well as the students in the advanced course on Graph-Based Natural Language Processing taught in Spring 2008 at the University of North Texas. Some of the material in this book was included in the courses on Network Theory at Columbia University, Graph-Based Natural Language Processing at the University of North Texas, Search Engine Technology at Columbia University, and Information Retrieval at the University of Michigan.

We also thank Ahmed Hassan, Al Aho, Amjad abu Jbara, Chen Huang, Chris Manning, Güneş Erkan, Lada Adamic, Mark Newman, Pradeep Muthukrishnan, Prem Ganeshkumar, Rich Caneba, Ryan McDonald, Sara Stolbach, Sasha Blair-Goldensohn, and Vahed Qazvinian for their comments and assistance with this book.

Finally, we express our appreciation to the institutions that partly funded the work of the authors: National Science Foundation grants IIS 0534323, “Collaborative Research: BlogoCenter – Infrastructure for Collecting, Mining and Accessing Blogs”; IIS 0329043, “Probabilistic and Link-Based Methods for Exploiting Very Large Textual Repositories”; and BCS 0527513, “DHB: The Dynamics of Political Representation and Political Rhetoric”; and several National Institutes for Health grants to Dragomir Radev, R01 LM008106, “Representing and Acquiring Knowledge of Genome Regulation” and U54 DA021519 “National Center for Integrative Bioinformatics.” Research Grant 003594, “Graph-Based Natural Language Processing” from the Texas Advanced Research Program; a Google grant on “Finding Important Information in Unstructured Text”; and National Science Foundation Grants IIS 0747340, “CAREER: Semantic Interpretation with Monolingual and Crosslingual Evidence,” IIS 0917170 “Multilingual Subjectivity and Word Senses,” and IIS 1018613, “Building a Large Multilingual Network for Text Processing Applications” were awarded to Rada Mihalcea. Any opinions, findings, and conclusions or recommendations expressed in this book are those of the authors and do not necessarily reflect the views of the National Science Foundation or the other sponsors.
Part I
Introduction to Graph Theory
1
Notations, Properties, and Representations

This chapter introduces the most commonly used graph notations that are referenced throughout the book. The chapter also covers the most important properties of graphs: diameter, clustering coefficient, degree distribution and average shortest path; and graph types and classifications, including random graphs, regular graphs, small-world graphs, and scale-free graphs. Finally, the chapter describes the representations that can be used in practical implementations, including adjacency lists and matrices.

1.1. Graph Terminology and Notations

A graph is a data structure consisting of a set of vertices connected by a set of edges that can be used to model relationships among the objects in a collection. Graphs are typically studied in the field of graph theory, which is an area of study in mathematical research.

Figure 1.1 shows the seven bridges in Königsberg that led to the invention of graph theory. The problem is to cross all bridges without crossing any bridge twice. Euler showed in 1741 that it was impossible to do so (Euler 1741).

Formally, a graph is defined as a set $G = (V, E)$, where $V$ is a collection of vertices $V = \{V_i, i = 1, n\}$ and $E$ is a collection of edges over $V$, $E_{ij} = \{(V_i, V_j), V_i \in V, V_j \in V\}$. In alternative terminology, graphs also are referred to as networks, vertices as nodes, and edges as links.

Graphs can be either directed or undirected, depending on whether a direction of travel is defined over the edges. In a directed graph (or digraph), an edge $E_{ij}$ can be traversed from $V_i$ to $V_j$ but not in the other direction; $V_i$ is called the tail of the edge and $V_j$ is called the head. In an undirected graph, edges can be traversed in both directions. Figure 1.2(a) is an example of an undirected-graph structure consisting of five nodes connected by eight edges. Figure 1.2(b) is a similar graph but with directed edges. In the figures,
$A, B, C, D,$ and $E$ are vertices, and $AB$, $AC$, $AD$, $BE$, $CD$, $CE$, and $DE$ are edges that connect them.

If two vertices, $V_i$ and $V_j$, are connected by an edge, they are said to be adjacent. The edge $E_{ij}$ connecting them is said to be incident on the two vertices $V_i$ and $V_j$. In a directed graph, because an edge implies a direction of traversal, the tail of an edge is said to be adjacent to the head of the edge; conversely, the head of the edge is said to be adjacent from the tail of the edge. A graph in which every two vertices are adjacent to one another is called a complete graph.

One of the most important properties of a vertex in a graph is its degree, defined as the total number of edges incident to that vertex. In a directed graph, the degree of a vertex $V_i$ is split into (1) the in-degree of the vertex, defined as the number of edges for which $V_i$ is their head, and (2) the out-degree of the vertex, which is the number of edges for which $V_i$ is their tail. For instance, in Figure 1.2(b), the node $A$ has a degree of 3, an in-degree of 1, and an out-degree of 2.

---

**Figure 1.2.** A sample graph: (a) undirected, (b) directed, and (c) weighted undirected.
The average degree of a graph is as follows:

\[ a = \frac{1}{N} \sum_{i=1..N} \text{Degree}(V_i) \]

where \( N \) is the number of vertices in the graph.

A sequence of vertices and edges in which every edge is incident on the vertices that precede and follow it in the sequence is called a path. A path with no repeating vertices is called a simple path. A cycle is a path in which the first and the last vertex coincide. For instance, the nodes \( A, B, \) and \( C \) in Figure 1.2 form a cycle. A graph in which any two vertices are connected by a path is called a connected graph.

A subgraph of a graph \( G = (V, E) \) is defined as a graph \( S = (V_S, E_S) \), in which \( V_S \) and \( E_S \) are subsets of \( V \) and \( E \), respectively, and \( E_S \) contains edges connecting vertices in \( V_S \). Conversely, in this definition, \( G \) is said to be the supergraph of \( S \). A connected component is a maximal connected subgraph, meaning that it is a subgraph that has the property of being connected and, at the same time, no vertex can be added to the subgraph without violating that property. A clique is a subgraph that is also complete; a \( k \)-clique is a clique with \( k \) vertices.

By default, edges in a graph are unweighted; thus, the corresponding graphs are said to be unweighted. When a weight is assigned to the edges in the graph, as in Figure 1.2(c), the graphs are said to be weighted.
neighborhood (i.e., between the vertex and its neighbors and between the
neighbors themselves) divided by the total number of possible edges in the
vertex neighborhood. Thus, if a vertex is part of a clique, then its clustering
coefficient is 1. Conversely, if the vertex is not connected to any other vertex
in the graph, its clustering coefficient is 0. The clustering coefficient of a
graph is defined as the average of the clustering coefficients of all nodes in
the graph.

The degree distribution of a graph is the probability distribution cal-
culated over the degrees of the vertices in the entire graph. For instance,
considering the vertices with degree \( k \), then \( P(k) \) is the fraction of nodes in
the graph that have degree \( k \) (i.e., the number of such vertices divided by the
total number of vertices in the graph). The degree distribution of a graph thus
can be plotted as the function \( P(k) \) with respect to the degree value \( k \). The
dergree distribution is an important property in the study of a real network.
For example, many social networks have a degree distribution following a
power law, mathematically written as \( P(k) = k^{-c} \), where \( c \) is a constant.

Chapter 3 discusses the clustering coefficient and degree distributions of
naturally occurring networks in more detail.

1.3. Graph Types

Depending on the process used to generate the vertices and edges in the
graph, as well as its structural properties, graphs can be roughly divided
into several types.

\textit{Bipartite graphs} are those in which vertices can be divided into two
disjoint subsets and all of the edges in the graph connect vertices in different
subsets.

\textit{Random graphs} are those in which edges are randomly generated. Start-
ing with a set of \( N \) vertices, a random graph is obtained by randomly adding
edges between the vertices. If every edge is added with a given probability
\( p \), the result is a specific type of random graphs – namely, the Erdős-Rényi
graph model.

\textit{Regular graphs} are those in which all the vertices have the same degree.
There can be 0-regular graphs, in which no vertex is connected to another,
and therefore the graphs consist of disconnected vertices; 1-regular graphs,
in which each vertex is connected to exactly one other vertex, and therefore
the graphs consist of disconnected edges; and 2-regular graphs, in which
each vertex is connected to exactly two other edges, and therefore the graphs
consist of disconnected cycles. Any complete graph is a regular graph.
1.4 Representing Graphs as Matrices

Scale-free graphs are those with a degree distribution following a power law. That is, given a constant $c$, the probability that a vertex has degree $k$ can be determined as $P(k) = k^{-c}$. Many real networks – such as computer networks including the World Wide Web, social networks including blogs or email communication networks, and semantic networks such as WordNet (Miller 1995) – are scale-free graphs.

Small-world graphs are those in which the average path between every two vertices is small. That is, in a small-world graph, a vertex can be reached from any other vertex through a small number of edges. Small-world networks are characterized by a small average shortest path and a large clustering coefficient. Small-world graphs also typically have a large number of cliques. Many (although not all) small-world graphs have a degree distribution following a power law; thus, they also are scale-free graphs. Social networks, discussed in more detail in Chapter 3, are usually small-world networks.

1.4. Representing Graphs as Matrices

A vector $V$ is a $1 \times N$ array of objects (such as numbers), whereas a matrix $M$ is an $M \times N$ array. Vectors can be used to represent the coordinates of a point. For example, in a room, the vector $(0, 0, 0)$ may represent the lower left-hand corner at the front of the room and $(1, 1, 1)$ may refer to the upper right-hand corner at the back of the room. Any point within the confines of the room has a unique vector representation.

Matrices often are used to represent ordered collections of vectors. For example, a set of three vectors, all having the same length of five elements, is represented collectively as a $3 \times 5$ matrix.

Graphs and matrices often are used interchangeably to represent relational data. Figure 1.3 shows the matrix representation for the graph $G$ shown in Figure 1.2(b). It has five nodes ($A$ through $E$) and seven edges.

$$G = \begin{pmatrix}
0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

Figure 1.3. Matrix representation for the graph in Figure 1.2(b).
\( G = \begin{pmatrix}
0 & 8 & 5.2 & 1 & 0 \\
8 & 0 & 0 & 0 & 12 \\
5.2 & 0 & 0 & 7 & 3 \\
1 & 0 & 7 & 0 & 0.5 \\
0 & 12 & 3 & 0.5 & 0
\end{pmatrix} \)

Figure 1.4. Matrix representation for the weighted graph in Figure 1.2(c).

\((AB, AC, BE, CD, CE, DA, \text{ and } DE)\). \(G\) is a directed graph because not all relationships are symmetric. \(G\) can be represented as a matrix, in which the rows and columns correspond, in order, to the vertices \(A\) through \(E\), and a value of 1 indicates the presence of a directed edge between the corresponding nodes.

Undirected graphs are represented in a similar way, but an edge is represented redundantly by using two cells. For instance, the edge \(AC\) is represented using a value of 1 stored in the cells corresponding to the (directed) edges \(AC\) and \(CA\). Alternatively, undirected graphs also can be represented using diagonal matrices to avoid redundancy.

Finally, weighted graphs can be represented using a similar matrix structure, with the values of the cells corresponding to the weight of the edges. For example, the matrix in Figure 1.4 is the representation of the weighted undirected-graph structure in Figure 1.2(c).

1.5. Using Matrices to Compute Graph Properties

The matrix representation of a graph can enable faster computation of graph properties. However, this representation is not always applicable because it tends to grow very quickly in size. For instance, whereas a graph of ten elements can be represented using a matrix of \(10 \times 10 = 100\) elements, a graph of a thousand vertices requires a matrix of a million cells, and so on.

The following formulas show how to quickly compute several graph properties when the matrix representation can be used:

Degree (for symmetric matrices): The degree of a vertex is the number of its neighboring nodes.

\[
\text{Degree}(V_i) = \sum_j G_{i,j} = \sum_i G_{j,i}
\]
In-degree: the number of incoming edges
\[ \text{Degree}(V_{i,in}) = \sum_j G_{i,j} \]

Out-degree: the number of outgoing edges
\[ \text{Degree}(V_{i,\text{out}}) = \sum_j G_{j,i} \]

Paths of length \( k \): To find all paths of length 1, we can count the number of ones in the matrix. For all paths of length 2, we similarly look at the square of the matrix. In general, the matrix \( G^k \) contains information about the number of paths of length \( k \), and it can be computed easily by repetitive multiplication. To find all paths up to length \( k \), the union of all powers \( i \) of \( G \), where \( i < k \), is needed:

Cycles of length \( k \)
\[ \frac{1}{k} \text{Tr} G^k \]

where \( \text{Tr} \) is the trace of the matrix \( G \).

Co-citation: Two vertices are co-cited if the same other vertex points to them:
\[ C_{V_i,V_j} = \sum_k G_{i,k} G_{j,k} \]

Bibliographic coupling: Two vertices are coupled if they point to the same other vertex:
\[ B_{V_i,V_j} = \sum_k G_{k,i} G_{k,j} \]

1.6. Representing Graphs as Linked Lists

Matrix representations of graphs are efficient for small graph structures. When the graphs grow large, however, these representations are no longer suitable because of the typical sparseness of the corresponding matrices. Consider, for example, the matrix representation of the graph in Figure 1.2(b). In this matrix (shown in Figure 1.3), only seven of twenty-five matrix cells are used; the other cells, although they still occupy memory space, correspond to nonexisting edges.

The solution in such cases is the use of linked lists (also referred to as adjacency lists), which store only the actual edges in the graph, thereby avoiding the space ineffectiveness associated with sparse matrices. In these
representations, each vertex is associated with a linked list, which stores its neighbor vertices. For weighted graphs, the linked-list elements also can store the edge weights. Considering the graph in Figure 1.2(b), the corresponding linked-list representation is shown in Figure 1.5.

1.7. Eigenvalues and Eigenvectors

Eigenvalues and eigenvectors appear frequently in matrix analysis. Figure 1.6 shows an example.

A scalar $\lambda$ is called an eigenvalue of the square matrix $G$ if there exists a nonzero vector $x$ such that:

$$Gx = \lambda(x)$$
The vector $x$ is then called the \textit{eigenvector} of $G$ associated with $\lambda$. Whereas the eigenvector is not allowed to be zero, it is acceptable for $\lambda$ to be zero (or any other complex number).

To compute all eigenvalues of a matrix, its determinant needs to be used:

$$\det(G - \lambda I) = 0$$

where $I$ is the identity matrix of the same size as $G$.

Consider an example with the $2 \times 2$ matrix $G$:

$$G = \begin{pmatrix} -1 - \lambda & 3 \\ 2 & -\lambda \end{pmatrix} = 0$$

$$G = \begin{pmatrix} -1 & 3 \\ 2 & 0 \end{pmatrix} = 0$$

The determinant $\det(G - \lambda I) = (-1 - \lambda) * (-\lambda) - 3 * 2 = 0$.

Then, $\lambda + \lambda^2 - 6 = 0; \lambda_1 = 2; \lambda_2 = -3$.

For $\lambda_1 = 2$:

$$\begin{pmatrix} -3 & 3 \\ 2 & -2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = 0$$

Therefore, the solutions for $\lambda_1$ are $x_1 = x_2$. The solutions for $\lambda_2$ can be computed similarly.

Further Reading

The following books present additional relevant graph theory: for percolation theory, see Grimmett 1999; for random graphs, see Bollobás 1985; and for modern graph theory, see Bollobás 1998.
Graph-Based Algorithms

This chapter discusses frequently used graph-theoretical algorithms, with emphasis on those algorithms that are relevant to the text-processing applications addressed in Parts III and IV. The chapter covers graph-traversal, including depth-first and breadth-first strategies, minimum path length, and minimum spanning trees; flow-on graphs, including algorithms for min-cut/max-flow; graph matching; random walks, with harmonic functions and electrical networks; and linear algebra on graphs.

2.1. Depth-First Graph Traversal

Algorithms for graph traversal are concerned with reaching all of the nodes in a graph while obeying certain constraints. The depth-first and breadth-first traversal strategies are concerned primarily with the order in which the nodes in the graph are traversed. The minimum-spanning-tree algorithm reaches to all nodes in the graph and ensures at the same time that the traversal path forms a tree.

Depth-first traversal is considered in this section. The next section addresses breadth-first traversal.

In depth-first traversal, the traversal of the graph starts from one node and then iteratively progresses by moving from each node to one of its neighbors until it reaches a node that has no untraversed neighbors. When such a “dead end” node is hit, the algorithm backtracks and continues the traversal with other candidate nodes until all nodes in the graph are covered.

Consider the graph provided as an example in Figure 1.2 and replicated in Figure 2.1. Assuming that we start with the node $A$ in the undirected graph shown in Figure 2.1(a), the neighbors of $A$ are $B$, $C$, and $D$. We choose one of its neighbors – say, $B$ – and add it to the traversal list. Next, we look up the neighbors of $B$ that have not been traversed already and choose $E$. Note that even if $A$ is one of $B$’s neighbors, it is not a valid
2.1 Depth-First Graph Traversal

Figure 2.1. Depth-first search on a sample graph: (a) undirected; and (b) directed.

candidate because it already has been included in the traversal list in a previous step. Next, from $E$, we could move to $C$ or $D$ ($B$ is already on the traversal list), and let us say that we choose $D$. Finally, $D$ has $A$, $C$, and $E$ as neighbors; however, $A$ and $E$ have been included already, so we move to $C$. At this point, the algorithm stops because all nodes in the graph have been included in the traversal list.\(^1\) Figure 2.1 shows the order of traversal of the nodes as numbers in parentheses next to each one.

The traversal algorithm is similar in directed graphs, but this time we also must obey the direction of the arcs when deciding on which nodes to follow in the traversal. For instance, starting again with node $A$ in the directed graph shown in Figure 2.1(b), the neighbors to which we can move are $B$ and $C$. Let us say that we choose $B$ to add to the traversal list. Next, the only candidate node to which to move is $E$. There are no nodes to which we can move from $E$ (i.e., $E$ is a head for all of the edges incident on $E$); thus, we must backtrack to $B$. $B$ is again a dead end because the only node that we could move to $E$ has been included already in the traversal list; therefore, we must backtrack again to $A$. The next node to which we can move from $A$ is $C$ and, finally, from $C$ we can move to $D$, at which point we have covered all of the nodes in the graph.

The depth-first traversal algorithm can be implemented easily with the help of a recursive function, as shown in the pseudocode in Figure 2.2.

In the worst case, every vertex and edge will be explored; therefore, the running time of the depth-first traversal algorithm is $O(|E| + |V|)$, where $|E|$ represents the number of edges and $|V|$ is the number of vertices in the graph.

---
\(^1\) If all of the nodes were not included at this point, the next step would be to backtrack: Move from $C$ back to the previous node in the traversal $D$ and attempt another path forward. When all paths starting with $D$ are completed, move from $D$ to the previous node $E$, and so forth.
L: a global variable containing the list of explored nodes

FUNCTION DFT(G, N)
INPUT
  G : graph with NODES nodes and EDGES edges
  N : a node in the graph
OUTPUT
  a labeling of edges as ‘‘discovery’’ or ‘‘back’’ edges
RECURSIVE STAGE
foreach E (E in edges and E incident on N) {
  M = endpoint of E, adjacent from N
  if M not in L {
    add M to L
    mark E as ‘‘discovery’’ edge
    call DFT (G, M);
  }
  else {
    mark E as ‘‘back’’ edge
  }
}
END FUNCTION DFT
return L

Figure 2.2. Depth-first traversal of a graph.

2.2. Breadth-First Graph Traversal

In the breadth-first traversal algorithm, the traversal of the graph starts from one node and then moves to all neighboring nodes – that is, to all nodes that are connected to the initial node by exactly one edge. The next set of nodes to be traversed consists of the neighboring nodes of the now-expanded “frontier” of the traversal, ensuring that no node is traversed twice. The algorithm continues iteratively until all nodes in the graph are covered.

To illustrate, consider the graph first shown in Figure 1.2 and replicated in Figure 2.3. Assuming that we start with node A in the undirected graph in Figure 2.3(a), the neighbors of A are B, C, and D, which are traversed in order. Next, we look at the neighbors of this new set of nodes that has been
2.3 Minimum Spanning Trees

A spanning tree for a connected, undirected graph \( G \) is a subgraph of \( G \) that contains all of the vertices in \( G \) and also is an undirected tree. In a weighted...
L: a global variable containing the list of explored nodes
Q: a global variable consisting of a queue of nodes
   (the "frontier" of the traversal)

FUNCTION BFT(G, N)

INPUT
   G : graph with NODES nodes and EDGES edges
   N : a node in the graph

OUTPUT
   a labeling of edges as "discovery" or "cross" edges

INITIAL STAGE
   Q = {N}

ITERATIVE STAGE
while Q not empty {
   M = dequeue Q
   foreach E (E in edges and E incident on M) {
      S = endpoint of E, adjacent from M
      if S not in L {
         add S to L
         mark E as "discovery" edge
         enqueue S on Q
      }
      else {
         mark E as "cross" edge
      }
   }
}

END FUNCTION BST

return L

Figure 2.4. Breadth-first traversal of a graph.

A graph, the weight of a subgraph is defined as the sum of the weights of the edges in the subgraph. We therefore define a minimum spanning tree as a spanning tree with the minimum weight. For instance, given the weighted graph in Figure 2.5(a), the subgraphs shown in (b), (c), and (d) represent minimum spanning trees of the original graph.
2.3 Minimum Spanning Trees

Minimum spanning trees have the following properties: (1) As shown in Figure 2.5, it is not unusual for a graph to have several minimum spanning trees. However, even if a graph admits several minimum spanning trees, all of the trees will have the same total weight. (2) For every edge not included in the minimum spanning tree, adding the edge to the tree results in the construction of a cycle. Moreover, the newly added edge will be a maximum-weight edge in that cycle. (3) The number of edges in a minimum spanning tree is equal to the number of nodes in the graph minus 1.

Several algorithms can be applied to identify the minimum spanning trees of a graph, including Prim’s algorithm. Briefly, the algorithm starts with an arbitrary node (referred to as the “root”). Next, at each iteration, it branches out from the tree constructed so far by choosing an edge that has the minimum weight among all the edges that could be attached. The algorithm adds to the tree the vertex that is associated with that edge. Thus, vertices in the graph can be divided into three disjoint categories: (1) tree vertices, which are the vertices belonging to the tree constructed so far; (2) fringe vertices, which are the vertices that are not part of the tree but are adjacent to some vertex in the tree; and (3) unseen vertices, which are all of the other vertices in the graph. Prim’s algorithm is described in the pseudocode in Figure 2.6.

Consider the example in Figure 2.5(a) and assume that we select the node $A$ as the root node to start the algorithm. We label all nodes adjacent to $A$ as fringe; thus, the fringe set contains the nodes $B$, $C$, and $D$. Next, we choose the edge between any tree node and fringe node that has minimum weight. Both $AB$ and $AD$ have a minimum weight of 2.0, so assume that we randomly pick $AB$. This edge is added to the tree and $B$ is relabeled as a tree node. Next, we add all nodes adjacent to $B$ to the list of fringe nodes: Node $D$ already is labeled as a fringe node from the previous iteration; therefore, no changes occur in the list of fringe nodes. We again seek an edge of minimum weight between the tree nodes ($A$, $B$) and the fringe nodes ($C$, $D$). Thus, we select the edge $BD$ and add it to the tree. Finally,
FUNCTION PRIM (G)

INPUT
G : graph with NODES nodes and EDGES edges

OUTPUT
a minimum spanning tree in G

INITIAL STAGE
set all nodes as UNSEEN
select arbitrary node N in G
label N as TREE
relabel all vertices adjacent to N as FRINGE

ITERATIVE STAGE
while (there exist nodes labeled as FRINGE) {
    select minimum weight edge between
    any TREE node T and any FRINGE node F
    relabel F as TREE
    add edge TF to the tree
    relabel all vertices adjacent to F as FRINGE
}

return tree

END FUNCTION PRIM

Figure 2.6. Prim’s algorithm for finding the minimum spanning tree.

we must choose a minimum-weight edge between the tree nodes (A, B, D) and the fringe node (C). Although AD has a smaller weight than AC, the addition of the edge AD to the tree adds a cycle, thereby transforming the tree into a graph. Therefore, we add the edge AC to the tree. The final minimum spanning tree obtained after all iterations consists of the edges AB, AC, and BD.

2.4. Shortest-Path Algorithms

Given a graph G and a source node N in the graph, the shortest-path problem is the problem of finding the shortest path from N to any other vertex in the graph. In the case of unweighted graphs, the length of a path is calculated as the number of edges on the path. In the case of weighted
2.4 Shortest-Path Algorithms

A graph, the length is calculated as the sum of the weights of all edges on the path. The algorithm used for the case of weighted graphs also is referred to as *Dijkstra’s algorithm*.

Consider, for example, the graph shown in Figure 2.7(a) and assume that we start with the source node *C*. The length of the shortest path from *C* to *C* is obviously 0; thus, we mark the length of the path next to the node. Next, we look for vertices that are within a path of length 1 from *C*, and we find *A* and *F*. The next step is to find all vertices that are adjacent to these newly found nodes (*A*, *F*), which are vertices with a path of length 2 from *C*; we find *D* and *B*. Finally, we look for nodes adjacent to *D* and *B*, and we find *E* and *G*, which have a shortest path of length 3 from the source node *C*. This algorithm is similar to the breadth-first traversal introduced in Section 2.2.

The pseudocode for finding the shortest paths from a source node to all other nodes in an unweighted graph is shown in Figure 2.8.

When the shortest path must be calculated between each pair of vertices in the graph, other algorithms such as the Floyd–Warshall algorithm can be used, which minimizes the number of path comparisons by using dynamic programming.

The shortest-path algorithm also can be applied to weighted graphs, with small modifications. Primarily, the path is calculated as the sum of weights of the edges; the length of the shortest path to a node can be updated if a new shorter path is found.

Assume that we have a weighted graph as shown in Figure 2.9(a), and the source node is the vertex *C*. We set the path from *C* to *C* as 0 and mark *C* as known (indicated by an asterisk in the figure). From *C*, we can reach nodes *A* and *F* for a shortest path of lengths 3 and 4, respectively. Next, from among all the unseen vertices, we choose the vertex with the smallest
FUNCTION SHORTEST_PATH (G, S)

INPUT
G : graph with NODES nodes and EDGES edges
S : a source node in G

OUTPUT
the shortest path from S to all the nodes in G

INITIAL STAGE
foreach N in nodes {
    shortest_path(N) = INFINITY
}
shortest_path(S) = 0
enqueue S on Q

select arbitrary node N in G
label N as TREE
relabel all vertices adjacent to N as FRINGE

ITERATIVE STAGE
while (Q is not empty) {
    N = dequeue Q;
    foreach M adjacent to N {
        if (shortest_path(M) = INFINITY) {
            shortest_path(M) = shortest_path(N) + 1
            enqueue M on Q
        }
    }
}

END FUNCTION SHORTEST_PATH

Figure 2.8. Pseudo-code algorithm for finding the shortest path in unweighted graphs.

shortest path; hence, we choose vertex A and mark it as known. From A, we can reach B and D, shortest paths of lengths 5 and 4, respectively. Next, from the vertices with the smallest shortest path, D and F (both with a shortest path of length 4), we choose one at random. We choose D and mark it as known. From D, we can reach E and G and update their shortest paths to lengths 6 and 7, respectively. We also can reach C, but the shortest
path through $D$ is 6, which is higher than the current shortest path of 0; thus, no update is performed. Next, we choose $F$ and mark it as known; then we pick $E$ and mark it as known; and, finally, $G$. Figure 2.9(b) shows the shortest paths marked next to each node.

2.5. Cuts and Flows

A cut is a partition of a graph into two disjoint subsets. The size of a cut is the number of edges that must be removed to convert $G$ into this partition. A bridge is an edge that creates a cut when it is removed.

Given a weighted graph $G$, a source node $S$, and a sink node $T$, the minimum cut (or min-cut) problem consists of finding a cut that partitions the graph in two, such that $S$ and $T$ do not belong to the same partition and the cut has minimum weight (i.e., the sum of weights of the edges on the cut) over all possible cuts.

In a weighted graph, each edge has a capacity (i.e., the weight) and each edge can receive some flow. For instance, in a pipe network, the flow could be the fluid that it transported over the pipes. In a computer network, the flow could be the data that are exchanged among computers in the network. Figure 2.10 is an example of a cut and flow in a graph consisting of eight vertices.

A flow in a graph must respect the constraint that the flow on any given edge does not exceed the capacity of the edge. Moreover, all the vertices have the property of preserving the flow, which means that the incoming flow is equal to the outgoing flow. Thus, the maximum flow (or, max-flow) problem consists of finding the largest-flow that can be sent from the source node to the sink node such that the flow does not exceed the capacity of the edges across which it is sent.
The minimum cut and maximum flow are related by the Ford–Fulkerson theorem, which states that in any graph, the value of the max-flow is equal to the value of the min-cut.

Similar to the min-cut algorithm, the normalized-cut algorithm seeks to minimize the cut between two (or more) graph partitions; however, it also seeks to maximize the intrapartition node similarity. This is achieved by normalizing the cost of a cut with the volume of the partition, as follows:

\[
\text{NormalizedCut}(A, B) = \frac{\text{Cut}(A, B)}{\text{Volume}(A)} + \frac{\text{Cut}(A, B)}{\text{Volume}(B)}
\]

(2.1)

where

\[
\text{Cut}(A, B) = \sum_{u \in A, v \in B} w_{uv}
\]

(2.2)

and

\[
\text{Volume}(A) = \sum_{u \in A, v \in V} w_{uv}
\]

(2.3)

For the case of \(N\) partitions, the normalized cut becomes:

\[
\text{NormalizedCut}(A_1, A_2, \ldots, A_N)
\]

\[
= \frac{\text{Cut}(A_1, V - A_1)}{\text{Volume}(A_1)} + \cdots + \frac{\text{Cut}(A_N, V - A_N)}{\text{Volume}(A_N)}
\]

(2.4)

where \(A_1, A_2, \ldots, A_N\) represent a candidate partitioning of the graph, and \(V - A_i\) denotes the difference between the entire set of vertices in the graph and the partition \(A_i\).

When \(N\) is larger than 2, the multiway partitioning is NP-complete. However, it can be solved using dynamic programming by adding a constraint that requires all nodes in a graph partition to form a consecutive

\[2\] NP-complete problems are intractable in most practical cases.
sequence. With this constraint, the running time for finding the normalized cut across several partitions becomes polynomial, which allows for efficient processing even in the case of large graphs.

2.6. Graph Matching

Two graphs $G = (V, E)$ and $H = (W, F)$ are isomorphic if there is a bijective function $f : V \rightarrow W$ such that for all of the nodes $A, B$ in $V$, we have \{A, B\} $\in E \iff \{f(A), f(B)\} \in F$. In other words, there is a one-to-one correspondence that can be found between the vertices and the edges in the two graphs $G$ and $H$. The graphs shown in Figure 2.11(a) and (b) are isomorphic to one another, with vertices carrying the same label being mapped to one another.

A graph $G = (V, E)$ is isomorphic to a subgraph of $H = (W, F)$ if there is a subgraph $H_1$ of $H$ such that $G$ and $H_1$ are isomorphic. For instance, the graph shown in Figure 2.11(c) is isomorphic to a subgraph of the graph in Figure 2.11(a). Going from (a) to (c), there is one missing vertex ($A$) and several missing edges (i.e., all of the edges incident on $A$ as well as the edge $AD$).

Finding isomorphisms between two graphs is an NP-complete problem. However, we can make use of heuristics for an approximate solution. One such heuristic is the iterative partitioning of the vertices. We partition the vertices in $G$ and $H$ into classes such that each class in $G$ has a corresponding class in $H$; if $f$ is an isomorphism from $G$ to $H$, then nodes $A$ in $G$ and $f(A)$ in $H$ belong to the same class. To find the partitioning into classes, a simple scheme is to partition vertices based on their degree. More sophisticated iterative schemes also exist.

Another error-tolerant graph-matching algorithm is the graph edit distance, which measures the dissimilarity of two graphs $G$ and $H$ as the number of changes needed to transform one graph into the other. Thus, the edit distance is determined as the number of edge additions and edge deletions required to transform $G$ into $H$. 
2.7. Dimensionality Reduction

An important technique in data analysis with wide applications to information retrieval and natural language processing is *dimensionality reduction*. To explain this concept, the data in a matrix must be perceived as a series of observations, each of which is represented by a vector. Consider the following dataset that represents two properties of the documents in a collection. Each row represents a document and each column represents a feature. Figure 2.12 shows an example of these properties.

A useful application of dimensionality reduction to natural language processing and information retrieval is *Latent Semantic Analysis* (LSA; sometimes referred to as LSI, or Latent Semantic Indexing). The goal is to identify hidden semantic dimensions in textual data and then use the mapping from the original words to these semantic dimensions to obtain a better similarity measure between documents and/or queries. For example, two documents (or sentences) may be about “car” and “automobile” (or “Toyota”), respectively. If these words co-occur in other documents, then in the lower dimensional representation, these two documents appear to be similar, even though they are dissimilar in their original “bag-of-words” representation.
2.7 Dimensionality Reduction

\[
A = [
1 1 1 1 1 0 0 1 0 ;
1 1 1 1 0 0 0 0 1 ;
0 1 0 1 0 0 0 0 0 ;
1 0 1 1 0 0 0 1 1 ;
0 0 0 1 1 1 0 0 1 ;
0 0 0 0 1 1 1 1 1 ;
0 0 0 0 1 1 1 1 0 ;
] ;
\]

Figure 2.13. Singular value decomposition matrix.

The most important equation in LSA is the following singular value decomposition:

\[ A = U S V^T \]

where \( U \) is the matrix of orthogonal eigenvectors of \( AA^T \), \( V \) is the matrix of the orthogonal eigenvectors of \( A^T A \). The components of \( S \) are the eigenvalues of \( A^T A \). This decomposition exists for all matrices, dense or sparse. If \( A \) has five columns and three rows, then \( U \) is \( 5 \times 5 \) and \( V \) is \( 3 \times 3 \). In Matlab, the following command is used:

\[ U, S, V = \text{svd}(A). \]

As an example, consider a collection that includes nine documents and seven terms. The seven terms are as follows: \( T_1 = \text{“role”}, T_2 = \text{“play”}, T_3 = \text{“cast”}, T_4 = \text{“actor”}, T_5 = \text{“dollar”}, T_6 = \text{“ticket”}, \) and \( T_7 = \text{“sale”} \). The first four are about acting and the last three are about sales of a movie.

The matrix in Figure 2.13 shows the terms and documents, and the correspondence between them. Here, the first four documents are from cluster one; (D5, D6, and D7) are from the second; and (D8, D9) belong to both.

\( A \) is a \( 9 \times 7 \) matrix such that \( A_{i,j} = 1 \iff \) document \( i \) includes term \( j \). \( A \) then must be column-normalized, and the singular value decomposition can be calculated (e.g., using a built-in function in software such as Matlab). The result, \( S \), is a \( 9 \times 7 \) matrix whose diagonal includes the eigenvalues.

Next, all the rows and columns in \( S \) except the first two are assigned to zero. This clears all except the first two largest eigenvalues:

\[ S_2 = S \]
\[ S_2(3:7,3:7) = 0 \]
Finally, the new space with 2-D coordinates is created:

\[ \mathbf{dd} = \mathbf{a}' \ast \mathbf{U} \ast \mathbf{S}_2 \]
\[ \mathbf{tt} = (\mathbf{S}_2 \ast \mathbf{V}' \ast \mathbf{a}')' \]

These 2-d coordinates show how the documents and terms are clustered:

\[
\begin{array}{cccccccc}
-1.7322 & -0.8659 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1.3744 & -0.8076 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1.7322 & -0.8659 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1.7927 & -0.6509 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1.6607 & 0.8375 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1.2739 & 1.2163 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1.0833 & 1.2489 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1.8004 & 0.4687 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1.7075 & 0.0817 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
-2.4369 & -0.6904 & 0 & 0 & 0 & 0 & 0 & 0 \\
-2.0357 & -1.0732 & 0 & 0 & 0 & 0 & 0 & 0 \\
-0.7299 & -0.4736 & 0 & 0 & 0 & 0 & 0 & 0 \\
-2.0846 & -0.6351 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1.4739 & 0.5444 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1.8695 & 1.4252 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1.4789 & 1.3996 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

Figures 2.14 and 2.15 show some of the intermediate steps of the singular value decomposition process. Figure 2.16 is the two-dimensional representations of these documents, showing how they can be grouped into two clusters, with the boundary being the X axis.

2.8. Stochastic Processes on Graphs

A large class of algorithms on graphs falls into the category of stochastic random-walk methods. A random walk on a graph is a process during which an agent moves one step at a time from one node of the graph to an adjacent node. Each move depends only on the current node and on the graph itself, not on the previously visited nodes. For example, if the graph were an
undirected circle (also known as 1-regular one-dimensional lattice) with eight nodes labeled from 1 to 8, then a random walk could be the sequence of steps \((1, 2, 3, 2, 3, 4, 5, 6, 5, 4)\). The number of steps is an integer and can be either finite or infinite. Random walks can be defined on either weighted or unweighted graphs.

A **stochastic graph** is one in which for each node \(V\), the sum of the weights on all edges incident to that particular node are non-negative and sum to 1. In that case, the weights can be interpreted as probabilities.

On a symmetric lattice, assuming that all edges have the same weight, that weight can be set to 0.5 (i.e., the normalized value), thereby turning the graph into a stochastic graph. Figure 2.17 is an example of a matrix that corresponds to a stochastic graph.
Graph-Based Algorithms

In a stochastic matrix, each row (or column) represents a probability distribution over all states of the graph. Its interpretation is as follows: if a token is in state \( S_i \), its next state \( S_{i+1} \) is chosen according to the probability distribution on row \( i \).

For example:

\[
S = \begin{pmatrix}
\frac{3}{8} & \frac{5}{8} \\
\frac{3}{4} & \frac{1}{4}
\end{pmatrix}
\]

**eigenvalues:**

\[
1, \quad -0.375
\]

**eigenvectors:**

\[
\begin{pmatrix}
0.707 & -0.640 \\
0.707 & -0.768
\end{pmatrix}
\]

Figure 2.15. Singular value decomposition steps, part two.
The largest eigenvalue of a stochastic matrix $S$ is real and equal to 1: $\lambda_1 = 1$. For that value of $\lambda$, the left principal eigenvector is $V$ and the right principal eigenvector is $1$. In other words, $S^T V = V$.

The following example illustrates the use of stochastic matrices. Assuming that at time $t = 0$, a token starts in node 1, the probability distribution of its position over all states of the lattice after a certain number of steps can be determined using matrix multiplication. For instance, knowing that the initial probability distribution is $P_0^T = (1, 0, 0, 0, 0, 0, 0, 0)$, after one iteration, the probability distribution is $P_1 = P_0 \ast S$, which is determined as $(0, 0.5, 0, 0, 0, 0, 0, 0.5)$. In other words, the token has an equal chance of being in state 2 and state 10 (but not in state 1).
2.9. Harmonic Functions

Let us consider a simple open chain labeled from node 0 to node $N$. It is clear that $f(0) = 0$ and that $f(N) = 1$. It also can be observed that the value of $f(x)$ at intermediate nodes is equal to the average of its values to the left and right. This follows from the fact that there is a 50 percent probability of moving in either direction. Therefore, we have the following constraints:

$$
\begin{align*}
  f(0) &= 0 \\
  f(N) &= 1 \\
  f(x) &= 1/2 f(x - 1) + 1/2 f(x + 1)
\end{align*}
$$

Note that in the general case of a graph that is not necessarily a chain, we obtain the following instead:

$$
  f(x) = \frac{1}{n} \sum_{i \in \text{neighbors}(x)} f(i)
$$

2.9.1. The Maximum Principle

**Theorem:** A harmonic function $f(x)$ defined on $S$ takes on its maximum value $M$ and its minimum value $m$ on the boundaries.

**Proof (inspired by Rolle’s Theorem):** Let $M$ be the largest value of the function $f(x)$. Let $x$ be a node on $S$ for which $f(x) = M$. Then, $f(x + 1) - f(x - 1) = M$. If $x - 1$ is still an interior point, continue with $x - 2$, and so on. In the worst case, we reach $x = 0$, for which $f(x) = M$.

2.9.2. The Uniqueness Principle

**Theorem:** If $f(x)$ and $g(x)$ are harmonic functions on $S$ such that $f(x) = g(x)$ on the boundary points 0 and $N$, then $f(x) = g(x)$ for all $x$.

**Proof:** Let $h(x) = f(x) - g(x)$. Then, if $x$ is an interior point, $h(x)$ is harmonic. However, $h(x) = 0$ for $x$ at both 0 and $N$; thus, by the maximum principle, its $M$ and $m$ are both equal to 0. Therefore, $h(x)$ is always 0. That proves $f(x) = g(x)$ for all $x$.

2.9.3. Reaching the Boundary

**Theorem:** Given $h(x)$ defined on sequence $S$, a random walker will reach either 0 or $N$ in the sequence $S$. 

Proof: Let $h(x) = 1/2h(x - 1) + 1/2h(x + 1)$, so that $h(x)$ is harmonic. Therefore, $h(0) = h(N) = 0$. According to the maximum principle, $h(x) = 0$ for all $x$.

A harmonic function can be interpreted in a random walk framework. The value of the harmonic function at a given node $i$ is equal to the probability of the random walk starting at that node and reaching a positive-labeled node.

In other words, the harmonic function is a way of learning from both labeled data (i.e., the nodes where $f = 0$ or $f = 1$) and unlabeled data (i.e., all other nodes, given the structure of the graph). The harmonic function has the same values as the labeled data. For the remainder, it satisfies the averaging property $f(x) = 1/2f(x - 1) + 1/2f(x + 1)$ in the simple chain case. This is an optimization problem equivalent to minimizing the square of the differences between the node values (i.e., potentials), subject to the labels. There is a unique solution that is globally optimal: the harmonic function.

2.9.4. The Laplace Equation

An example of a physical problem that involves harmonic functions is the following: we are given a sheet of metal with a square piece removed from the center. We are looking for an expression of the temperature at all points on the metal sheet, under the condition that its outer edge is heated uniformly. Each point on the outer edge is held at $U = 1$ and each point on the inner edge is held at $U = 0$ (Figure 2.18).

Solving this problem requires the use of second-order gradients in general and Laplace’s differential equation in particular:

$$\nabla^2 u = u_{xx} + u_{yy} = 0$$
This equation is a special (steady-state) case of the (transient) heat equation given by:

\[ k \nabla^2 u = u_t \]

In general, the solutions to this equation are called harmonic functions, hence the name of the methods described in this section.

2.10. Random Walks

Imagine a person, Paul, standing on the corner of 43rd Street and Madison Avenue in Midtown Manhattan during a blackout (Figures 2.19 and 2.20). Paul is a few blocks away from his home on the corner of 47th Street and Madison Avenue. However, in the dark, he cannot see anything and wanders aimlessly from one street corner to another, going up and down Madison Avenue. At each step, he has a 50 percent chance of going up the avenue and a 50 percent chance of going down. His goal is to get home safely rather than fall in the open manhole located at the corner of 42nd Street and Madison Avenue. We assume that the random walk will stop when one of two things happens: Paul either ends up in the manhole or he arrives home safely. It can be proven mathematically that for a given probability \( p \), he

Figure 2.20. Random walk with 6 steps (\( \phi \) to \( N = 5 \)). For simplicity, node labels starting at zero are used.
eventually will end in one of those two places with a probability no larger than $p$.

If he is unlucky, Paul’s very first step will take him to the manhole. So, the probability of this event (let us call it $M$) is at least 0.5. If he is lucky, Paul will move away from the manhole in the first step; however, if after that he makes two more steps toward the manhole, he will fall into it. The probability of this happening is $0.5 \times 0.5 \times 0.5 = 0.5^3 = 0.125$. Obviously, there are many other sequences of steps that lead to the event $M$. As can be seen, the probability of the event $H = 1 - M$ (i.e., arriving home safely) seems rather small. How can we compute it?

Clearly, the value of $p_H$ is a function of the starting point. In one extreme case, Paul would start at 47th Street. In that case, he is already safely at home, so the probability $p_H(47) = 1$. In the other extreme case, he starts at 42nd Street and falls immediately into the hole, resulting in $p_H(42) = 0$. How can we find the value of $p_H(i)$ for an arbitrary street number $i$? This is an example of a random-walk model in which a walker takes random steps on the graph $G$, with the walk being modeled as a Markov process—that is, the decision on what edge to follow is solely based on the vertex where the walker is currently located. Under certain conditions, this model converges to a stationary distribution of probabilities, associated with vertices in the graph. Chapter 5 provides more details on random-walk algorithms, and specifically on the PageRank algorithm used in information retrieval.

2.11. Spreading Activation

Initiated from different theoretical backgrounds (i.e., early theories of spreading activation are due to cognitive psychology, whereas random-walk algorithms were first introduced in graph theory), models of spreading activation and random walks are connected by their common underlying graph representations and their fundamental idea of exploiting flow over a network.

Spreading activation refers to network-based models in which activation started from one or more source nodes and is propagated over the network, activating increasing numbers of more nodes, until a certain termination condition is met (usually reading a certain distance from the source nodes). In these models, it is the activation that matters; thus, the information recorded at node level is whether the node is active or passive.

The idea underlying spreading activation is the propagation of a pulse (activation) starting from one or more source nodes via links throughout
the network. The algorithm is based on a process that is divided into two main steps: (1) one or more pulses are triggered from the source nodes; and (2) a termination check determines whether the process must continue.

Assume, the same notation as in the previous chapter – that is, a directed graph $G = (V, E)$, the set of vertices that point to a node (i.e., predecessors) $In(V_i)$, and the set of vertices to which a vertex points (i.e., successors), the spreading-activation algorithm implements propagation of activation over the network, according to the following recursive formula:

$$I_p(V_i) = \sum_{V_j \in In(V_i)} (O_{p-1}(V_j))$$

where $I$ determines the input activation at pulse $p$, given the output activation $O$ at pulse $p - 1$ for the neighboring nodes. The input can be expressed using binary values (0/1) or inhibitory/reinforcing values (+1/−1). Before firing the activation to adjacent nodes, the output of a node is determined as a function of the input $O(V_i) = f(I(V_i))$, using various functions such as the linear-activation functions, sigmoid function, or – most commonly – threshold function, which sets the state of a node to active if the activation level exceeds a certain threshold. The process stops when a specific termination condition is met, which usually consists of distance or path constraints.

2.12. Electrical Interpretation of Random Walks

A connection between random walks and electrical networks was described in detail in Doyle and Snell (1984). Using a previous example (see Figure 2.20), Figure 2.21 shows the analogy between the electrical, potential and the value of the function $p_H(i)$. 

Figure 2.21. Electrical-circuit interpretation of a random walk.
To illustrate this electrical interpretation, Figure 2.22 shows how to determine the potentials at each node of a more complicated electrical network. It turns out that the potentials at the nodes of the electrical circuit are equivalent to the node probabilities in the random walk. A node’s potential is equal to the probability that a random walk starting there will reach a node with potential 1 before reaching a node with potential 0. Therefore, in the following example, the potential at a is 0, the potential at b is 1, and the potentials at c and d are in the middle. However, the potential at c is smaller than that at d. Why? Because overall, the graph is symmetric; however, the resistance between c and d is larger than that between d and b. Figure 2.23 shows the equations that must be solved to determine the potential of all nodes.

We examine the properties of electrical circuits – specifically, a closed-loop network with resistors (Figure 2.24) (the conductance at each edge

\[
P_{xy} = \frac{C_{xy}}{C_x} \quad C_x = \sum_y C_{xy} \quad C_{xy} = \frac{1}{R_{xy}}
\]

Figure 2.23. Matching equations from Doyle and Snell (1984).
Graph-Based Algorithms

Figure 2.24. An electrical circuit with a sequence of five resistors.

is 1/R). Note that resistance is additive and that from Ohm’s law $I = \frac{V}{R}$, resistance is proportional to the endpoints.

We note an additional property from Kirchhoff’s first law – specifically, for any point in an electrical circuit, the sum of currents flowing toward that point is equal to the sum of currents flowing away from that point.

Thus, we can derive a solution for determining the voltage potential at any state $V(i)$ in Figure 2.24 for $0 < i < N$ using Ohm’s law and Kirchhoff’s first law, as follows:

$$i_{xy} = \frac{V(x) - V(y)}{R}$$

$$\frac{V(x - 1) - V(x)}{R} + \frac{V(x + 1) - V(x)}{R} = 0$$

$$V(x) = \frac{V(x + 1) - V(x - 1)}{2}$$

We can generalize this to arbitrary resistances as follows:

$$V(x) = \frac{1}{R_{x-1}} V(x - 1) + \frac{1}{R_x} V(x) + \frac{1}{R_{x+1}} V(x + 1)$$

2.13. Power Method

As we showed earlier, there is a unique function (the harmonic function) that is a solution to a particular random-walk instance. If the graph is unweighted, then the value of the function at each node is the average of its values at all adjacent nodes. Furthermore, the value of the function is constrained by the so-called fixed points where its value is given in advance. In our example, this happens at the two extreme ends of the chain (i.e., 42nd and 47th Streets), where the values of the function are 0 and 1, respectively.

It can be shown easily that the value of the function at the intermediate points is proportional to their distance from the zero end of the chain. In our
example, the value of the function at each of the four intermediate nodes \( k = 1 \ldots 4 \) is equal to \( k/n \), where \( n = 5 \) is the total number of edges in the network.

2.14. Linear Algebra Methods for Computing Harmonic Functions

One method for computing graph-based harmonic functions subject to certain initial conditions is to use linear algebra. Consider the following example shown in Figure 2.25. An ant is located at node 1. It randomly walks on the graph and stops whenever it reaches the anteater (where it dies) or the anthill (where it thrives). We can model this process using harmonic functions. Let us define the survival function \( p_d \) to indicate the probability that starting from node \( s \), the ant will die.

The first observation is that \( p_5 = 1 \); similarly, \( p_3 = 0 \). Because the ant starts at node 1, we also need to compute the values of \( p_i \) at all values of \( i \) from 1 to 5. Note that the probability \( p_2 \) is actually equal to the average of \( p_1 \) and \( p_3 \) because it is equally likely to move from node 2 to either node 1 or node 3.

We now formulate all values of \( p \) as a system of equations, as follows:

\[
\begin{align*}
    p_1 &= 1/2 p_2 + 1/2 p_4 \\
    p_2 &= 1/2 p_1 + 1/2 p_3 \\
    p_3 &= 0 \\
    p_4 &= 1/3 p_1 + 1/3 p_3 + 1/3 p_5 \\
    p_5 &= 1
\end{align*}
\]
INPUT
  G : graph with NODES nodes
  PINNEDVAL[] : initial values of all pinned nodes

INITIAL STAGE
foreach I (1..NODES) {
  if (I is a pinned node) {
    VAL[I] = PINNEDVAL[I]
  }
  else {
    VAL[I] = 0.5
  }
}

ITERATIVE STAGE
while (delta > epsilon) {
  VAL[I] = AVERAGE(VAL[J] for all neighbors J of I)
}
return VAL[]

Figure 2.26. Method of relaxations.

After substituting the known values for $p_3$ and $p_5$ in the system, from the equation for $p_2$, we obtain $p_1 = 2p_2$.

We now have the following new system:

$$\begin{align*}
2p_2 &= 1/2p_2 + 1/2p_4 \\
p_4 &= 2/3p_2 + 1/3
\end{align*}$$

We now have $3/2p_2 = 1/2p_4$; therefore, $p_4 = 3p_2$:

$$\begin{align*}
3p_2 &= 2/3p_2 + 1/3
\end{align*}$$

Therefore, $7/3p_2 = 1/3$ and, therefore, $p_2 = 1/7$. After substitution, we obtain $p(1) = 2/7$ and $p(4) = 3/7$. We can stop now because we have all necessary values of $p_i$ for $i = 1 \ldots 5$: $p = (2/7, 1/7, 0, 3/7, 1)$. In other words, the probability of the ant dying after starting at node 1 is 2/7, which corresponds to a survival probability of 5/7.

2.15. Method of Relaxations

The method of relaxations (Figure 2.26) is used to compute iteratively the potentials of each node. The method is simple and efficient, and it often
converges in just a few iterations – even for large graphs with relatively small diameters. It is also fairly stable relative to its initial seeds, as shown in the following example.

The algorithm is simple. At step 0, a tentative potential is assigned to each node. For “pinned“ nodes (whose value is part of the set of initial conditions), the tentative potential is equal to the actual potential. The other nodes have arbitrary potentials (e.g., all 0 or 1, all 1/2, or all random).

Then, at each iteration, the new value of the tentative potential of each (nonpinned) node is set as the average of all of its neighbors.

As indicated in Table 2.1, the method matches the correct values of $p$ up to two decimal digits after twelve steps. It matches the correct values up to four decimal digits after another nine steps. If we had chosen different initial values, then the rate convergence would be almost the same (Table 2.2).

Table 2.1. *First set of initial values. Pinned nodes are shown in parentheses.*

<table>
<thead>
<tr>
<th>Node</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual</td>
<td>0.2857</td>
<td>0.1429</td>
<td>(0.0000)</td>
<td>0.4286</td>
<td>(1.0000)</td>
</tr>
<tr>
<td>1</td>
<td>0.5000</td>
<td>0.5000</td>
<td>0.0000</td>
<td>0.5000</td>
<td>1.0000</td>
</tr>
<tr>
<td>2</td>
<td>0.5000</td>
<td>0.2500</td>
<td>0.0000</td>
<td>0.5000</td>
<td>1.0000</td>
</tr>
<tr>
<td>3</td>
<td>0.3750</td>
<td>0.2500</td>
<td>0.0000</td>
<td>0.5000</td>
<td>1.0000</td>
</tr>
<tr>
<td>4</td>
<td>0.3750</td>
<td>0.1875</td>
<td>0.0000</td>
<td>0.4583</td>
<td>1.0000</td>
</tr>
<tr>
<td>5</td>
<td>0.3229</td>
<td>0.1875</td>
<td>0.0000</td>
<td>0.4583</td>
<td>1.0000</td>
</tr>
<tr>
<td>6</td>
<td>0.3229</td>
<td>0.1615</td>
<td>0.0000</td>
<td>0.4410</td>
<td>1.0000</td>
</tr>
<tr>
<td>7</td>
<td>0.3012</td>
<td>0.1615</td>
<td>0.0000</td>
<td>0.4410</td>
<td>1.0000</td>
</tr>
<tr>
<td>8</td>
<td>0.3012</td>
<td>0.1506</td>
<td>0.0000</td>
<td>0.4337</td>
<td>1.0000</td>
</tr>
<tr>
<td>9</td>
<td>0.2922</td>
<td>0.1506</td>
<td>0.0000</td>
<td>0.4337</td>
<td>1.0000</td>
</tr>
<tr>
<td>10</td>
<td>0.2922</td>
<td>0.1461</td>
<td>0.0000</td>
<td>0.4307</td>
<td>1.0000</td>
</tr>
<tr>
<td>11</td>
<td>0.2884</td>
<td>0.1461</td>
<td>0.0000</td>
<td>0.4307</td>
<td>1.0000</td>
</tr>
<tr>
<td>12</td>
<td>0.2884</td>
<td>0.1442</td>
<td>0.0000</td>
<td>0.4295</td>
<td>1.0000</td>
</tr>
<tr>
<td>13</td>
<td>0.2868</td>
<td>0.1442</td>
<td>0.0000</td>
<td>0.4295</td>
<td>1.0000</td>
</tr>
<tr>
<td>14</td>
<td>0.2868</td>
<td>0.1434</td>
<td>0.0000</td>
<td>0.4289</td>
<td>1.0000</td>
</tr>
<tr>
<td>15</td>
<td>0.2862</td>
<td>0.1434</td>
<td>0.0000</td>
<td>0.4289</td>
<td>1.0000</td>
</tr>
<tr>
<td>16</td>
<td>0.2862</td>
<td>0.1431</td>
<td>0.0000</td>
<td>0.4287</td>
<td>1.0000</td>
</tr>
<tr>
<td>17</td>
<td>0.2859</td>
<td>0.1431</td>
<td>0.0000</td>
<td>0.4287</td>
<td>1.0000</td>
</tr>
<tr>
<td>18</td>
<td>0.2859</td>
<td>0.1430</td>
<td>0.0000</td>
<td>0.4286</td>
<td>1.0000</td>
</tr>
<tr>
<td>19</td>
<td>0.2858</td>
<td>0.1430</td>
<td>0.0000</td>
<td>0.4286</td>
<td>1.0000</td>
</tr>
<tr>
<td>20</td>
<td>0.2858</td>
<td>0.1429</td>
<td>0.0000</td>
<td>0.4286</td>
<td>1.0000</td>
</tr>
<tr>
<td>21</td>
<td>0.2857</td>
<td>0.1429</td>
<td>0.0000</td>
<td>0.4286</td>
<td>1.0000</td>
</tr>
</tbody>
</table>

2.16. Monte Carlo Methods

Monte Carlo methods are direct computer simulations of random walks on a graph. For example, to compute the value of the potential function...
Table 2.2. *Second set of initial values. Pinned nodes are shown in parentheses.*

<table>
<thead>
<tr>
<th>Node</th>
<th>Actual</th>
<th>1</th>
<th>2</th>
<th>3 (0.0000)</th>
<th>4</th>
<th>5 (1.0000)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0000</td>
<td>1.0000</td>
<td>0.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.5833</td>
<td>0.5000</td>
<td>0.0000</td>
<td>0.6667</td>
<td>1.0000</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.4097</td>
<td>0.2917</td>
<td>0.0000</td>
<td>0.5278</td>
<td>1.0000</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.4097</td>
<td>0.2049</td>
<td>0.0000</td>
<td>0.4699</td>
<td>1.0000</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.3374</td>
<td>0.1687</td>
<td>0.0000</td>
<td>0.4458</td>
<td>1.0000</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.3072</td>
<td>0.1536</td>
<td>0.0000</td>
<td>0.4357</td>
<td>1.0000</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0.2947</td>
<td>0.1536</td>
<td>0.0000</td>
<td>0.4357</td>
<td>1.0000</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0.2947</td>
<td>0.1473</td>
<td>0.0000</td>
<td>0.4316</td>
<td>1.0000</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0.2895</td>
<td>0.1447</td>
<td>0.0000</td>
<td>0.4298</td>
<td>1.0000</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0.2873</td>
<td>0.1436</td>
<td>0.0000</td>
<td>0.4291</td>
<td>1.0000</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0.2864</td>
<td>0.1436</td>
<td>0.0000</td>
<td>0.4291</td>
<td>1.0000</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0.2864</td>
<td>0.1432</td>
<td>0.0000</td>
<td>0.4288</td>
<td>1.0000</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>0.2860</td>
<td>0.1432</td>
<td>0.0000</td>
<td>0.4288</td>
<td>1.0000</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>0.2860</td>
<td>0.1430</td>
<td>0.0000</td>
<td>0.4287</td>
<td>1.0000</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>0.2858</td>
<td>0.1429</td>
<td>0.0000</td>
<td>0.4286</td>
<td>1.0000</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>0.2858</td>
<td>0.1429</td>
<td>0.0000</td>
<td>0.4286</td>
<td>1.0000</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>0.2857</td>
<td>0.1429</td>
<td>0.0000</td>
<td>0.4286</td>
<td>1.0000</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.3. *Monte Carlo simulation.*

<table>
<thead>
<tr>
<th>Node</th>
<th>Actual</th>
<th>1</th>
<th>2</th>
<th>3 (0.0000)</th>
<th>4</th>
<th>5 (1.0000)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run 1</td>
<td>0.2860</td>
<td>0.1480</td>
<td>0.0000</td>
<td>0.4260</td>
<td>1.0000</td>
<td></td>
</tr>
<tr>
<td>Run 2</td>
<td>0.2888</td>
<td>0.1495</td>
<td>0.0000</td>
<td>0.4199</td>
<td>1.0000</td>
<td></td>
</tr>
<tr>
<td>Run 3</td>
<td>0.2856</td>
<td>0.1459</td>
<td>0.0000</td>
<td>0.4342</td>
<td>1.0000</td>
<td></td>
</tr>
<tr>
<td>Run 4</td>
<td>0.2826</td>
<td>0.1390</td>
<td>0.0000</td>
<td>0.4233</td>
<td>1.0000</td>
<td></td>
</tr>
<tr>
<td>Run 5</td>
<td>0.2852</td>
<td>0.1405</td>
<td>0.0000</td>
<td>0.4184</td>
<td>1.0000</td>
<td></td>
</tr>
<tr>
<td>Run 6</td>
<td>0.2932</td>
<td>0.1420</td>
<td>0.0000</td>
<td>0.4297</td>
<td>1.0000</td>
<td></td>
</tr>
<tr>
<td>Run 7</td>
<td>0.2847</td>
<td>0.1389</td>
<td>0.0000</td>
<td>0.4248</td>
<td>1.0000</td>
<td></td>
</tr>
<tr>
<td>Run 8</td>
<td>0.2830</td>
<td>0.1404</td>
<td>0.0000</td>
<td>0.4308</td>
<td>1.0000</td>
<td></td>
</tr>
<tr>
<td>Run 9</td>
<td>0.2827</td>
<td>0.1417</td>
<td>0.0000</td>
<td>0.4399</td>
<td>1.0000</td>
<td></td>
</tr>
<tr>
<td>Run 10</td>
<td>0.2926</td>
<td>0.1408</td>
<td>0.0000</td>
<td>0.4352</td>
<td>1.0000</td>
<td></td>
</tr>
</tbody>
</table>

Values in parentheses are “pinned” (not allowed to change from their initial values).
at node \( i \), ten thousand random walks are performed starting at that node and ending at any boundary node using the observed ratio of successful outcomes per ten thousand.

Table 2.3 illustrates a sample Monte Carlo simulation, showing the values of the nodes after ten runs of ten thousand iterations each.

Further Reading

There is a large body of literature on graph theory. The books by Diestel (2005), Gross and Yellen (2005), and Cook and Holder (2006) provide a good overview of graph algorithms and their applications.
Part II

Networks
3
Random Networks

This chapter discusses random and naturally occurring networks. We look at their properties as well as some algorithms for computing those properties.

3.1. Networks and Graphs

In the literature, the terms network and graph often are used interchangeably. Here, we use network to indicate a representation of a natural relationship among objects (in the context of this book, mostly linguistic objects) and graph for relationships generated through an automatic process. Furthermore, networks have a more complex structure than some types of graphs, such as lattices (i.e., predictable degree) or random graphs (i.e., completely arbitrary edges).

Network theory is a relatively new field. Apart from classic papers that appeared in the literature for other domains of science (e.g., economics and information science), most work in this field has been published since 1998.

The field of network theory studies the behavior of networks (mostly naturally occurring networks), for example, among people, drugs, animals, and scientific papers – under specific assumptions. It is concerned with the measurement of certain topological aspects of networks such as their degree distribution (i.e., how many nodes are connected to a specific node), the distribution of their connected components, and their diameter (e.g., the maximal distance between any two arbitrarily chosen nodes). Some properties can be computed exactly, in closed form, whereas others are computed using sampling. One goal of network theory is to predict the extrinsic behavior of a network based on its measurable properties. Network theory also encompasses generative models for different types of networks, as well as the study of processes that occur on graphs (e.g., epidemics).
3.2. Random Graphs

A fundamental concept in graph theory is that of the random graph, which can be defined in a number of ways. The simplest method is to use an adaptation of the two-parameter Erdős-Rényi (ER) model. The parameters are \( n \), the total number of nodes, and \( p \), the probability that any two nodes are connected. The ER model is usually defined on undirected graphs. There are a total of \( n(n-1)/2 \) possible pairs of nodes on a graph with \( n \) nodes; therefore, the expected number of actual edges is \( pn(n-1)/2 \).

Following is a simple example: Given \( n = 4 \) and \( p = 0.5 \), the selected edges among A-B, A-C, A-D, B-C, B-D, and C-D may happen to be A-B, A-C, and C-D. This results in a random graph shaped like a chain of four nodes. It can be seen that in this example, the graph happens to be fully connected (one can walk from any node to any other node by following edges) and that its diameter is 3 (because the shortest path that connects B and D goes through both A and C).

The ER model also is called a random graph. Two definitions of random graphs are in common use. We are not discussing individual graphs but rather ensembles (i.e., sets) of graphs. We describe such a random graph with the notation \( G_{n,p} \), where \( n = \) number of nodes and \( p = \) probability that two nodes are connected. For example, the ensemble \( G_{3,0.5} \) is shown in Figure 3.1. On a larger scale, Figure 3.2 is a sample ER random graph with fifty nodes and \( p = 0.05 \).

An important question is how well the ER random-graph model models real-life networks (including linguistic networks). It turns out that the diameters of random graphs are typically small (which matches real-life observations), but they exhibit no clustering and their degree distributions do not match those of real-life networks (which have a small number of very-high-degree hubs).

3.3. Degree Distributions

The degree distribution of a graph is the probability distribution that an arbitrarily chosen node from it will have degree \( k \).
As stated in the previous section, a random graph with \( n \) nodes has \( n(n - 1)/2 \) potential edges. The probability that any single edge is present is \( p \); therefore, the average number of edges is \( pn(n - 1)/2 \).

For a random graph, the degree distribution then can be calculated as follows:

\[
p(k) = \binom{n-1}{k} p^k (1-p)^{n-1-k}
\]

where \( p(k) \) is the probability that a node has degree \( k \). We use \( (n - 1) \) instead of \( n \) because we do not allow nodes to connect to themselves.

For large \( n \) (i.e., \( n \gg kp \)), this becomes the Poisson distribution, as follows:

\[
p(k) = \frac{\lambda^k e^{-\lambda}}{k!}
\]

where \( \lambda = 2m/n = p(n - 1) \approx pn \) is the mean degree.

The expected value is then:

\[
E(p(k)) = \lambda, \text{ the variance, } var(p(k)) = \lambda, \text{ and the mode } = \text{floor}(\lambda)
\]

except when lambda is a positive integer, in which case there are two modes: \( \lambda - 1 \) and \( \lambda \).
Figure 3.3. Poisson distributions with four different values for $\lambda$.

Figure 3.3 shows several Poisson distributions for $\lambda = 1, 3, 5, \text{ and } 10$. As shown, all of the distributions have the same general shape as well as a characteristic (i.e., most likely) value. Figure 3.4 shows the degree distribution of the ER graph.

We should also mention that the degree distributions in naturally occurring networks turn out to be of the power-law form.

Figure 3.4. Degree distribution for an Erdős-Rényi random graph.
3.4 Power Laws

Naturally occurring networks often do not display the same Poisson degree distribution as random networks. Figure 3.5 shows a power-law network; Figure 3.6 shows the degree distribution of a power-law network. Power-law networks do not have a typical degree; therefore, they sometimes are referred to as scale invariant.

Figure 3.5. A small example of a power law network.

Figure 3.6. Degree distribution of a power law degree distribution in log-log space. The $x$-axis shows the degree whereas the $y$-axis shows the count of nodes with such a degree.
Consider the following equation:

\[ \ln p(x) = C - \alpha \ln x \]

This is the equation of a straight line in log-log space (\( C \) is simply a constant). If we exponentiate both sides, we obtain:

\[ p(x) = Cx^{-\alpha} \]

which is the characteristic power-law relationship.

Briefly, one method for determining the coefficient of the power law entails this formula (Clausen et al. 2002):

\[ \alpha = 1 + n \left[ \sum_{i=1}^{n} \ln \frac{x_i}{x_{\min}} \right]^{-1} \]

where \( x_{\min} \) is a lower cutoff, below which the power law is not observed. Alternatively, consider the fact that the integral of a power law is also a power law (with an exponent that is larger by 1), thereby fitting the cumulative distribution instead of the original degree distribution.

\[ e = 1 - \frac{N \sum_{i=1}^{N} i \times x_i - \sum_{i=1}^{N} x_i \times \sum_{i=1}^{N} i}{N \sum_{i=1}^{N} i^2 - (\sum_{i=1}^{N} i)^2} \]

Furthermore, in many networks, the value of \( x \) at which the power law becomes effective may not be at \( x = 0 \) but rather at a different value, to be called \( x_{\min} \).

For details about fitting the parameters of the power model, readers are referred to (Clauset et al. 2007) in which the authors explain how to compute Maximum Likelihood Estimates (MLEs), possibly using binning, cumulative distributions, and initial cutoffs. Table 3.1 shows the parameters of power laws in real-life datasets.

### 3.5. Zipf’s Law

A common distribution in linguistics is the Zipfian distribution (also known as Zipf’s law). It is used to describe the relationship between the frequency-based rank of a word and its frequency. These two quantities have been shown to be inversely correlated. In other words, if the frequency of the most frequent word is \( f \), then the frequency of the second-most frequent word is \( f/2 \), that of the third-most frequent word is \( f/3 \), and so forth. Figure 3.7 illustrates a scale-free distribution.
Table 3.1. *Power law parameters of known distributions* (Newman 2006).

<table>
<thead>
<tr>
<th>Distribution</th>
<th>$x_{\text{min}}$</th>
<th>Exponent $\alpha$</th>
</tr>
</thead>
<tbody>
<tr>
<td>word frequencies</td>
<td>1</td>
<td>2.20</td>
</tr>
<tr>
<td>paper citation counts</td>
<td>100</td>
<td>3.04</td>
</tr>
<tr>
<td>website link counts</td>
<td>1</td>
<td>2.40</td>
</tr>
<tr>
<td>book copies sold in U.S.</td>
<td>2,000,000</td>
<td>3.51</td>
</tr>
<tr>
<td>telephone calls received</td>
<td>10</td>
<td>2.22</td>
</tr>
<tr>
<td>earthquake magnitudes</td>
<td>3.8</td>
<td>3.04</td>
</tr>
<tr>
<td>diameters of moon craters</td>
<td>0.01</td>
<td>3.14</td>
</tr>
<tr>
<td>intensities of solar flares</td>
<td>200</td>
<td>1.83</td>
</tr>
<tr>
<td>war intensities</td>
<td>3</td>
<td>1.80</td>
</tr>
<tr>
<td>net worths of Americans</td>
<td>$600m$</td>
<td>2.09</td>
</tr>
<tr>
<td>frequencies of family names</td>
<td>10,000</td>
<td>1.94</td>
</tr>
<tr>
<td>populations of U.S. cities</td>
<td>40,000</td>
<td>2.30</td>
</tr>
</tbody>
</table>

Table 3.2 lists the most common words in Jane Austen’s novel *Emma*, after lowercasing and stemming.

Remember that mathematically, the Zipfian distribution diverges (albeit slowly; see numerous proofs), which makes it not quite realistic for linguistic observations.

Many linguistic phenomena (e.g., word frequency and number of senses per word) can be described using Zipf’s as law, as can a wide range of nonlinguistic observations (e.g., the frequency of visits to YouTube videos). Historically, human wealth also has been known to be distributed according to similar patterns. In the Pareto model (i.e., a power-law model similar to Zipf’s law).
Table 3.2. Zipf example from Emma.

<table>
<thead>
<tr>
<th>Word</th>
<th>Frequency</th>
<th>Score</th>
<th>Word</th>
<th>Frequency</th>
<th>Score</th>
<th>Word</th>
<th>Frequency</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>to</td>
<td>5242</td>
<td>5242</td>
<td>all</td>
<td>848</td>
<td>24592</td>
<td>think</td>
<td>464</td>
<td>26448</td>
</tr>
<tr>
<td>the</td>
<td>5207</td>
<td>10414</td>
<td>could</td>
<td>837</td>
<td>25110</td>
<td>an</td>
<td>463</td>
<td>26854</td>
</tr>
<tr>
<td>and</td>
<td>4898</td>
<td>14694</td>
<td>would</td>
<td>821</td>
<td>25451</td>
<td>thing</td>
<td>460</td>
<td>27140</td>
</tr>
<tr>
<td>a</td>
<td>4560</td>
<td>18240</td>
<td>him</td>
<td>771</td>
<td>24672</td>
<td>ar</td>
<td>452</td>
<td>27120</td>
</tr>
<tr>
<td>i</td>
<td>4421</td>
<td>22105</td>
<td>been</td>
<td>760</td>
<td>25080</td>
<td>weston</td>
<td>445</td>
<td>27145</td>
</tr>
<tr>
<td>of</td>
<td>4295</td>
<td>25770</td>
<td>no</td>
<td>744</td>
<td>25296</td>
<td>them</td>
<td>436</td>
<td>27032</td>
</tr>
<tr>
<td>it</td>
<td>2652</td>
<td>18564</td>
<td>my</td>
<td>732</td>
<td>25620</td>
<td>everi</td>
<td>424</td>
<td>26712</td>
</tr>
<tr>
<td>her</td>
<td>2503</td>
<td>20024</td>
<td>do</td>
<td>683</td>
<td>24588</td>
<td>than</td>
<td>415</td>
<td>26560</td>
</tr>
<tr>
<td>wa</td>
<td>2374</td>
<td>21366</td>
<td>ani</td>
<td>653</td>
<td>24161</td>
<td>am</td>
<td>414</td>
<td>26910</td>
</tr>
<tr>
<td>she</td>
<td>2364</td>
<td>23640</td>
<td>miss</td>
<td>614</td>
<td>23332</td>
<td>elton</td>
<td>404</td>
<td>26664</td>
</tr>
<tr>
<td>be</td>
<td>2337</td>
<td>25707</td>
<td>were</td>
<td>601</td>
<td>23439</td>
<td>well</td>
<td>403</td>
<td>27001</td>
</tr>
<tr>
<td>in</td>
<td>2188</td>
<td>26256</td>
<td>will</td>
<td>586</td>
<td>23440</td>
<td>know</td>
<td>392</td>
<td>26365</td>
</tr>
<tr>
<td>not</td>
<td>2151</td>
<td>27963</td>
<td>me</td>
<td>583</td>
<td>23903</td>
<td>sai</td>
<td>385</td>
<td>26565</td>
</tr>
<tr>
<td>you</td>
<td>2000</td>
<td>28000</td>
<td>by</td>
<td>571</td>
<td>23982</td>
<td>your</td>
<td>373</td>
<td>26110</td>
</tr>
<tr>
<td>mr</td>
<td>1840</td>
<td>27600</td>
<td>must</td>
<td>571</td>
<td>24553</td>
<td>how</td>
<td>372</td>
<td>26412</td>
</tr>
<tr>
<td>that</td>
<td>1818</td>
<td>29088</td>
<td>which</td>
<td>556</td>
<td>24464</td>
<td>knightlei</td>
<td>372</td>
<td>26784</td>
</tr>
<tr>
<td>he</td>
<td>1811</td>
<td>30787</td>
<td>there</td>
<td>551</td>
<td>24795</td>
<td>should</td>
<td>371</td>
<td>27083</td>
</tr>
<tr>
<td>had</td>
<td>1626</td>
<td>29268</td>
<td>from</td>
<td>546</td>
<td>25116</td>
<td>good</td>
<td>364</td>
<td>26936</td>
</tr>
<tr>
<td>have</td>
<td>1468</td>
<td>27892</td>
<td>what</td>
<td>537</td>
<td>25239</td>
<td>littl</td>
<td>364</td>
<td>27300</td>
</tr>
<tr>
<td>but</td>
<td>1441</td>
<td>28820</td>
<td>thei</td>
<td>522</td>
<td>25056</td>
<td>when</td>
<td>364</td>
<td>27664</td>
</tr>
<tr>
<td>for</td>
<td>1348</td>
<td>28308</td>
<td>thi</td>
<td>509</td>
<td>24941</td>
<td>never</td>
<td>358</td>
<td>27566</td>
</tr>
<tr>
<td>with</td>
<td>1220</td>
<td>26840</td>
<td>harriet</td>
<td>500</td>
<td>25000</td>
<td>we</td>
<td>357</td>
<td>27846</td>
</tr>
<tr>
<td>veri</td>
<td>1183</td>
<td>27209</td>
<td>or</td>
<td>496</td>
<td>25296</td>
<td>did</td>
<td>353</td>
<td>27887</td>
</tr>
<tr>
<td>hi</td>
<td>1132</td>
<td>27168</td>
<td>such</td>
<td>489</td>
<td>25428</td>
<td>onli</td>
<td>327</td>
<td>26160</td>
</tr>
<tr>
<td>on</td>
<td>1129</td>
<td>28225</td>
<td>much</td>
<td>486</td>
<td>25758</td>
<td>might</td>
<td>326</td>
<td>26406</td>
</tr>
<tr>
<td>at</td>
<td>1037</td>
<td>26962</td>
<td>if</td>
<td>485</td>
<td>26190</td>
<td>now</td>
<td>313</td>
<td>25666</td>
</tr>
<tr>
<td>so</td>
<td>974</td>
<td>26298</td>
<td>said</td>
<td>484</td>
<td>26620</td>
<td>own</td>
<td>308</td>
<td>25564</td>
</tr>
<tr>
<td>emma</td>
<td>862</td>
<td>24136</td>
<td>more</td>
<td>469</td>
<td>26264</td>
<td>time</td>
<td>307</td>
<td>25788</td>
</tr>
</tbody>
</table>

The words have been stemmed, then ranked by frequency. Frequency is the observed frequency whereas score is the expected frequency a, computed by Zipf’s law.

to Zipf’s law but preceding it by several decades), the fraction of wealth occupied by the richest is as follows:

\[ w = \rho^{(\alpha-2)/(\alpha-1)} \]

The values of the Zipfian coefficient range from 0.7 to 1.2 in natural language (Zanette and Montemurro 2005). A few properties related to word frequencies have been observed to not follow Zipf’s law (e.g., the distribution of letter frequencies).

Power-law functions are often confused with the concept of a long tail (which happens when a nontrivial amount of probability mass is contained in the tail end of the distribution); however, not all long tails are power law.
3.6 Preferential Attachment

One explanation given in the literature for the emergence of power laws in degree distributions is attributed to Barabasi and Albert (1999), which we refer to as the Barabasi–Albert model. It is a temporal (i.e., growth) model that attaches newly formed nodes to existing nodes that already have more connections than average (hence, this model is called a temporal model). This model was first considered by Price (1965) in the context of citation networks. In that model, each paper is generated with a mean number of $m$ citations. New papers cite previous papers with a probability proportional to their in-degree (i.e., number of incoming citations). Papers without citations are addressed as follows: Each paper is considered to have a citation probability of degree $k$, proportional to $k + 1$. It can be proved that such a model gives rise to a power law with an exponent $a = 2 + 1/m$, which – for large values of $m$ – tends to $a = 2$.

(Note: Not all temporal models follow the Barabasi–Albert model; for example, not using degree as a factor results in an age-weighted Poisson.)

The Barabasi–Albert model (Figure 3.8) is undirected. It starts with a small cluster of $k$ vertices and $k - 1$ edges per node (i.e., fully connected graph). Then, at each iteration, new nodes are created and connected to existing nodes with probabilities proportional to the old nodes’ degrees. This method implements preferential attachment (i.e., memory-based) in which older nodes are more likely to get the highest degrees. The graph is guaranteed to be connected by design. Ultimately, it also is not highly clustered. The Barabasi–Albert model shows that the power law has the coefficient 3.
It can be shown that the diameter \( D \) is \( \ln(n)/\ln(\ln(n)) \) and the clustering coefficient \( c \) is five times larger than random. This latter point is important because it also is known that naturally occurring networks have a clustering coefficient that is three to five orders of magnitude larger than random.

3.7. Giant Component

The **giant component** of a network is a connected subnetwork such that it contains a majority of the nodes in the network, as follows:

\[
s = \max_{i \in C} \text{size}(i)
\]

where \( C \) is the set of connected components found by a depth-first search. Note that a giant connected component may not exist in all networks.

The size of the giant component can be derived as follows: Let \( v \) be the number of nodes that are not in the giant component. Then, \( u = v/n \) is the fraction of nodes that are not in the giant component.

If a node is in the giant component, its \( k \) neighbors are as well. The probability of this happening is \( u^k \):

\[
u = \sum_{k=0}^{\infty} p_k u^k = e^{-\lambda} \sum_{k=0}^{\infty} \frac{(\lambda u)^k}{k!} = e^{\lambda(u-1)}
\]

Now let \( S = 1 - u \). We have \( S = 1 - e^{-\lambda S} \). For \( \lambda < 1 \), the only non-negative solution is \( S = 0 \). For \( \lambda > 1 \) (after the phase transition), the only non-negative solution is the size of the giant component. At the phase transition, the component sizes are distributed according to a power law with exponent \( 5/2 \). Similarly, it can be proven that \( <s> = \frac{1}{1-\lambda + \lambda S} \) (Newman 2010).

3.8. Clustering Coefficient

The **clustering coefficient** of a network is a measure of the network’s transitivity – that is, if \( a \) is directly linked to \( b \) and \( b \) is directly linked to \( c \), what is the probability that \( a \) also will be directly linked to \( c \)?

Figure 3.9 shows how the clustering coefficient varies from a random network to a small-world network.

The **Watts–Strogatz Clustering Coefficient** is the average clustering coefficient of each node, as follows:

\[
cc_1 = \frac{\sum_{i=1}^{n} C(i)}{n}
\]
The Newman Clustering Coefficient is defined as follows:
\[
c_{c_2} = \frac{3 \times \text{nbtriangles}}{\text{nbtriplets}}
\]
where \(\text{nbtriangles}\) is the number of closed (i.e., connected) triangles of nodes \(i, j,\) and \(k\) of the network and \(\text{nbtriplets}\) is the number of all combinations of three nodes. Clearly, when these two numbers are the same, the clustering coefficient reaches 1, its largest possible value. For the graph in Figure 3.10, the average clustering coefficient is \(\frac{1}{4}(1 + 1 + 1 + 1) = 1\). Typical values of clustering coefficients in actual graphs can be as high as 0.5 (Newman 2002). In random graphs, \(C = p\), ignoring the fact that \(j\) and \(k\) share a neighbor \((i)\).

For the entire graph, the global clustering coefficients (CC) is the average of all vertex clustering coefficients (cc). Vertices with degree 0 or 1 are excluded from this computation.

3.9. Small Worlds

Naturally occurring networks tend to have much higher clustering coefficients than equivalently sized random networks. They also have a relatively small diameter.

Figure 3.9. Statistics of random graphs versus small-world networks.

Figure 3.10. Clustering coefficient example.
In the 1960s, Stanley Milgram analyzed social networks in the United States. In his famous experiment (Travers and Milgram 1969), he asked 296 random people from Omaha, Nebraska, and Wichita, Kansas, to forward a letter to a stockbroker in Boston, Massachusetts. For the researchers to understand the structure of their social network, participants were allowed to forward the letter only to people whom they knew personally. Among those people, they had to choose the one who was most likely to take the letter closer to the target. Only 20 percent of the letters reached the stockbroker; however, the average trip was about 6.5 hops long (which does not account for the 80 percent lost in the process).

In 2003, Dodds et al. repeated this experiment, this time using email. Their work was at a much more global scale (i.e., eighteen targets, thirteen countries, and sixty thousand participants), which resulted in 384 messages reaching their target and yielding an average path length of 4.

A network that exhibits such small diameter and that also has high transitivity is called a small-world network. The definition originated with Watts and Strogatz (1998). In addition to social networks, many other networks have been shown to be small-world networks – for example, the C. elegans neuronal network, many bibliometric networks, and the U.S. power grid.

Small-world graphs are an intermediate step between regular lattices (i.e., high clustering coefficients and high diameters) and random graphs (i.e., low clustering coefficients and low diameters). Such graphs are obtained by gradually rewiring a lattice by disconnecting an edge from an endpoint and then connecting it to a random node.

The Watts–Strogatz model uses two parameters: \( z \), the degree of each node, and \( p \), the probability of a shortcut between two random nodes. The total number of shortcuts is \( m_p = nzp/2 \).

Barthélemy and Amaral (1999) report that for an initial lattice with \( N = 1,000 \) vertices and with a coordination number \( z = 10 \), they obtain a network with diameter \( d = 7.6 \) when \( p = 0.016 \); the diameter would decrease to \( d = 3.6 \) when \( p = 0.25 \).

The clustering coefficient mirrors the structure of the underlying lattice. According to Barrat and Weigt (2000), the clustering coefficient \( C \) in fact can be modeled as follows:

\[
C = \frac{3(z-1)}{2(2z-1)} \ast (1-p)^3
\]

reaching the \( C = \frac{3}{4} \) limit when \( p \to 0 \) (see Figure 3.12).

Figure 3.11 shows how starting with a regular lattice with diameter \( n/4 \) and, after only a few random rewirings, that diameter can be reduced to a very small number compared to \( n \).
Figure 3.11. Rewiring of a regular lattice. Left to right: regular lattice; graph after only a few edge rewirings; and random graph. (Watts and Strogatz 1998).

Figure 3.12 shows how small worlds emerge for \( p \) between 0.001 and 0.1. In that region, the diameter of the graph is approximately the same as in a random graph; however, the clustering coefficient is much larger than random.

3.10. Assortativity

Assortativity is a measure of the preference of nodes to attach to other similar or dissimilar nodes. Although the similarity measure may vary we will consider degree assortativity.

Assortativity is high when high-degree nodes tend to connect to other high-degree nodes; it is low (i.e., negative) when high-degree nodes are linked to low-degree nodes. An assortativity of 0 indicates no correlation (random): \( \sigma^2 = 0 \).

Figure 3.12. Evolution of the diameter and clustering coefficients as a function of the shortcuts probability, from regular lattice (leftmost), to small-world networks (center), to random networks (rightmost) (Watts and Strogatz 1998).
The degree assortativity coefficient is given as follows:

\[ r = \frac{M^{-1} \sum_i j_i k_i - (M^{-1} \sum_i \frac{1}{2}(j_i + k_i))^2}{M^{-1} \sum_i \frac{1}{2}(j_i^2 + k_i^2) - (M^{-1} \sum_i \frac{1}{2}(j_i + k_i))^2} \]

where \( M \) is the number of edges and \( j_i \) and \( k_i \) are the degrees of the vertices at the ends of the \( i \)th edge, for \( i = 1, \ldots, M \).

Consider an example with four nodes (\( M = 4 \)), as shown in Figure 3.13. The associativity value of this graph is computed as follows. Because the value is negative and small by absolute value, we can conclude that the network is mildly disassortative:

\[
\begin{align*}
\sum_i j_i k_i &= 2 \times 2 + 2 \times 2 + 2 \times 2 + 1 \times 2 = 14 \\
\sum_i \frac{1}{2}[j_i + k_i] &= \frac{1}{2}(2 + 2) + \frac{1}{2}(2 + 2) + \frac{1}{2}(2 + 2) + \frac{1}{2}(1 + 2) = 7.5 \\
\left[\frac{1}{4} \sum_i \frac{1}{2}[j_i + k_i]\right]^2 &= 3.516 \\
\sum_i \frac{1}{2}[j_i^2 + k_i^2] &= \frac{1}{2}(2^2 + 2^2) + \frac{1}{2}(2^2 + 2^2) + \frac{1}{2}(2^2 + 2^2) + \frac{1}{2}(1^2 + 2^2) = 14.5 \\
r &= \frac{\frac{1}{4} \times 14 - 3.516}{\frac{1}{4} \times 14.5 - 3.516} = -0.143
\end{align*}
\]
3.11. Centrality

Centrality is a function of the importance of a node in a graph. The simplest type of centrality is degree centrality, which indicates the number of neighbors of a given node.

Figure 3.14 shows that a node (i.e., the one in the center) can be more central than others even though it has a smaller degree.

An example outside of natural language processing is the Erdős number, used primarily by mathematicians to show their distance to the hub of the collaboration network in mathematics – namely, Paul Erdős. Figure 3.15 is a classic example from the domain of paper coauthorship. In the center of the graph is Paul Erdős. A total of 511 people have an Erdős number of 1, whereas 8,000 have a number of 2. The second coauthor of this book has a number of 4 according to Erdos and Greenbaum (1973), Burr et al. (1972), Aho and Sloane (1973), and Aho et al. (1998).

In the domain of movies, it has been claimed that Kevin Bacon has the highest collaboration centrality of all actors (see oracleofbacon.org; www.cinfn.com; and www.thekevinbacongame.com), but studies have shown that this is not the case. Instead, Dennis Hopper (who passed away on the day this paragraph was written) is currently the most central actor on the Internet Movie Database (IMDB).

In the following sections, several methods for computing node centrality are described. One such method, eigenvector centrality, is introduced in Chapter 5.

3.12. Degree Centrality

There are two ways to compute degree centrality: raw and normalized. In the simpler (i.e., raw) case, the degree centrality of a node is simply the
number of its neighbors. This is true of undirected networks. In directed networks, *in-degree* and *out-degree* must be considered instead, which count only incoming or outgoing neighbors, respectively. The normalized degree centrality is computed by dividing the raw-degree centrality by the maximum possible number of edges that a node can have (typically, $N - 1$), thereby translating the values to a range of $0 \ldots 1$.

### 3.13. Closeness Centrality

Another metric is closeness centrality, which reflects how close to all other nodes a given node is.

Pajek (2010) uses the following formula to calculate *closeness centrality*:

$$CC(x) = \frac{1}{\sum_y d(x, y)}$$

where $d(x, y)$ is the length of the shortest path between nodes $x$ and $y$. If the network is not strongly connected, we consider only reachable vertices.

In some implementations, the centrality value for each vertex $x$ is normalized by multiplying it by $(N - 1)$, where $N$ is the number of vertices reachable from $x$. 

Figure 3.15. Erdős network (Reprinted with permission from Ronald Graham).
3.14. Betweenness Centrality

Betweenness centrality measures how important a node is by counting the number of shortest paths of which it is a part. For example, the number of paths between $B$ and $D$ in Figure 3.16 is two. The matrix of shortest paths for this graph is shown in Figure 3.17. If node $C$ is removed, a path between $B$ and $D$ remains. Therefore, $C$ is not that important to connect $B$ and $D$. If node $B$ is removed, all paths to $A$ are broken. Therefore, $B$ is important in keeping the graph connected; that is, it is quite central. Betweenness centrality on edges can be defined in the same sense.

The betweenness centrality of a node $x$ is the ratio of all shortest paths from one node to another node in the graph that pass through $x$:

$$BC(x) = \sum_{i,j \in V - \{x\}} \frac{\text{Number of shortest paths between } i \text{ and } j \text{ that pass through } x}{\text{Number of shortest paths between } i \text{ and } j}$$

Figure 3.16. Betweenness centrality example.

Figure 3.17. Matrix of shortest paths from the example.
For example, from Figure 3.16 and as shown by the shortest-paths matrix in Figure 3.17, node C lies on a shortest path between the pair A and D and a shortest path between B and D. There are two shortest paths between A and D: one passes through C; and there are two shortest paths between B and D, one passes through C. Node C lies only on those two shortest paths; therefore, the betweenness centrality of node C is as follows:

\[ BC(C) = \frac{1}{2} + \frac{1}{2} = 1 \]

This value is normalized by dividing by \((N - 1)(N - 2)\) where \(N\) is the number of nodes to give a value of 0.083.

Likewise, node B lies on the shortest paths between \((A, E)\), \((A, C)\), \((C, E)\), and \((A, D)\). The number of shortest paths between A and E is one. The number of shortest paths between A and C is one. The number of shortest paths between A and D is two. The number of shortest paths between C and E is two. Therefore, the betweenness centrality of node B is as follows:

\[ BC(C) = \frac{1}{1} + \frac{1}{1} + \frac{1}{2} + \frac{1}{2} = 3 \]

Finally, the normalized centrality is 0.25.

To find all of the shortest paths between two nodes in a graph, the Floyd–Warshall algorithm is typically used on dense graphs and Johnson’s algorithm is used on sparse graphs.

As an example, consider the Medici Family. Historically, although they started from behind, the Medici became the most prominent family in Florence, part of a dynasty that lasted for three centuries. Their prominence was due to a large extent to their marriages with other families, as shown in the network example in Figure 3.18. The Medici Family has the highest betweenness centrality just as it has the largest degree centrality. However, using betweenness, its importance seems to be even greater. A challenge to the reader is to look at all nodes with degree centrality of 0.20 and then explain why they have such different betweenness centralities.

3.15. Network Example

Consider a classic example from the social networks literature: the Florentine families network shown in Figure 3.18, which illustrates family ties in fifteenth-century Florence.

We can compute a number of properties of the Florentine network with \(n = 16\) (i.e., one single node and a giant connected component of size 15)
and \( m = 20 \). The diameter is 5 and the average degree is 2.5; assortativity is \(-0.37\). The centrality values are shown in Table 3.3. Other properties are shown in Figures 3.19 through 3.21.

There are three triangles in the network: (2, 4, 5), (2, 3, 4), and (6, 10, 12). Figure 3.22 is a summary of statistics for real networks.

<table>
<thead>
<tr>
<th>Node</th>
<th>Family</th>
<th>Degree</th>
<th>Degree Centrality</th>
<th>Betweenness</th>
<th>Closeness</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PUCCI</td>
<td>0</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>2</td>
<td>PERUZZI</td>
<td>3</td>
<td>0.2000</td>
<td>0.0190</td>
<td>0.3947</td>
</tr>
<tr>
<td>3</td>
<td>BISCHERI</td>
<td>3</td>
<td>0.2000</td>
<td>0.0905</td>
<td>0.4286</td>
</tr>
<tr>
<td>4</td>
<td>STROZZI</td>
<td>4</td>
<td>0.2667</td>
<td>0.0889</td>
<td>0.4688</td>
</tr>
<tr>
<td>5</td>
<td>CASTELLAN</td>
<td>3</td>
<td>0.2000</td>
<td>0.0476</td>
<td>0.4167</td>
</tr>
<tr>
<td>6</td>
<td>RIDOLFI</td>
<td>3</td>
<td>0.2000</td>
<td>0.0984</td>
<td>0.5357</td>
</tr>
<tr>
<td>7</td>
<td>BARBADORI</td>
<td>2</td>
<td>0.1333</td>
<td>0.0810</td>
<td>0.4688</td>
</tr>
<tr>
<td>8</td>
<td>PAZZI</td>
<td>1</td>
<td>0.0667</td>
<td>0.0000</td>
<td>0.3061</td>
</tr>
<tr>
<td>9</td>
<td>SALVIATI</td>
<td>2</td>
<td>0.1333</td>
<td>0.1238</td>
<td>0.4167</td>
</tr>
<tr>
<td>10</td>
<td>MEDICI</td>
<td>6</td>
<td>0.4000</td>
<td>0.4524</td>
<td>0.6000</td>
</tr>
<tr>
<td>11</td>
<td>ACCIAIUOL</td>
<td>1</td>
<td>0.0667</td>
<td>0.0000</td>
<td>0.3947</td>
</tr>
<tr>
<td>12</td>
<td>TORNABUON</td>
<td>3</td>
<td>0.2000</td>
<td>0.0794</td>
<td>0.5172</td>
</tr>
<tr>
<td>13</td>
<td>ALBIZZI</td>
<td>3</td>
<td>0.2000</td>
<td>0.1841</td>
<td>0.5172</td>
</tr>
<tr>
<td>14</td>
<td>GINORI</td>
<td>1</td>
<td>0.0667</td>
<td>0.0000</td>
<td>0.3571</td>
</tr>
<tr>
<td>15</td>
<td>GUADAGNI</td>
<td>4</td>
<td>0.2667</td>
<td>0.2206</td>
<td>0.5000</td>
</tr>
<tr>
<td>16</td>
<td>LAMBERTES</td>
<td>1</td>
<td>0.0667</td>
<td>0.0000</td>
<td>0.3488</td>
</tr>
</tbody>
</table>
Watts–Strogatz clustering coefficient: 0.1600
Newman clustering coefficient: 0.1915
clairlib avg. undirected shortest path: 2.49
Ferrer avg. undirected shortest path: 2.32
harmonic mean geodesic distance: 2.28
harmonic mean geodesic distance with self-loops counted in: 2.584
full average shortest path: 0.0095

Figure 3.19. Network properties of the Florentine network as computed by clairlib (Radev et al. 2007).

Another example is the social network shown in Figure 3.23, which is the co-occurrence of the characters in “Les Miserables.” Each visible cluster corresponds to one of the main logical sections of the book.

A different type of network of the Central London Underground is shown in Figure 3.24.


Networks can be studied not only as fixed objects but also as structures on which different phenomena can occur (e.g., transmission of information or infection). For example, in a social network, a rumor can be propagated from one node to many others by following the topology of the network.

Figure 3.20. Shortest paths of the Florentine network. A value of $-1$ indicates that no path exists.
3.16 Dynamic Processes: Percolation

A classic example is related to the conductivity of a square grid of atoms such that two adjacent nodes may or may not be connected. The question then is: Would a current flow through the grid if a source of power were to be attached to two opposing ends? It turns out that the probability of such a flow is a function of the fraction of edges that are actually connected. Furthermore, this function is nonlinear.

Figures 3.25 and 3.26 are examples of two types of percolation: bond percolation and site percolation. In both cases, the graph undergoes changes over time. The challenge is to find out whether the properties of the resulting

<table>
<thead>
<tr>
<th>Network</th>
<th>( n )</th>
<th>Mean degree ( z )</th>
<th>( Cc )</th>
<th>( Cc ) for random graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet (AS level)</td>
<td>6,374</td>
<td>3.8</td>
<td>0.24</td>
<td>0.00060</td>
</tr>
<tr>
<td>WWW (sites)</td>
<td>153,127</td>
<td>35.2</td>
<td>0.11</td>
<td>0.00023</td>
</tr>
<tr>
<td>Power grid</td>
<td>4,941</td>
<td>2.7</td>
<td>0.080</td>
<td>0.00054</td>
</tr>
<tr>
<td>Biology collaborations</td>
<td>1,520,251</td>
<td>15.5</td>
<td>0.081</td>
<td>0.000010</td>
</tr>
<tr>
<td>Mathematics collaborations</td>
<td>253,339</td>
<td>3.9</td>
<td>0.15</td>
<td>0.000015</td>
</tr>
<tr>
<td>Film actor collaborations</td>
<td>449,913</td>
<td>113.4</td>
<td>0.20</td>
<td>0.00025</td>
</tr>
<tr>
<td>Company directors</td>
<td>7,673</td>
<td>14.4</td>
<td>0.59</td>
<td>0.0019</td>
</tr>
<tr>
<td>Word co-occurrence</td>
<td>460,902</td>
<td>70.1</td>
<td>0.44</td>
<td>0.00015</td>
</tr>
<tr>
<td>Neural network</td>
<td>282</td>
<td>14.0</td>
<td>0.28</td>
<td>0.049</td>
</tr>
<tr>
<td>Metabolic network</td>
<td>315</td>
<td>28.3</td>
<td>0.59</td>
<td>0.0090</td>
</tr>
<tr>
<td>Food web</td>
<td>134</td>
<td>8.7</td>
<td>0.22</td>
<td>0.065</td>
</tr>
</tbody>
</table>

Figure 3.22. Parameters of some real networks (from Newman 2002).
Figure 3.23. Character connections in Les Miserables.

evolving graph evolve smoothly or they undergo phase transitions. Figure 3.27 shows the phase-transition characteristic of grid percolation.

3.17. Strong and Weak Ties

Mark Granovetter (1973) studied how people find jobs. He discovered that most job referrals are through personal contacts; however, more were from acquaintances than close friends.

Figure 3.24. Network of the Underground in Central London.
3.17 Strong and Weak Ties

Figure 3.25. Bond percolation (from Grimmett 1999).

Figure 3.26. Site percolation (from Grimmett 1999).

Figure 3.27. The phase-transition plot for modeling percolation. For values of $p$ lower than some threshold value ($x$ axis), the probability of percolation ($y$ axis) is 0. Over the threshold of $p = 0.5059$, the probability of percolation rises very steeply to 1.
He defined *strong ties* as edges in a nearly fully connected subgraph, *weak ties* or “bridges” as edges between two modules or clusters, and *missing ties* as edges required to make a subgraph fully connected. Missing ties also are created by *triadic closure* as the network clustering coefficient increases with time. Granovetter also found that a node’s linkage to local bridges provides structural advantages. For example, a bilingual employee in a primarily monolingual company would be crucial when the company wants to globalize or collaborate with other partnerships.

### 3.18. Assortative Mixing

*Assortative mixing* is the degree to which nodes with similar characteristics preferentially connect to one another. An assortative graph can be built by specifying a vector of features for each node. Two nodes are connected if their vectors are highly similar (e.g., can be measured in terms of the cosine distance or the Pearson correlation coefficient). This is the method for creating an unweighted graph. For a weighted graph, the weight of an edge can be made proportional to the similarity between the vectors of the nodes that it connects.

### 3.19. Structural Holes

When a network can be easily converted to another with a smaller diameter, we are speaking of *structural holes*. They can be filled strategically by connecting one or more links to link together with other points.
Individuals can exercise influence or act as brokers within their social networks by bridging two networks that are not directly linked.

In Figure 3.29, researchers connecting the two groups are “filling” the structural hole between the two groups, which is more likely to control the communication between them.

Further Reading

For additional reading, Herings et al. (2001) is a good overview of graph centrality measures; Newman et al. (2006) and Barabasi (2003) provide an overview of random networks; and the mathematics of small worlds are explained in detail in Watts (1999). Newman (2010) is a comprehensive study of networks and their properties.
Language Networks

This chapter discusses language networks from an empirical perspective, including co-occurrence networks, syntactic and semantic networks, similarity networks, and networks of summaries and machine translation. Language networks are used in several methods and applications in natural language processing and information retrieval, such as syntax, semantics, and summarization.

4.1. Co-Occurrence Networks

Word co-occurrence networks can be used in several applications, including discovery of semantic classes, part-of-speech tagging, and keyword extraction, among others. Although these applications are addressed in more detail in subsequent chapters, an overview of the general method for constructing word co-occurrence networks, as well as their properties, is presented here.

A co-occurrence network can be defined in many ways: two words that appear in a window of $N$ words of each other, in the same sentence, or in the same paragraph; or are connected to one another inside the network. Weighted edges then are drawn to indicate the strength of the connection between words. The networks can be directed or undirected.

In some studies, only words that appear near one another with a significant frequency are considered; in other studies, only words that belong to a certain part of speech are added to the network. To draw weighted edges, different measures can be used. One way to compute the strength of a pair is to use the log-likelihood ratio (Dunning 1993). Other possibilities include Chi-square $\sum_{i=1}^{k} \left( \frac{X_i - \mu_i}{\sigma_i} \right)^2$ and pointwise mutual information $SI(x, y) = \log \frac{p(x, y)}{p(x)p(y)}$.

According to Ferrer-i-Cancho and Sole (2001), word co-occurrence graphs show small-world characteristics (i.e., a small diameter of approximately 2.65 and a clustering coefficient of 0.5) as well as a scale-free
distribution of degrees. In their study, they examined word pairs that appear within two words of one another in running text extracted from the British National Corpus. This distance considers dependencies such as “red ink” and “hit (the) ball.” Their corpus size has $n \approx 470,000$ and $m \approx 1.7 \times 10^7$. The resulting degree-distribution graph seems to be best fit by two power laws, corresponding to $\gamma$ values of $-1.50$ and $-2.70$, respectively. The explanation for this two-regime behavior could be the presence of a kernel lexicon (which corresponds to the fifty thousand most commonly used English words) and an extended list of less commonly used words. The assumption is that only the kernel lexicon is shared by all speakers of English. Each word in the kernel lexicon has a higher degree (i.e., on average, each kernel word is linked to 24 percent of the other kernel words); therefore, it has a higher availability for speech production.

In a separate experiment, word pairs that co-occur within the two-word window with a statistically stronger frequency (i.e., collocations) also were considered, providing similar results.

The two-regime observations made by Ferrer-i-Cancho and Sole (2001) also found an explanation in a model proposed by Dorogovtsev and Mendes (2001). Starting with the preferential-attachment model previously observed in social networks (Barabasi 2003), a network is built as follows: At each step, each new node $j$ connects to an old node $i$ with a probability proportional to $k_i$ (i.e., the degree of node $i$). In the same step, a number of new edges emerge among the old words. The new edges between $i$ and $j$ appear with a probability proportional to $k_i \ast k_j$. It also was shown that the size of the kernel lexicon does not depend on the size of the language.

A variant of this study was conducted on the word-adjacency network (i.e., considering only adjacent words). Masucci and Rodgers (2006) studied the co-occurrence of words that follow one another in the novel 1984 by George Orwell (Figure 4.1 shows the first paragraph) and found very strong evidence for a power law. However, it is important that some preprocessing was performed on the nodes prior to the network construction and that punctuation also was included, which may add significant “noise.” In their
network, \( n = 8,992 \), \( m = 117,687 \), and \( \alpha = 1.1 \). The graph corresponding to the first sentence in 1984 is shown in Figure 4.2.

4.2. Syntactic Dependency Networks

After looking at the network of co-occurring words (i.e., the “word web”), Ferrer-i-Cancho, Sole, and Koehler (2004) examined syntactic universals (i.e., language properties that persist across languages) in the form of networks of syntactic dependencies using corpora from three European languages (i.e., Czech, Romanian, and German) belonging to three different language families. Similar to the co-occurrence network, these syntactic networks also were shown to display small-world properties as well as disassortative mixing (because service words like prepositions do not connect to one another).

Syntactic dependencies link “head” words with their “modifiers.” For example, the sentence, “Mary has a cat” includes three syntactic dependencies: “has Mary,” “has cat,” and “cat a” in which the node “has” is the root of the dependency tree (see Chapter 8). The corresponding networks are directed, with the arrow pointing from the modifier to the head (i.e., child to parent). It was observed that about 70 percent of such dependencies appear within a distance of two words. For example, Figure 4.3 shows the graph corresponding to the neighborhood of the word pentru in Romanian.
4.3 Semantic Networks

Semantic networks are structures used to represent semantic relationships among concepts (Sowa 1983). They can be directed or undirected and they consist of vertices (which represent the concepts) and edges (which represent the relationships between concepts). One of the most widely used

Table 4.1. Properties of syntactic dependency networks.

<table>
<thead>
<tr>
<th>Property</th>
<th>Czech</th>
<th>German</th>
<th>Romanian</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C$</td>
<td>0.1</td>
<td>0.02</td>
<td>0.09</td>
</tr>
<tr>
<td>$C_r$</td>
<td>$4 \times 10^{-4}$</td>
<td>$6 \times 10^{-6}$</td>
<td>$9.2 \times 10^{-4}$</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>2.29</td>
<td>2.23</td>
<td>2.19</td>
</tr>
<tr>
<td>$n$</td>
<td>33,336</td>
<td>6,789</td>
<td>5,563</td>
</tr>
<tr>
<td>$k_{avg}$</td>
<td>13.4</td>
<td>4.6</td>
<td>5.1</td>
</tr>
<tr>
<td>$D$</td>
<td>3.5</td>
<td>3.8</td>
<td>3.4</td>
</tr>
<tr>
<td>$D_r$</td>
<td>4</td>
<td>5.7</td>
<td>5.2</td>
</tr>
</tbody>
</table>

$C$ is the clustering coefficient of the network. $C_r$ is the clustering coefficient of an equally large random network. $\gamma$ is the power low coefficient. $n$ is the number of nodes in the graph. $k_{avg}$ is the average degree. $D$ and $D_r$ are the diameters of the network and an equally sized random graph, respectively.
Two of the first studies concerned with properties of semantic networks are attributed to Sigman and Cecchi (2002a) and Steyvers and Tenenbaum (2005b). The semantic network studied by Sigman and Cecchi (2002a) is based on several semantic relationships as shown in WordNet: polysemy and synonymy, antonymy, hypernymy and hyponymy (i.e., more general and more specific concepts), and meronymy and holonymy (i.e., part-of relationships such as branch-tree, member relationships such as tree-forest, and made-of relationships, such as glass-bottle). They note that the presence of polysemous words gives small-word properties to the resulting network of 66,025 noun meanings (i.e., “synsets” in WordNet parlance). Figure 4.4 shows synonyms for the word idea; each triangle represents a group of three (near) synonyms.

Steyvers and Tenenbaum (2005b) studied WordNet as well as an associative network described by Nelson et al. (1999). Figure 4.5 illustrates this associative network, which is based on experiments with six thousand
participants. They were given five thousand cue words (e.g., “cat”); a link between two words in the network was drawn if at least two people made that association. For instance, the edge between the nodes “dog” and “cat” indicates how many among the six thousand participants associated “dog” and “cat.” Figures 4.6 and 4.7 illustrate the mental-lexicon network.
Table 4.2. Properties of semantic networks.

<table>
<thead>
<tr>
<th>Property</th>
<th>Associative</th>
<th>Roget</th>
<th>WordNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C$</td>
<td>0.186</td>
<td>0.875</td>
<td>0.0265</td>
</tr>
<tr>
<td>$C_r$</td>
<td>$4.35 \times 10^{-3}$</td>
<td>0.613</td>
<td>$1 \times 10^{-4}$</td>
</tr>
</tbody>
</table>

The comparison is between the actual clustering coefficient and the clustering coefficient of a random graph of the same size.

In the study reported in Steyvers and Tenenbaum (2005b), it was found that several semantic networks, including the associated network described previously, Roget’s thesaurus (i.e., twenty-nine thousand words, structured in the form of a bipartite graph with one thousand semantic categories), and WordNet (i.e., one hundred twenty thousand word forms and ninety-nine thousand meanings), are all scale-free, small-world graphs. Moreover, it was found that these semantic networks have large clustering coefficients as well as good power-law distribution fits regardless of the model used (i.e., undirected or randomly directed). However, it also was found that the graphs exhibit higher clustering than the social networks in Barabasi (2003). The results from this study are summarized in Table 4.2.

The words that were found to have the highest degree were food, money, water, car, and good in the associative network; light, cut, hold, set, and turn in Roget, and break, cut, run, make, and clear in WordNet. The degree corresponds to the number of meanings and matches Zipf’s second law, the so-called Zipf law of meanings (i.e., the number of meanings of words in a language also is distributed according to Zipf).

Figure 4.7. Syntactic dependency degree distribution (log–log space). The x-axis shows the logarithm of the degree whereas the y-axis displays the corresponding frequency.
Another illustration of a semantic network, Figure 4.8, shows a sample adjective–noun bipartite network. This representation is similar to that used in Chapter 2 for dimensionality reduction; in fact, dimensionality-reduction methods such as singular-value decomposition can be applied naturally to this dataset.

4.4. Similarity Networks

Similarity networks are representations that encode pairwise relationships between the entities stored in the nodes. They can be built using the inverse of the cosine as the similarity metric between nodes. Different approaches also call for the removal of the most and/or the least frequent entities (i.e., nodes).

Sentence-similarity networks are shown in Figures 4.9 and 4.10. The former shows a network of news headlines reporting on the swine flu; the latter is about baseball. Figure 4.11 is a sample network of paper titles from the Association for Computational Linguistics Anthology.

4.4.1. Networks of Summaries

Another type of lexical networks consists of the networks of summaries, which have been studied by Pardo et al. (2006) and Antiqueira et al. (2006, 2009). Pardo’s group correlated the quality of a summary and the dynamics
of its network growth. Specifically, they checked the number of components as increasingly more words from the summary are included. In the work of Antiqueira et al. (2006), a summary is represented as a graph of sentences with edges linking nodes that share multiple-content nouns.
The best summaries are produced when considering node degree, length of shortest paths, and other network statistics.

### 4.4.2. Networks of Machine Translations

Finally, sets of networks of translations of texts were studied by Amancio et al. (2008). Specifically, starting with Portuguese-to-English and Portuguese-to-Spanish translations, networks are built based on word adjacency (content words only) in all texts, in which the weight of an edge indicates the number of times that the two words at its endpoints co-occur. Interestingly, it was found that the average out-degree in the automatic translation networks consistently exceeds the one in the manual-translation networks. In addition, the clustering coefficient of the original texts exceeds the one in the automatic translations.

We also can look at the networks of sentence similarity among multiple translations of the same input sentence. Figure 4.12 is a sample network of translations (Patel and Radev 2006).

**Further Reading**

The following papers are relevant readings on lexical networks: additional studies on semantic networks (Motter et al. 2002); networks of concepts.
in written texts (Caldeira et al. 2006); and syntactic dependency networks (Ferrer-i-Cancho et al. 2007a and b). A comparison of a number of studies on lexical networks (Mehler 2007) includes a large table that compares several studies on language networks.
Part III
Graph-Based Information Retrieval
THIS CHAPTER addresses link-analysis methods used by search engines, such as PageRank and HITS, and covers topics relevant to their application, including method stability, the combination of link- and content-based models, topic-sensitive ranking, and query-dependent link analysis.

5.1. The Web as a Graph

The Web – a common abbreviation for the World Wide Web – consists of billions of interlinked hypertext pages. These pages contain text, images, videos, or sounds and are usually viewed using Web browsers, such as Firefox or Internet Explorer. Users can navigate the Web by either directly typing the address of a Web page (i.e., the URL) inside a browser or following the links that connect Web pages among them.

The Web is a typical example of a graph, with Web pages corresponding to vertices in the graph and links between pages corresponding to directed edges. For instance, if the page http://www.unt.edu includes a link to the page http://www.cs.unt.edu and another to the page http://www.htsc.unt.edu, and the latter page in turn links to the page of the National Institutes of Health http://www.nih.gov and also back to the http://www.unt.edu page, it means that these four pages form a subgraph of four vertices with four edges, as illustrated in Figure 5.1.

Although the size of the Web is generally considered to be unknown, there are various estimates concerning the size of the indexed Web – that is, the subset of the Web that is covered by search engines. In 2009, the average estimate from several studies ranged between 15 billion and 30 billion pages in the indexed Web.
5.2. PageRank

PageRank (Brin and Page 1998) is perhaps one of the most well-known link analysis algorithms, introduced as a method for ranking the importance of Web pages as part of the Google search engine.

PageRank is a graph-centrality algorithm, from the same family of graph algorithms described in Section 3.11. Essentially, it is a way of deciding the importance of a Web page within the Web graph based on global information recursively drawn from the entire graph.

The basic idea implemented by the PageRank model is that of “voting” or “recommendation.” When one Web page links to another, it basically is casting a vote for that other page. The higher the number of votes cast for a page, the higher the importance of the page. Moreover, the importance of the page casting the vote determines how important the vote is, and this information also is taken into account by the ranking model. Hence, the score associated with a Web page is determined based on the votes that are cast for it and the score of the vertices casting those votes.

Formally, let $G = (V, E)$ be a directed graph with the set of vertices $V$ and set of edges $E$, where $E$ is a subset of $V \times V$. For a given vertex $V_i$, let $In(V_i)$ be the set of vertices that point to it (i.e., predecessors) and let $Out(V_i)$ be the set of vertices to which that vertex $V_i$ points (i.e., successors). The PageRank score of a vertex $V_i$ is defined as follows (Brin 1998):

$$S(V_i) = \frac{(1-d)}{|V|} + d \sum_{j \in In(V_i)} \frac{1}{|Out(V_j)|} S(V_j)$$

(5.1)

where $d$ is a damping factor that can be set between 0 and 1 and which has the role of integrating into the model the probability of jumping from a
given vertex to another random vertex in the graph. The factor \( d \) is usually set to 0.85 (Brin and Page 1998).

This vertex-scoring scheme is based on a random-walk model, in which a walker takes random steps on the graph \( G \), with the walk being modeled as a Markov process — that is, the decision about which edge to follow is based solely on the vertex where the walker is currently located. Under certain conditions, this model converges to a stationary distribution of probabilities that are associated with vertices in the graph. Based on the Ergodic theorem for Markov chains (Grimmett and Stirzaker 1989), the algorithm is guaranteed to converge if the graph is both aperiodic and irreducible. The first condition is achieved for any graph that is a nonbipartite graph, whereas the second condition holds for any strongly connected graph — a property achieved by PageRank through the random jumps introduced by the \((1 - d)\) factor.

In matrix notation, the PageRank vector of stationary probabilities is the principal eigenvector for the matrix \( A_{\text{row}} \), which is obtained from the adjacency matrix \( A \) representing the graph, with all rows normalized to sum to 1: \( (\bar{P} = A_{\text{row}}^T \bar{P}) \).

Intuitively, the stationary probability associated with a vertex in the graph represents the probability of finding the walker at that vertex during the random walk; therefore, it represents the importance of the vertex within the graph. In the context of Web surfing, the PageRank algorithm implements the “random-surfer model,” in which a user clicks on links at random with a probability \( d \) and jumps to a completely new page with probability \( 1 - d \). A value of 0 for the factor \( d \) means that the ranking of a page does not depend at all on its outgoing links, whereas a value of 1 indicates that the score is determined exclusively by the links with neighboring pages.

Starting from arbitrary values assigned to each node in the graph, the PageRank computation iterates until convergence below a given threshold is achieved. Convergence is achieved when the error rate for any vertex in the graph falls below a given threshold. The error rate of a vertex \( V_i \) is defined as the difference between the “real” score of the vertex \( S(V_i) \) and the score computed at iteration \( k \), \( S^k(V_i) \). Because the real score is not known a priori, this error rate is approximated with the difference between the scores computed at two successive iterations: \( S^{k+1}(V_i) - S^k(V_i) \).

To illustrate the working of the PageRank algorithm, consider the graph in Figure 5.2(a) and assume that each vertex is assigned an initial value of 0.25. The PageRank score then can be calculated by using the formula in
Figure 5.2. A sample graph: (a) directed and (b) undirected.

Equation 5.1. Running the algorithm for ten iterations leads to the scores shown in Table 5.1.

After running the algorithm through convergence, a score is associated with each vertex, which represents the “importance” of the vertex within the graph. The final values obtained after PageRank runs to completion are not affected by the choice of the initial values; only the number of iterations to convergence may be different.

Moreover, another important property of PageRank is its stability to small variations in the graph. Ng et al. (2001) showed that PageRank is stable to perturbation in the dataset, which means that small changes in the input sequence will not influence the labeling process.

Faster or more compact implementations of PageRank also have been suggested (Kamvar et al. 2003; Raghavan and Garcia-Molina 2003), with the additional time or memory savings being particularly relevant in cases such as the large, complete Web graph.

Table 5.1. Output of the PageRank algorithm run on the directed graph from Figure 5.2(a).

<table>
<thead>
<tr>
<th>Iteration</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.250</td>
<td>0.250</td>
<td>0.250</td>
<td>0.250</td>
<td>0.250</td>
<td>0.250</td>
<td>0.250</td>
</tr>
<tr>
<td>2</td>
<td>0.128</td>
<td>0.128</td>
<td>0.075</td>
<td>0.234</td>
<td>0.181</td>
<td>0.393</td>
<td>0.075</td>
</tr>
<tr>
<td>3</td>
<td>0.053</td>
<td>0.076</td>
<td>0.071</td>
<td>0.130</td>
<td>0.125</td>
<td>0.166</td>
<td>0.071</td>
</tr>
<tr>
<td>4</td>
<td>0.052</td>
<td>0.044</td>
<td>0.049</td>
<td>0.076</td>
<td>0.081</td>
<td>0.140</td>
<td>0.049</td>
</tr>
<tr>
<td>5</td>
<td>0.042</td>
<td>0.043</td>
<td>0.038</td>
<td>0.062</td>
<td>0.056</td>
<td>0.100</td>
<td>0.038</td>
</tr>
<tr>
<td>6</td>
<td>0.037</td>
<td>0.039</td>
<td>0.035</td>
<td>0.058</td>
<td>0.053</td>
<td>0.083</td>
<td>0.035</td>
</tr>
<tr>
<td>7</td>
<td>0.036</td>
<td>0.037</td>
<td>0.034</td>
<td>0.054</td>
<td>0.050</td>
<td>0.078</td>
<td>0.034</td>
</tr>
<tr>
<td>8</td>
<td>0.036</td>
<td>0.037</td>
<td>0.033</td>
<td>0.053</td>
<td>0.049</td>
<td>0.076</td>
<td>0.033</td>
</tr>
<tr>
<td>9</td>
<td>0.035</td>
<td>0.037</td>
<td>0.033</td>
<td>0.052</td>
<td>0.048</td>
<td>0.075</td>
<td>0.033</td>
</tr>
<tr>
<td>10</td>
<td>0.035</td>
<td>0.036</td>
<td>0.033</td>
<td>0.052</td>
<td>0.048</td>
<td>0.074</td>
<td>0.033</td>
</tr>
</tbody>
</table>
5.4 Weighted Graphs

Table 5.2. Output of the PageRank algorithm run on the undirected graph in Figure 5.2(b).

<table>
<thead>
<tr>
<th>Iteration</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.250</td>
<td>0.250</td>
<td>0.250</td>
<td>0.250</td>
<td>0.250</td>
<td>0.250</td>
<td>0.250</td>
</tr>
<tr>
<td>2</td>
<td>0.199</td>
<td>0.234</td>
<td>0.199</td>
<td>0.517</td>
<td>0.128</td>
<td>0.234</td>
<td>0.128</td>
</tr>
<tr>
<td>3</td>
<td>0.217</td>
<td>0.205</td>
<td>0.217</td>
<td>0.375</td>
<td>0.161</td>
<td>0.205</td>
<td>0.161</td>
</tr>
<tr>
<td>4</td>
<td>0.194</td>
<td>0.205</td>
<td>0.194</td>
<td>0.398</td>
<td>0.133</td>
<td>0.205</td>
<td>0.133</td>
</tr>
<tr>
<td>5</td>
<td>0.191</td>
<td>0.189</td>
<td>0.191</td>
<td>0.360</td>
<td>0.136</td>
<td>0.189</td>
<td>0.136</td>
</tr>
<tr>
<td>6</td>
<td>0.180</td>
<td>0.184</td>
<td>0.180</td>
<td>0.352</td>
<td>0.126</td>
<td>0.184</td>
<td>0.126</td>
</tr>
<tr>
<td>7</td>
<td>0.175</td>
<td>0.176</td>
<td>0.175</td>
<td>0.335</td>
<td>0.123</td>
<td>0.176</td>
<td>0.123</td>
</tr>
<tr>
<td>8</td>
<td>0.168</td>
<td>0.171</td>
<td>0.168</td>
<td>0.325</td>
<td>0.119</td>
<td>0.171</td>
<td>0.119</td>
</tr>
<tr>
<td>9</td>
<td>0.164</td>
<td>0.166</td>
<td>0.164</td>
<td>0.314</td>
<td>0.116</td>
<td>0.166</td>
<td>0.116</td>
</tr>
<tr>
<td>10</td>
<td>0.159</td>
<td>0.162</td>
<td>0.159</td>
<td>0.306</td>
<td>0.113</td>
<td>0.162</td>
<td>0.113</td>
</tr>
</tbody>
</table>

5.3. Undirected Graphs

Although traditionally applied on directed graphs, a recursive graph-based ranking algorithm such as PageRank also can be applied to undirected graphs, in which case the out-degree of a vertex is equal to the in-degree of the vertex. For loosely connected graphs, with the number of edges proportional to the number of vertices, undirected graphs tend to have more gradual convergence curves as compared to their directed counterparts.

For example, consider the graph in Figure 5.2(b), which is similar to the graph in Figure 5.2(a) – the only difference being that the edges are not directed. Running PageRank on this undirected graph results in the scores shown in Table 5.2.

Comparing the convergence rates in Tables 5.1 and 5.2, note that PageRank tends to converge faster when run on directed graphs. This is illustrated in Figure 5.3, which plots the convergence curves for two randomly generated graphs with 250 vertices and 250 edges – one directed and one undirected – using a convergence threshold of 0.0001. It is interesting that as the connectivity of the graph increases (i.e., larger number of edges), convergence usually is achieved after fewer iterations, and the convergence curves for directed and undirected graphs practically overlap.

5.4. Weighted Graphs

In the context of Web surfing, it is unusual for a page to include multiple or partial links to another page; hence, the original PageRank definition for graph-based ranking assumes unweighted graphs.
However, when the edges are induced between natural language texts (e.g., see the information retrieval application in Section 5.9), they may include multiple or partial links between the vertices. For such cases, it is useful to indicate and incorporate into the PageRank model the “strength” of the connection between two vertices $V_i$ and $V_j$ as a weight $w_{ij}$ added to the corresponding edge that connects the two vertices.

A new weighted PageRank was introduced by Mihalcea and Tarau (2004), which considers the weights on the edges when computing the score associated with a vertex in the graph, as follows:

$$WS(V_i) = \frac{(1 - d)}{|V|} + d \sum_{V_j \in \text{In}(V_i)} \frac{w_{ji}}{\sum_{V_k \in \text{Out}(V_j)} w_{jk}} WS(V_j)$$  \hspace{1cm} (5.2)

For example, consider the weighted graph in Figure 5.4 and assume that each vertex is assigned an initial value of 0.25. Using Equation 5.2, we can calculate the weighted PageRank score for each vertex, as shown in Table 5.3. A similar formula can be defined to integrate vertex weights.

In terms of convergence, although the final vertex scores (and, therefore, rankings) obtained after the application of PageRank on a weighted graph can differ significantly compared to the unweighted alternative, the number of iterations to convergence and the shape of the convergence curves are almost identical for weighted and unweighted graphs.
5.5. Combining PageRank with Content Analysis

Link analysis alone cannot provide a ranking that adapts to the user query. Intuitively, a good ranking should account for both the relevance of a page to the input query and the overall importance of the page. One way of combining link-analysis and content-based information retrieval is simply to total the scores obtained from the two different methods. However, Richardson and Domingos (2002) noticed that such a linear combination leads to poor results, mostly due to a skewed distribution of the PageRank scores. To address this problem, the solution that they proposed was to scale each vector of scores before totaling them so that both vectors have the same average value for the top-ten ranked results. This normalization was found to improve significantly the quality of the ranking.

5.6. Topic-Sensitive Link Analysis

The original PageRank algorithm computes one set of scores for all pages in the Web graph, regardless of the topic to which they belong. This means

Table 5.3. Output of the PageRank algorithm run on the weighted graph in Figure 5.4.

<table>
<thead>
<tr>
<th>Iteration</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.250</td>
<td>0.250</td>
<td>0.250</td>
<td>0.250</td>
<td>0.250</td>
<td>0.250</td>
<td>0.250</td>
</tr>
<tr>
<td>2</td>
<td>0.112</td>
<td>0.163</td>
<td>0.057</td>
<td>0.145</td>
<td>0.216</td>
<td>0.444</td>
<td>0.075</td>
</tr>
<tr>
<td>3</td>
<td>0.042</td>
<td>0.085</td>
<td>0.042</td>
<td>0.088</td>
<td>0.146</td>
<td>0.164</td>
<td>0.052</td>
</tr>
<tr>
<td>4</td>
<td>0.037</td>
<td>0.045</td>
<td>0.034</td>
<td>0.051</td>
<td>0.088</td>
<td>0.117</td>
<td>0.040</td>
</tr>
<tr>
<td>5</td>
<td>0.034</td>
<td>0.042</td>
<td>0.029</td>
<td>0.041</td>
<td>0.058</td>
<td>0.090</td>
<td>0.032</td>
</tr>
<tr>
<td>6</td>
<td>0.032</td>
<td>0.041</td>
<td>0.027</td>
<td>0.040</td>
<td>0.054</td>
<td>0.078</td>
<td>0.030</td>
</tr>
<tr>
<td>7</td>
<td>0.031</td>
<td>0.040</td>
<td>0.027</td>
<td>0.039</td>
<td>0.053</td>
<td>0.075</td>
<td>0.030</td>
</tr>
<tr>
<td>8</td>
<td>0.031</td>
<td>0.039</td>
<td>0.027</td>
<td>0.039</td>
<td>0.052</td>
<td>0.074</td>
<td>0.030</td>
</tr>
<tr>
<td>9</td>
<td>0.031</td>
<td>0.039</td>
<td>0.027</td>
<td>0.039</td>
<td>0.052</td>
<td>0.074</td>
<td>0.030</td>
</tr>
<tr>
<td>10</td>
<td>0.031</td>
<td>0.039</td>
<td>0.027</td>
<td>0.039</td>
<td>0.052</td>
<td>0.073</td>
<td>0.030</td>
</tr>
</tbody>
</table>
that a Web page on “politics” could have a score higher than a Web page
on “sports,” even if the query posed by a user belongs to the sports domain
and is unrelated to politics. To address this shortcoming, Haveliwala (2002)
proposed a variation of the PageRank algorithm that also accounts for the
topic of a page based on the intuition that the importance of a Web page
should depend on its relevance to the topic of the query.

**Topic-sensitive PageRank** computes a separate set of scores for each
topic under consideration and then calculates the importance of a page for
a given query by combining the scores calculated relative to the topics
relevant to the query.

Given a topic, a topic-sensitive PageRank is calculated by using a varia-
tion of the PageRank formula in Equation 5.2. Briefly, instead of assuming
that a random jump to a given page occurs with a probability of \(\frac{(1-d)}{|V|}\)
as in the original PageRank algorithm, the biased PageRank uses a probability
that accounts for the relevance of the page to the topic. That is:

\[
S_T(V_i) = (1-d)T + d \sum_{j \in \text{In}(V_i)} \frac{1}{|\text{Out}(V_j)|} S(V_j)
\]  

(5.3)

where \(T = \frac{1}{|T_j|}\) if node \(V_i\) belongs to topic \(T_j\) and \(T = 0\) otherwise. \(|T_j|\)
represents the number of nodes in the graph that belong to the topic \(T_j\). In
this way, in the random-walk interpretation, the likelihood for a walker to
arrive at the node \(V_i\) through a random jump is 0 if the node is not relevant
to the topic under consideration; otherwise, it is larger than 0.

To determine whether a node belongs to a certain topic, a classification
approach can be used. Given a number of training documents for each topic
under consideration, a page (i.e., node) can be classified automatically for
its relevance to the topics using a machine-learning algorithm such as Naïve
Bayes or SVM. Note that a node can belong simultaneously to more than
one topic because it is possible that a document concurrently addresses
multiple topics (e.g., “politics” and “history”).

For example, assume the two different topics, “politics” and “sports,” and
assume that the nodes in Figure 5.2(a) are split between the two topics as
shown in Figure 5.5. Using Equation 5.3, we can then calculate two sets of
PageRank scores, one biased toward “politics” and the other biased toward
“sports.” Table 5.4 shows the topic-sensitive scores obtained for each topic
at iteration 10. For comparison, the PageRank calculated without any bias
(i.e., iteration 10 in Table 5.1) is also shown in Table 5.4.

As shown in Table 5.4, there is a significant difference among the values
calculated under the two topics. Node \(E\), for instance, has the highest
importance among all of the other vertices under the “sports” topic; it has an average importance when a generic (i.e., no bias) PageRank is used; and it has low importance when the biasing is done with respect to “politics.”

When the topic-sensitive PageRank scores are computed for a number of different topics (the calculations can be performed offline) for a given query, the topic-sensitive PageRank scores are linearly combined based on the relevance of the topics to the query. For example, consider the query “budget cuts in sports” and assume that the query is found to belong to the topics “sports” and “politics” with probabilities of 0.90 and 0.10, respectively. The final score for each node in the graph is calculated as $0.90 \times S_S(V_i) + 0.10 \times S_P(V_i)$, where $S_S(V_i)$ is the importance of the vertex $V_i$ under the “sports” topic and $S_P(V_i)$ is the importance of the vertex $V_i$ under the “politics” topic. Thus, in the sample graph shown in Figure 5.5, the importance of node $A$ for this query is 0.027, for node $B$ it is 0.056, and so forth.

Haveliwala (2002) evaluated this approach using sixteen different topics from the Open Directory Project. Using a testbed of ten different queries, five users were asked to indicate their preference for one of two document rankings: a topic-sensitive ranking and a no-bias ranking. In eight of the ten cases, the majority of users preferred the topic-sensitive ranking, which indicates the superiority of this method over the original PageRank scoring algorithm.

Table 5.4. Output of the topic-sensitive PageRank algorithm run on the directed graph in Figure 5.5.

<table>
<thead>
<tr>
<th>Topic Bias</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sports</td>
<td>0.025</td>
<td>0.060</td>
<td>0.058</td>
<td>0.036</td>
<td>0.084</td>
<td>0.039</td>
<td>0.008</td>
</tr>
<tr>
<td>Politics</td>
<td>0.043</td>
<td>0.018</td>
<td>0.014</td>
<td>0.064</td>
<td>0.022</td>
<td>0.100</td>
<td>0.051</td>
</tr>
<tr>
<td>No Bias</td>
<td>0.035</td>
<td>0.036</td>
<td>0.033</td>
<td>0.052</td>
<td>0.048</td>
<td>0.074</td>
<td>0.033</td>
</tr>
</tbody>
</table>

Initial value assigned to each node: 0.25; number of iterations: 10.
Topic-sensitive PageRank also can be used in other scenarios in which the biasing is performed with respect to a user-specific background, such as browsing history or a set of bookmarked pages. In this scenario, pages in the Web graph receive a different importance score based on their relevance to the user’s profile and thus are likely to result in more relevant search rankings. For example, a user with a main interest in sports will obtain a different page ranking compared to a user with a main interest in computer science.

5.7. Query-Dependent Link Analysis

Another approach to personalize the PageRank ranking is to calculate a separate set of scores based on the relevance of a page to the words belonging to a given query. Rather than biasing the PageRank calculation with respect to the topic of the user’s query, Richardson and Domingos (2002) propose a method that biases the scores with respect to the query itself. That is, given a query $Q$, a query-dependent score is calculated using the following formula:

$$S_Q(V_i) = (1 - d)P_Q(V_i) + d \sum_{j \in \text{in}(V_i)} S_Q(V_j)P_Q(V_j \rightarrow V_i)$$  \hspace{1cm} (5.4)

where $P_Q(V_i)$ represents the probability that the random walker will jump to node $V_i$ given that he or she is searching for query $Q$; $P_Q(V_j \rightarrow V_i)$ represents the probability that the walker will transition from $V_j$ to $V_i$ given the search for query $Q$. Formally, these probabilities are defined as follows:

$$P_Q(V_i) = \frac{R_Q(V_j)}{\sum_{V_k \in V} R_Q(V_k)}$$  \hspace{1cm} (5.5)

and

$$P_Q(V_j \rightarrow V_i) = \frac{R_Q(V_i)}{\sum_{V_k \in \text{out}(V_j)} R_Q(V_k)}$$  \hspace{1cm} (5.6)

where $R_Q(V)$ represents the relevance of the node $V$ for the query $Q$. It can be defined simply as the number of times that the query $Q$ appears in the page represented by the node $V$.

Thus, given a query $Q$, a ranking over the importance of the pages in the graph with respect to this query is determined by sorting the corresponding scores $S_Q(V_i)$. When a query consists of more than one term, the ranking for the individual terms can be combined linearly:

$$\sum_{t \in Q} P(t)S_t(V_i)$$

where $P(t)$ represents the weight assigned to the term $t$ within the query $Q$. 

Experiments performed on two different crawls – one consisting of 1.7 million Web pages from the education domain and the other consisting of 15 million pages from the Stanford University repository – and using two testbeds of nine different queries showed that users clearly preferred the query-dependent ranking when compared to the original PageRank ranking.

In terms of efficiency, a possible drawback of the query-based ranking is the fact that a separate ranking must be computed for each query term, which could significantly slow down the performance of an information retrieval system using this approach. The efficiency issue can be addressed by precomputing at crawl time the required rankings for numerous terms in the vocabulary and by performing the calculations for only those pages that are relevant to a given term (i.e., those pages that contain the term).

5.8. Hyperlinked-Induced Topic Search

Hyperlinked-Induced Topic Search (HITS) is another graph-centrality algorithm that was designed for ranking Web pages according to their degree of "authority" (Kleinberg 1999). HITS is somewhat similar to the PageRank algorithm, the main difference being that it makes a distinction between authorities and hubs. In the context of the Web, authorities correspond to pages considered good reliable resources that have numerous incoming links. For instance, http://www.weather.com is considered a good resource for weather-related information; thus, many other pages link to it. Conversely, hubs are pages with numerous outgoing links. They are compilations of pointers to other pages – for example, http://www.bestwebbuys.com/books, which points to several book sellers – thus, it can be used as a starting point for finding good online bookstores. In general, a good hub points to a number of good authorities; conversely, a good authority is pointed to by many good hubs.

For each vertex, HITS produces the following two sets of scores – an authority score and a hub score:

\[
HITS_A(V_i) = \sum_{V_j \in \text{In}(V_i)} HITS_H(V_j) \tag{5.7}
\]

\[
HITS_H(V_i) = \sum_{V_j \in \text{Out}(V_i)} HITS_A(V_j) \tag{5.8}
\]

At each iteration, the scores are normalized so that the authority (or hub) scores for all the vertices add up to 1. Similar to PageRank, the HITS algorithm is guaranteed to converge if the graph is both aperiodic and irreducible.
Table 5.5. The HITS authority scores obtained for the directed graph in Figure 5.2(a).

<table>
<thead>
<tr>
<th>Iteration</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.091</td>
<td>0.091</td>
<td>0.091</td>
<td>0.182</td>
<td>0.182</td>
<td>0.273</td>
<td>0.091</td>
</tr>
<tr>
<td>2</td>
<td>0.075</td>
<td>0.057</td>
<td>0.132</td>
<td>0.132</td>
<td>0.208</td>
<td>0.264</td>
<td>0.132</td>
</tr>
<tr>
<td>3</td>
<td>0.069</td>
<td>0.038</td>
<td>0.149</td>
<td>0.107</td>
<td>0.218</td>
<td>0.271</td>
<td>0.149</td>
</tr>
<tr>
<td>4</td>
<td>0.067</td>
<td>0.029</td>
<td>0.156</td>
<td>0.093</td>
<td>0.221</td>
<td>0.277</td>
<td>0.156</td>
</tr>
<tr>
<td>5</td>
<td>0.068</td>
<td>0.024</td>
<td>0.160</td>
<td>0.086</td>
<td>0.221</td>
<td>0.282</td>
<td>0.160</td>
</tr>
<tr>
<td>6</td>
<td>0.069</td>
<td>0.022</td>
<td>0.161</td>
<td>0.082</td>
<td>0.221</td>
<td>0.285</td>
<td>0.161</td>
</tr>
<tr>
<td>7</td>
<td>0.069</td>
<td>0.020</td>
<td>0.162</td>
<td>0.079</td>
<td>0.221</td>
<td>0.287</td>
<td>0.162</td>
</tr>
<tr>
<td>8</td>
<td>0.069</td>
<td>0.019</td>
<td>0.162</td>
<td>0.078</td>
<td>0.221</td>
<td>0.288</td>
<td>0.162</td>
</tr>
<tr>
<td>9</td>
<td>0.070</td>
<td>0.019</td>
<td>0.162</td>
<td>0.077</td>
<td>0.221</td>
<td>0.288</td>
<td>0.162</td>
</tr>
<tr>
<td>10</td>
<td>0.070</td>
<td>0.019</td>
<td>0.163</td>
<td>0.077</td>
<td>0.221</td>
<td>0.289</td>
<td>0.163</td>
</tr>
</tbody>
</table>

Using the same directed graph in Figure 5.2(a), Tables 5.5 and 5.6 show the HITS authority and hub scores calculated for each vertex for the first ten iterations.

HITS also can be applied to undirected graphs. Moreover, a weighted version of HITS was introduced by Mihalcea (2004), which considers edge weights when computing the authority and hub scores associated with a vertex in the graph, as follows:

\[
HITS^W_A(V_i) = \sum_{V_j \in \text{In}(V_i)} w_{ji}HITS^W_H(V_j) \quad (5.9)
\]

\[
HITS^W_H(V_i) = \sum_{V_j \in \text{Out}(V_i)} w_{ij}HITS^W_A(V_j) \quad (5.10)
\]

Table 5.6. The HITS hub scores obtained for the directed graph in Figure 5.2(a).

<table>
<thead>
<tr>
<th>Iteration</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.143</td>
<td>0.190</td>
<td>0.190</td>
<td>0.333</td>
<td>0.000</td>
<td>0.000</td>
<td>0.143</td>
</tr>
<tr>
<td>2</td>
<td>0.101</td>
<td>0.182</td>
<td>0.182</td>
<td>0.394</td>
<td>0.000</td>
<td>0.000</td>
<td>0.141</td>
</tr>
<tr>
<td>3</td>
<td>0.078</td>
<td>0.174</td>
<td>0.182</td>
<td>0.421</td>
<td>0.000</td>
<td>0.000</td>
<td>0.145</td>
</tr>
<tr>
<td>4</td>
<td>0.065</td>
<td>0.168</td>
<td>0.185</td>
<td>0.434</td>
<td>0.000</td>
<td>0.000</td>
<td>0.148</td>
</tr>
<tr>
<td>5</td>
<td>0.059</td>
<td>0.164</td>
<td>0.187</td>
<td>0.439</td>
<td>0.000</td>
<td>0.000</td>
<td>0.151</td>
</tr>
<tr>
<td>6</td>
<td>0.055</td>
<td>0.162</td>
<td>0.189</td>
<td>0.442</td>
<td>0.000</td>
<td>0.000</td>
<td>0.152</td>
</tr>
<tr>
<td>7</td>
<td>0.053</td>
<td>0.160</td>
<td>0.190</td>
<td>0.444</td>
<td>0.000</td>
<td>0.000</td>
<td>0.153</td>
</tr>
<tr>
<td>8</td>
<td>0.052</td>
<td>0.160</td>
<td>0.191</td>
<td>0.444</td>
<td>0.000</td>
<td>0.000</td>
<td>0.153</td>
</tr>
<tr>
<td>9</td>
<td>0.051</td>
<td>0.159</td>
<td>0.191</td>
<td>0.445</td>
<td>0.000</td>
<td>0.000</td>
<td>0.154</td>
</tr>
<tr>
<td>10</td>
<td>0.051</td>
<td>0.159</td>
<td>0.191</td>
<td>0.445</td>
<td>0.000</td>
<td>0.000</td>
<td>0.154</td>
</tr>
</tbody>
</table>
5.9. Document Reranking with Induced Links

In the absence of explicit links among documents – which are often available among Web pages but are not present among other documents – the ranking can be based on links induced using language models.

Kurland and Lee (2005, 2006) introduced a method called **structural reranking**, which uses automatically induced links to rerank documents provided by an initial retrieval method. For instance, starting with the top fifty documents returned by a search engine, the reranking method can be used to reorder the documents so that the more relevant documents are brought toward the top of the ranking.

Unlike the explicit links between documents – which are usually provided by humans and therefore are often guaranteed to represent a true relatedness between two documents – the induced links can be noisier because they are automatically generated. Nonetheless, experiments with structural reranking revealed the effectiveness of this approach.

In their first set of experiments, Kurland and Lee (2005) explored the use of the PageRank algorithm on a graph with asymmetric links generated by using language models. Given a set of documents returned by an initial retrieval system, a graph is constructed by adding the documents as vertices in the graph and then drawing weighted edges by using generation probabilities among documents. Centrality scores then are calculated for all documents in the graph using the weighted PageRank in Equation 5.2; consequently, the documents are reranked in reverse order of their centrality scores.

The intuition behind the asymmetric document-generation probabilities is illustrated in Figure 5.6. Considering a simple language model, the probability of the word *Graph* based on the first document is 1; however, the probability of the other two words, *Information* and *Retrieval*, based on
the same first document is 0. Thus, the generation probability for the second document based on the first document is \(1 \times 0 \times 0 = 0\). In reverse, the probability of each of the three words based on the second document is \(1/3\), which means that the generation probability of the first document based on the second document is \(1/3 \times 1/3 \times 1/3 = 1/27\).

In Kurland and Lee (2004), the generation probability between two documents is calculated based on the Kullback–Leibler divergence (Kullback 1987) between the language models of the documents; however, other vector-relatedness measures also can be used.

Overall, structural reranking was found to improve over the baseline consisting of the ranking provided by the initial information retrieval method. It also is interesting that the relative performance is consistent, meaning that no significant performance variations were found with respect to the quality of the original document ranking.

Further experiments led to additional findings. First, a comparison between the use of weighted and unweighted graphs for document-centrality scoring demonstrated that weighted graphs lead to better performance. For this comparison, unweighted graphs were constructed by drawing an edge between two documents if a generation probability existed, without adding the probability into the edge weight.

Second, a comparison between simple degree centrality (see Section 3.12) and recursive centrality computed with PageRank showed that the recursive formulation performs better by considering information drawn from the entire graph of documents.

Finally, the addition of a query-based language model was found to enhance performance by multiplying a document-query-generation probability by the document-centrality score. This is consistent with the previous use of PageRank for document ranking, which usually combines the Web graph-centrality scores with the traditional content-based information retrieval models.

A related set of experiments addressed the use of the HITS algorithm on graphs constructed using induced links (Kurland and Lee 2006). As mentioned previously, HITS makes a distinction between authorities and hubs; thus, for the application of this centrality algorithm, it is important to define the role of the entities in the graph. A possible distinction is documents versus clusters of documents, in which the clusters can be obtained using techniques such as nearest-neighbor clustering. The links between clusters and documents can be induced by using the same language-model-generation probabilities as for the PageRank graphs.
With this formulation, the documents take the role of authorities and the clusters take the role of hubs, or the reverse. This distinction can lead to three different settings for HITS graphs: (1) document-to-document graphs, in which a document is assigned both authority and hub scores; (2) document-as-authority graphs, in which the documents are authorities and the clusters are hubs; and (3) document-as-hub graphs, in which the documents are hubs and the clusters are authorities.

As with the previous PageRank experiments, the application of the HITS algorithm on graphs of documents with induced links was found to improve the initial ranking provided by an information retrieval system. The document-as-authority setting gave the best overall performance, followed by the document-as-hub setting; both settings exceeded the performance obtained using document-to-document graphs. These results demonstrated that mutually reinforced relationships among clusters and documents can be useful for structural reranking and that such cluster-document bipartite graphs are superior to document-only graphs.

Further Reading

For additional information on methods of link-based analysis and their applications, the following papers are recommended: bibliometrics and author co-citation analysis (White and Griffith 1981); document reranking using affinity graphs (Zhang et al. 2005) and interdocument distances (Baliński and Danilowicz 2005); stable algorithms for link analysis (Ng et al. 2001); and query expansion using random walks (Collins-Thompson and Callan 2005).
6

Text Clustering

This chapter addresses text clustering using a variety of graph-based algorithms, including the Fiedler method, the Kernighan–Lin method, text categorization with min-cuts, betweenness, and random walks.

Text clustering is a technique that is frequently used to organize collections of documents, and it is often used in conjunction with other tasks such as information retrieval and topic identification. The goal of text clustering is to partition a given set of documents into meaningful classes, without assuming a predefined list of categories. For example, given a collection of documents that contain the word apple, a text categorizer would divide the documents into two major clusters, one pertaining to computers (related to the meaning of Apple Macintosh) and the other pertaining to food and agriculture (related to the meaning of apple fruits and trees). A sample set of documents and the resulting clusters are illustrated in Figure 6.1.

Clustering can be either flat, in which all clusters are at the same level, or hierarchical, in which the clusters are organized into a hierarchy, which is often represented as a tree structure, or a dendrogram. Assuming the example in Figure 6.1, a hierarchical clustering algorithm would further divide the cluster on food and agriculture into two smaller clusters, one pertaining to apple trees and the other to apple fruits. The corresponding dendrogram is shown in Figure 6.2.

The quality of an automatically generated set of clusters is usually measured using two metrics considered standard in clustering evaluation—namely, purity and entropy (Zhao and Karypis 2001), which are computed for the automatically generated sense groupings relative to the “gold-standard” clusters (i.e., clusters that are considered correct for the purpose of evaluation).

The entropy shows how the various gold-standard clusters are distributed within each automatically generated cluster. The entropy of a cluster $C_r$ of
The size \( n_r \) is defined as follows:

\[
E(C_r) = -\frac{1}{\log q} \sum_{i=1}^{q} \frac{n_r^i}{n_r} \log \frac{n_r^i}{n_r}
\]

where \( q \) is the number of clusters in the gold standard and \( n_r^i \) is the number of documents from the gold-standard cluster \( i \) found in cluster \( r \). The overall entropy of a clustering solution is calculated as the weighted sum of the entropy values of individual clusters, in which each cluster is weighted by the number of documents forming the cluster and the overall sum is normalized with the total number of documents:

\[
Entropy = \sum_{r=1}^{k} \frac{n_r}{n} E(C_r)
\]

Figure 6.1. Two clusters over a set of six documents.

Figure 6.2. A dendrogram.
The entropy takes its values in the range of 0 to 1, in which the smaller values show the better clustering solution.

The purity of an automatically generated cluster is defined as the fraction represented by the largest cluster of documents from the gold standard assigned to that automatically generated cluster, as follows:

\[ P(C_r) = \frac{1}{n_r} \max_i (n_r^i) \]

The overall purity of a clustering solution is the weighted sum of the individual cluster purities:

\[ Purity = \sum_{r=1}^{k} \frac{n_r}{n} P(C_r) \]

6.1. Graph-Based Clustering

The goal of graph-based clustering is to partition the nodes of the graph into communities such that each community is relatively dense internally and, at the same time, there are only a few links across communities. For instance, Figure 6.3 is an example of a graph with three inherent communities. If we were to identify and remove the four intercommunity edges, the graph would be split into three communities with three, five, and seven internal edges, respectively.
To determine the optimal clustering, two metrics usually are calculated. The first is the **intracluster density** $\delta(C)$, which is defined as the ratio of the number of existing intracluster edges and the number of possible intracluster edges given the number of nodes $N_c$ in the cluster, as follows:

$$\delta(C) = \frac{\text{intracluster_edges} \in C}{N_c(N_c - 1)/2}$$

Similarly, the **intercluster density** $\epsilon(C)$ is the ratio of the number of existing intercluster edges and the number of possible intercluster edges given the number of nodes $N_c$ in the cluster, as follows:

$$\epsilon(C) = \frac{\text{intercluster_edges} \in C}{N_c(N - N_c)}$$

To illustrate, for the graph shown in Figure 6.3, the intracluster-density values $\delta(C)$ are calculated as $7/10 = 0.7$, $4/6 = 0.75$, and $3/3 = 1.0$. The intercluster density $\epsilon(C)$ is determined as $2/35$, $3/32$, and $3/27$.

Given a collection of documents and a similarity metric that measures the degree of similarity of any two documents, one common approach for building clustering graphs is to use fully connected graphs and the Gaussian kernel, as follows:

$$w_{i,j} = \exp\left(-\frac{||x_i - x_j||^2}{2\sigma^2}\right)$$

where $x_i$ and $x_j$ are the matching features (e.g., pixels or word counts) for a given pair of documents.

The maximum value of $w_{i,j}$ is 1 when $x_i = x_j$ (identical documents), whereas it approaches 0 when the two documents are very different. An alternative is to use a $k$ nearest-neighbors graph, in which all the documents within a given distance are connected by an edge (i.e., based on Euclidean distance). This method is asymmetric but can be made symmetric by taking the union of nearest neighbors in both directions. The original distance between the neighbors may or may not be considered when running the various algorithms. Finally, a third approach is to link each node to all nodes that are at most a certain distance from them in Euclidean space.

To illustrate, Figure 6.4 is an example of a clustering graph built from Optical Character Recognition (OCR) bitmaps, which connects digit handwritings that are most similar to one another.

Another example is illustrated in Figure 6.5, which shows that partitioning on a graph may be quite different from partitioning in Euclidean space. It represents two communities that are difficult to separate without using
graph similarity. The graph representation allows algorithms to enforce the principle that locally similar nodes (i.e., those that are directly connected in the graph) should belong to the same community.

Figure 6.6 shows two classic examples of datasets that are difficult to cluster using Euclidean distance but that is easier using a graph-based,
local-ambiguity method. This example is well known from the machine-learning literature.

6.2. Spectral Methods

Some of the most powerful methods for partitioning a graph into two clusters are based on \textit{spectral analysis}. When applied to a matrix, the word \textit{spectrum} indicates the set of all its eigenvalues. Correspondingly, spectral methods are based on clustering applied to the eigenvectors of a graph’s matrix representation.

Given a graph representation of a document collection, clustering spectral methods are based on the spectrum of either the adjacency matrix of the graph or its \textit{Laplacian} (see graph representations in Chapter 1). The unnormalized Laplacian of an undirected graph $G$ without multiple edges or self-loops is defined as follows:

$$L(G) = D - G$$

where $D$ is the diagonal \textit{degree matrix} $d_1 \ldots d_n$ and $G$ is the adjacency matrix of the graph. For example, for the graph shown in Figure 6.7, with the following matrix representation:

$$G = \begin{pmatrix}
0 & 1 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 0
\end{pmatrix}$$
the diagonal *degree matrix* is:

\[
D = \begin{pmatrix}
2 & 0 & 0 & 0 & 0 \\
0 & 2 & 0 & 0 & 0 \\
0 & 0 & 2 & 0 & 0 \\
0 & 0 & 0 & 3 & 0 \\
0 & 0 & 0 & 0 & 1
\end{pmatrix}
\]

and the Laplacian is:

\[
L = \begin{pmatrix}
2 & -1 & 0 & -1 & 0 \\
-1 & 2 & -1 & 0 & 0 \\
0 & -1 & 2 & -1 & 0 \\
-1 & 0 & -1 & 3 & -1 \\
0 & 0 & 0 & -1 & 1
\end{pmatrix}
\]

Spectral clustering algorithms are based on one or more of the eigenvectors \( \lambda_i \) of the Laplacian. For instance, one spectral algorithm is based on the first eigenvector of the Laplacian \( \lambda_1 \), which can be computed easily by the *Lanczos method* (Lanczos 1950). Doing so avoids having to compute the entire set of eigenvectors, which would be an \( O(n^3) \) operation. Another spectral method relies on the second smallest eigenvalue \( \lambda_2 \) of the Laplacian of a graph \( G \), also called the graph’s *algebraic connectivity* (see the following section).

Let us consider an example. The graph in Figure 6.8 has nine nodes and can be split intuitively into two components by cutting either the edge 3-5 or the edge 5-6. This example can be extended to a semi-supervised setting (Zhu and Goldberg 2009), and the graph can be ordered so that the labeled nodes come first. In that case, the Laplacian matrix can be split into four, as follows:

\[
M = \begin{bmatrix}
L_{ll} & L_{lu} \\
L_{ul} & L_{uu}
\end{bmatrix}
\]
6.3 The Fiedler Method

The solutions are:

\[ f_i = y_l \]
\[ f_u = -L^{-1}_{uu} L_{ul} y_l \]

The \( u \) part of the matrix corresponds to the unlabeled data points.

Because spectral methods require the calculation of one or more eigenvectors, which is a computationally intensive process, spectral partitioning generally does not scale to millions of vertices, however, it is suitable for a number of natural language processing problems with dozens of nodes.

6.3. The Fiedler Method

The *Fiedler algorithm* is straightforward and relies on the eigenvector \( v_2 \) that corresponds to the second eigenvalue \( \lambda_2 \) of the Laplacian (i.e., the Fiedler vector). Figure 6.9 shows the pseudocode for the Fiedler algorithm.

To illustrate, consider the example shown in Figure 6.10, which consists of a graph with eight nodes modeling a word-meaning problem. In the

```
Given a graph G
Compute \( \lambda_2 \)
Compute the corresponding \( v_2 \)
For each node \( i \) of G
  If \( v_2(i) < 0 \)
    Assign \( i \) to cluster \( C_1 \)
  Else if \( v_2(i) > 0 \)
    Assign \( i \) to cluster \( C_2 \)
  Else if \( v_2(i) = 0 \)
    Assign \( n \) to cluster \( C_1 \) or \( C_2 \) at random
```

Figure 6.9. Fiedler method for spectral partitioning.
graph, four nodes correspond to words that are synonymous with the “very warm” sense of the word *hot* and four others are synonymous with its “spicy” sense. The edges represent the fact that two adjectives have been used to modify the same noun or set of nouns. The Laplacian for the graph in Figure 6.10 is as follows:

\[
\begin{bmatrix}
2 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\
-1 & 3 & -1 & -1 & 0 & 0 & 0 & 0 \\
-1 & -1 & 4 & -1 & 0 & -1 & 0 & 0 \\
0 & -1 & -1 & 2 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 3 & -1 & -1 & -1 \\
0 & 0 & -1 & 0 & -1 & 4 & -1 & -1 \\
0 & 0 & 0 & 0 & -1 & -1 & 3 & -1 \\
0 & 0 & 0 & 0 & -1 & -1 & -1 & 3
\end{bmatrix}
\]

The corresponding Fiedler vector is shown in Table 6.1. After partitioning, the two resulting clusters are (boiling, warm, tropical, hot) and (tasty, peppery, highly seasoned, spicy).

6.4. The Kernighan–Lin Method

As shown in the previous example on clustering the meaning of *hot*, the graph exhibits a clustered structure – namely, two clusters connected to one
6.5 Betweenness-Based Clustering

Another graph-clustering method is the Girvan–Newman algorithm (Girvan and Newman 2002), which repeatedly removes the edge that has the highest betweenness centrality. The algorithm is illustrated in Figure 6.13.

Table 6.1. Fiedler vector.

<table>
<thead>
<tr>
<th>Node</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>boiling</td>
<td>-0.3825</td>
</tr>
<tr>
<td>very warm</td>
<td>-0.3825</td>
</tr>
<tr>
<td>hot</td>
<td>-0.2470</td>
</tr>
<tr>
<td>tropical</td>
<td>-0.3825</td>
</tr>
<tr>
<td>tasty</td>
<td>0.3825</td>
</tr>
<tr>
<td>peppery</td>
<td>0.2470</td>
</tr>
<tr>
<td>seasoned</td>
<td>0.3825</td>
</tr>
<tr>
<td>spicy</td>
<td>0.3825</td>
</tr>
</tbody>
</table>

6.5. Betweenness-Based Clustering

another in a minimal way. This “minimal way” indicates that a min-cut can be used to split the network into two communities.

The Kernighan–Lin method (Kernighan and Lin 1970) attempts to partition the nodes of a graph into two disjoint subsets of predefined size such that the sum of the weights of the edges among nodes in the two subsets is minimized. The pseudocode of the algorithm is shown in Figure 6.11.

Kernighan–Lin is a hill-climbing algorithm and is often used on large networks. It always produces two clusters, and the size of each cluster must be known in advance. Figure 6.12 illustrates the clustering output for the example in Figure 6.10.

Input: a graph with an even number of nodes N.
Randomly assign each node to one of the two partitions A and B
2: Compute the cost Q of the current partition
   (Q = number of edges within groups = number edges across groups)
Find a pair of nodes N1 and N2 that can be swapped without increasing the cost of the split
If no such pair exists, exit and return the current partition
Else swap N1 and N2 and go to step 2

Figure 6.11. Kernighan–Lin algorithm.
<table>
<thead>
<tr>
<th>Node</th>
<th>Item</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>boiling</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>very warm</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>hot</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>tropical</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>tasty</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>peppery</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>highly seasoned</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>spicy</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 6.12. Output of Kernighan–Lin algorithm on the graph in Figure 6.10.

1. Compute all edge centralities.
2. Remove the edge with the largest betweenness centrality.
3. Recalculate the centralities and restart.

Figure 6.13. Girvan and Newman’s edge-betweenness-centrality algorithm.

The betweenness centrality of an edge corresponds to the number of shortest paths between nodes that go through that edge. The betweenness can be computed in $O(mn)$ time, where $m$ is the number of edges and $n$ is the number of nodes in the graph. To perform the actual partitioning, the running time is $O(m^2n)$ because at most there are $m$ edges to be removed. One way to speed up the computation is to look only at a sample of vertices at each iteration. The following example corresponds to the graph in Figure 6.14:

Figure 6.14. Sample graph with seven nodes.
6.6 Min-Cut Clustering

According to the Ford–Fulkerson theorem (see Section 2.5), the minimum cut between any two nodes $S$ (source) and $T$ (target) in a graph carries the maximum flow that can be moved from $S$ to $T$ through the graph. This min-cut can be used to partition the graph into clusters.

A min-cut can be performed in both an unsupervised and a transductive way when labels are associated with the nodes in the graph. In the former case, we are looking for the smallest set of edges that block flow from the source $S$ to the target $T$. The size of the cut is the sum of weights of the edges in it. After the edges that belong to the min-cut are removed from the graph, the resulting graph consists of two clusters: one including the $S$ node and one including the $T$ node.

Iteration 1: Removed edge $(B, D)$
Iteration 2: Removed edge $(A, F)$
Iteration 3: Removed edge $(C, F)$
Iteration 4: Removed edge $(G, A)$
Iteration 5: Removed edge $(G, B)$
Iteration 6: Removed edge $(A, B)$
Iteration 7: Removed edge $(E, D)$
Iteration 8: Removed edge $(E, C)$
Iteration 9: Removed edge $(D, C)$

The dendrogram

\[
\begin{align*}
E &= 0\,1\,2\,3\,4\,5\,0 \\
D &= 0\,1\,2\,3\,4\,4\,3 \\
B &= 0\,0\,0\,0\,0\,0\,6 \\
F &= 0\,1\,1\,2\,3\,3\,4 \\
A &= 0\,0\,0\,0\,1\,1\,2 \\
G &= 0\,0\,0\,1\,2\,2\,1 \\
C &= 0\,1\,2\,3\,4\,4\,5
\end{align*}
\]

Partition 1: A B G
Partition 2: F
Partition 3: C D E
Text Clustering

The random-walk interpretation of the min-cut for an undirected graph is as follows. The probability that a long walk along the graph will end in a node $i$ is the same as the degree of that node. We can prove that the total probability of the random walk going from one partition to the other in the long run is the same as the value of the $Cut$, which is determined as the sum of weights of the edges along the min-cut. In other words, $Cut$ bipartitions the graph such that random walks starting in each part are more likely to remain there.

$$\lambda_2 = 1 - \text{Cut}$$

The harmonic function value $f(i)$ also is related to a random walk on the graph. There, it means that if the random walk starts at node $i$, it will reach a node labeled 1 before it reaches one that is labeled 0. Harmonic functions minimize:

$$\sum_{i,j \in G} w_{ij} (f(i) - f(j))^2$$

where $f(i) \in [0, 1]$. Conversely, the objective function minimized by min-cuts is given by:

$$\sum_{i,j \in G} w_{ij} |f(i) - f(j)|$$

Therefore, the only difference between the cut and the harmonic function is that the former is a discrete function and it tries to define a decision boundary, whereas the latter attempts to do the same but uses a continuous function over the graph.

One drawback of the min-cut method is that it tends to choose partitions of uneven sizes. For example, given an input chain of $n$ nodes, it may put one node in one cluster and the remaining $n - 1$ nodes in another. This problem can be addressed with the normalized-cut method, which in addition to minimizing the cut between the graph partitions seeks to maximize the intrapartition-node similarity (see Section 2.5.1):

$$p^T L p = 4 \times \text{cut}(V_1, V_2)$$

where $V_1$ and $V_2$ are the two partitions. This means that putting everything in one cluster (e.g., $V_2 = 0$) is a lowest-cost partition. In practice, the cuts are expected to be reasonably balanced in size.

The min-cut method also can be generalized to clusters of unequal sizes using the ratio-cut method (Wei and Cheng 1989, 1990).
6.7. Text Clustering Using Random Walks

Typical text-clustering algorithms are based on vector representations of the documents, in which each dimension of the vector corresponds to a term in a collection of documents. Although successful to some extent, these representations have an important drawback because they do not account for the fact that texts generated by a similar underlying language model should belong to the same cluster.

To address this drawback, documents can be represented as vectors of generation probabilities rather than bags of words (Erkan 2006). This representation is based on the induced links described in Section 5.9, except that the documents included in the graph are not those returned by an information retrieval system but instead they are the documents in the corpus to be clustered.

Starting with a corpus of documents, a graph is constructed by adding all of the documents as vertices in the graph. Generation probabilities among the documents then are used to draw weighted edges. Next, a t-step random walk is applied, which reinforces the existing generation probabilities and discovers inferential relationships among documents.

A t-step random walk implements the intuition that two documents that are not connected to one another but have links to a common third document should be related semantically. Thus, the t-step random walk leads to a t-step generation probability \( \text{Gen}_t(d_i|d_j) \), which is an inferred link among documents that are not directly linked to one another:

\[
\text{Gen}_t(d_i|d_j) = \frac{\sum_{s=1}^{t} q_{d_i d_j}^s}{t}
\]

(6.1)

where

\[
q_{d_i d_j}^t = \prod_{i=0}^{t-1} q_{d_i d_{i+1}}
\]

(6.2)

and

\[
q_{d_i d_{i+1}} = \frac{\text{Gen}(d_i|d_{i+1})}{\sum_{t \in C} \text{Gen}(d_i|t)}
\]

(6.3)

where \( C \) represents the corpus of documents.

In this representation, \( \text{Gen}_t(d_i|d_j) \) is a measure of the likelihood of a random walk started at \( d_i \) to visit \( d_j \) in \( t \) steps or less, and each document
is represented as a vector consisting of the t-step generation probabilities to all of the other documents in the corpus.

To cluster documents, any clustering algorithms can be applied to these vectors, such as K-means or hierarchical clustering. However, instead of applying these algorithms to the traditional term frequency/inverse document frequency (tf.idf) vector-space representation, the clustering is applied on the generation-link vectors, which explicitly account for the relationships among the documents.

When tested on standard document-clustering datasets, the t-step random-walk method led to significant improvements over the tf.idf representation (Erkan 2006). The best results are obtained for a step of three \((t = 3)\), meaning that generation probabilities are used only for documents that are reachable with a three-step random walk in the graph representation of the corpus. For larger values, the graphs become very dense, also suggesting a “small-world” effect in such generation graphs.

**Further Reading**

The following papers are recommended for further reading: a large survey of several graph-clustering algorithms (Lancichinetti and Fortunato 2009); semi-supervised learning with graphs (Zhu 2006; Zhu and Goldberg 2009a); sparse-graph partitioning (De Wit 2000); cuts for image segmentation (Shi and Malik 2000); weakly supervised graph-based methods for classification (Radev 2004); bipartite spectral-graph partitioning for co-clustering of documents and words (Dhillon 2001); learning with graph min-cuts (Blum and Chawla 2001); and semi-supervised learning with harmonic mixtures (Zhu and Lafferty 2005).
Part IV
Graph-Based Natural Language Processing
7
Semantics

This chapter addresses the application of graph-based algorithms to problems in the area of semantics. There has been growing interest in the automatic semantic analysis of text to support natural language processing applications, ranging from machine translation and information retrieval to question answering and knowledge acquisition. Significant research has been carried out in this area, including work on word-sense disambiguation, semantic-role labeling, textual entailment, lexical acquisition, and semantic relations.

The chapter describes synonym detection and automatic construction of semantic classes using measures of graph connectivity on graphs built from either raw text or user-contributed resources; measures of semantic distance on semantic networks, including simple path-length algorithms and more complex random-walk methods; textual entailment using graph-matching algorithms on syntactic or semantic graphs; word-sense disambiguation and name disambiguation, including random-walk algorithms and other structural approaches for knowledge-based word-sense disambiguation, as well as semi-supervised methods using label propagation on graphs; and sentiment classification using semi-supervised graph-based learning or prior subjectivity detection with min-cut/max-flow algorithms.

7.1. Semantic Classes

Some of the largest graph representations constructed to support a natural language processing task are perhaps those built from large text collections for unsupervised lexical acquisition (Widdows and Dorow 2002). One of the immediate applications of such large graphs is the construction of semantic classes by automatically extracting from raw corpora all of the elements belonging to a certain semantic category (e.g., “fruits” or “musical instruments.”)
The method starts by constructing a large graph consisting of all nouns in a large corpus (e.g., British National Corpus) linked by the conjunction and or or. A cutoff value is used to filter out rare words, resulting in a graph of almost 100,000 nouns, linked by more than half-million edges. To identify the elements of a semantic class, first, a few representative nouns are selected manually and used to form a seed set. Next, in an iterative process, the node found to have the largest number of links with the seed set in the co-occurrence graph is selected as potentially correct and thus added to the seed set. The process is repeated until no new elements can be added reliably to the seed set, in which the reliability is determined based on the number of links that the new element has with existing elements in the graph. Figure 7.1 illustrates a sample graph built to extract semantic classes.

An evaluation against ten semantic classes from WordNet indicated an accuracy of 82 percent – significantly better than previous work in semantic-class extraction (e.g., Riloff and Jones 1999). The drawback of this method, however, is the low coverage, given that the method is limited to those words found in a conjunction relationship. Nonetheless, whenever applicable, the graph representation has the ability to identify precisely the words belonging to a semantic class.

Another research area related to the construction of graphs from large textual corpora is the study of lexical network properties (Ferrer-i-Cancho and Sole 2001), which is addressed in detail in Chapter 4. By building large lexical networks of nearly a half-million nodes with more than ten million
edges constructed by linking words appearing in English sentences within a distance of at most two words, it has been demonstrated that complex system properties hold on such co-occurrence networks.

Specifically, there is a small-world effect with a relatively small number of two–three jumps required to connect any two words in the lexical network. Additionally, it has been observed that the distribution of node degrees inside the network is scale-free, which reflects the tendency of a link to be formed with an already highly connected word. Perhaps it is not surprising that the small-world and scale-free properties observed over lexical networks automatically acquired from corpora also were observed on manually constructed semantic networks such as WordNet (Sigman and Cecchi 2002a; Steyvers and Tenenbaum 2005a).

7.2. Synonym Detection

Semantic classes can have different granularities, and they range from large classes that encompass a large number of entities (e.g., “animal”) to very small classes that cover only one entity at a time. In the latter case, the relationship between the elements in the class is that of synonymy, which indicates that the words in the class have identical or very similar meaning.

Synonyms are essential for a number of language-processing applications including query expansion, text similarity, and co-reference resolution. The identification of synonyms is a simple task when a comprehensive thesaurus is available but becomes a challenging problem when the only resource is a raw corpus or a regular dictionary.

A graph-based method that has been used successfully for the automatic detection of synonyms for a given word is a topic-sensitive random walk applied on graphs built from Wikipedia and Wiktionary (Weale et al. 2009). The method begins by constructing a graph that exploits the structure of these user-contributed resources, which include numerous articles connected to one another by user-created links. These connections form a large graph structure with vertices consisting of Wikipedia or Wiktionary articles and edges formed by the explicit links among articles.

Given this graph and an input word, a variation of the topic-sensitive random walk described in Section 5.6 can be used to obtain a list of candidate synonyms by setting a bias on the input word. The random walk results in a list of words ranked by their importance relative to the given word, which represents the candidate synonyms. In addition, the overlap between the N-best neighbors determined by the random walk for the input word and for each candidate synonym provides another relatedness score that can be
used to measure the degree of synonymy between the candidate synonym and the word.

The random-walk synonym-detection method was demonstrated as an effective solution for synonym detection, with an accuracy that exceeds the performance of other methods such as word relatedness on the Roget thesaurus or co-occurrence-based metrics. In fact, when evaluated on datasets consisting of lists of synonyms from Tests of English as a Foreign Language (TOEFL) evaluations, the method was found to be up to 93 percent accurate in detecting the right synonym for a given word.

7.3. Semantic Distance

In addition to synonymy, words can be semantically connected without sharing a meaning. In this case, we speak of semantic distance, which measures the closeness of words in a semantic space. Semantic distance can take the form of similarity or relatedness, depending on the relationship that holds among the words. For instance, “animal” and “dog” are similar and related, whereas “doctor” and “hospital” are related but not similar.

Graph-based algorithms have been used successfully to identify word similarity and relatedness. Given a semantic network, such as the one illustrated in Figure 7.2, the semantic-distance task consists of finding the distance (i.e., similarity or relatedness) between two concepts such as “dog” and “wolf.”

A large class of methods for semantic distance consists of metrics calculated on existing semantic networks such as WordNet and Roget by applying, for instance, shortest-path algorithms that identify the closest semantic relationship between two input concepts. The following measures,
which build on this idea, have been used and cited frequently in the literature (see Budanitsky and Hirst [2006] for a comprehensive overview).

The similarity measure introduced by Leacock and Chodorow (1998) is determined as follows:

$$\text{Sim}_{\text{ich}} = - \log \frac{\text{length}}{2 \times D}$$ (7.1)

where \(\text{length}\) is the length of the shortest path between two concepts using node-counting and \(D\) is the maximum depth of the input taxonomy. For instance, assuming the network in Figure 7.2, the similarity between “dog” and “wolf” is calculated as 0.47, whereas the similarity between “wolf” and “bear” is calculated as 0.30.

The Wu and Palmer (1994) similarity metric measures the depth of two given concepts in the taxonomy and the depth of the least common subsumer concept (LCS) and then combines these figures into a similarity score, as follows:

$$\text{Sim}_{\text{wup}} = \frac{2 \times \text{depth}(\text{LCS})}{\text{depth}(\text{concept}_1) + \text{depth}(\text{concept}_2)}$$ (7.2)

For instance, in the sample network shown in Figure 7.2, the LCS for “dingo” and “bear” is determined as “carnivore.”

In addition to the LCS of the two input concepts, the measure introduced by Resnik (1995) incorporates the information content as derived from a large corpus. The measure returns the information content (IC) of the LCS of two concepts, as follows:

$$\text{Sim}_{\text{res}} = IC(\text{LCS})$$ (7.3)

where the IC is defined as \(IC(C) = - \log P(C)\) and \(P(C)\) is the probability of encountering an instance of concept \(C\) in a large corpus.

The probability \(P(C)\) is measured as the number of occurrences of words representing the concept \(C\) itself, as well as all of its subsumed instances, divided by the total number of words in the corpus. Moreover, when a word is ambiguous, the number of occurrences of that word is equally divided between its possible senses. For example, if the concept under consideration is “animal,” then we count all the occurrences of the word “animal,” as well as those of “dog,” “wolf,” “cat,” and so forth. For those words that are ambiguous – for example, assume that cat has five senses – the number of occurrences of “cat” is divided by five before adding it up to the overall concept count.

A related metric is introduced in Lin (1998), which builds on Resnik’s measure of similarity and adds a normalization factor consisting of the IC
of the following two input concepts:

\[
Sim_{\text{lin}} = \frac{2 \times IC(LCS)}{IC(\text{concept}_1) + IC(\text{concept}_2)}
\]  

Lin’s metric is similar to the measure introduced by Jiang and Conrath (1997), which combines the IC of the two concepts and their LCS in a different way, as follows:

\[
Sim_{\text{jnc}} = \frac{1}{IC(\text{concept}_1) + IC(\text{concept}_2) - 2 \times IC(LCS)}
\]

The semantic distance between words also can be measured by using topic-sensitive random-walk algorithms as described in Section 5.6. In this method, the PageRank algorithm is used to calculate the stationary distribution of nodes in the WordNet graph, biased on each input word in a given word pair (Hughes and Ramage 2007).

First, a graph is constructed from WordNet: The nodes are created by adding all of the synsets (e.g., “dog” as a noun with sense #3), words with a part of speech (e.g., “dog” as noun), and tokens (e.g., “dog” regardless of being a verb or a noun). The edges are drawn using the semantic relationships available in WordNet, such as is-a and part-of, as well as measures of association derived based on annotated corpora. This results in an enormous graph, with more than 400,000 nodes and as many as 5 million edges. Next, each word in the input pair is mapped to one or more nodes in this graph, where each node corresponds to a possible sense. The nodes then are assigned with an initial weight, which acts as a bias in the topic-sensitive version of the PageRank (see Section 5.6). Running PageRank through convergence results in a vector consisting of the stationary distribution of the nodes in the entire WordNet graph, which also accounts for the initial bias associated with the input word. Finally, a measure of vector similarity is used to determine the divergence between the distributions corresponding to each of the two words, which also reflects their relatedness.

A similar method also can be applied when the graph is built from other resources. For instance, it can be used on a graph built from Wikipedia, in which the nodes are formed by the articles in Wikipedia and the edges are created by content, categorical, or infobox links (Yeh et al. 2009). After some filtering steps, which eliminate the articles with fewer than two thousand non stopwords or fewer than five incoming or outgoing links, the resulting graph can have as many as one million vertices and thirty million edges. The distance between two words is calculated then by measuring the vector similarity between the random-walk distributions on this graph,
using a bias that assigns a uniform weight to all articles that refer to each input word.

Measures of semantic distance typically are evaluated against gold-standard datasets consisting of word pairs with associated human judgments of relatedness. One such standard is the Miller–Charles dataset (1998), consisting of thirty-word pairs ranging from synonymy pairs (e.g., “car”—“automobile”) to completely unrelated terms (e.g., “noon”—“string”). The relatedness of each word pair in this dataset was rated by thirty-eight human subjects, using a scale from 0 (not related) to 4 (perfect synonymy).

Comparative evaluations of the shortest-path measures on a network built from WordNet found the metric introduced by Jiang and Conrath to work best, with a Pearson correlation of 0.85 as measured on the Miller–Charles dataset (Budanitsky and Hirst 2006). This is comparable to the results obtained on a network built from another resource: that is, the semantic distance measured using shortest-path algorithms on a network built from the Roget thesaurus can lead to a correlation with human ratings as high as 0.87 when measured on the same dataset (Jarmasz and Szpakowicz 2003).

Even better results can be obtained with the random-walk algorithm applied to WordNet. When evaluated on the Miller–Charles dataset, the method was found to improve significantly compared to other measures of relatedness with a Spearman correlation of 0.90, coming close to the upper bound represented by the inter-annotator agreement on this dataset. Not far below is the accuracy of the same random walk applied to a graph built from Wikipedia, which has a Spearman correlation of up to 0.82 obtained when words not covered by Wikipedia were excluded.

Finally, the random-walk method also can be applied to the semantic similarity of pieces of text, in which the graph is biased with respect to all words in an input text (Ramage et al. 2009). Similar to the evaluations run at word level, the biased random walk was found to improve compared to the methods based on shortest-path measures of semantic distance (Mihalcea et al. 2006), even if only by a small margin.

7.4. Textual Entailment

The semantic distance can be calculated not only between words but also between texts. A specific type of text relatedness is **textual entailment**, which targets the identification of a directional inferential relationship between texts. Given a pair of two texts as input, typically referred to as text and hypothesis, the textual-entailment task consists of automatically finding whether the hypothesis is entailed by the text. For instance, given the text
“Yahoo purchased Overture” and the hypothesis “Overture was sold to Yahoo,” a textual entailer would find an entailment relationship between them.

*Graph-matching* techniques have proven useful for finding such inferential relationships between texts (Haghighi et al. 2005; Rus et al. 2005). First, both input texts are turned into a graph by using the dependency relationships obtained from a parse tree. For example, Figure 7.3 is a sample dependency graph obtained for the sentence “Yahoo purchased a company.”

Next, a matching score is calculated, using a graph-edit distance that accounts for both vertex and edge matches. The vertex-matching function seeks to identify all possible vertices in the text that match a given vertex in the hypothesis and then normalizes these matches by a normalization constant. The match will have an increasingly higher cost depending on whether the match is an exact match, a stem match, a synonym match, a more general match, a similar WordNet concept, a match with the same part of speech, and so forth.

The edge-matching function considers all possible paths in the text graph that link the two vertices connected by the given edge in the hypothesis. The cost of all of the matches again is normalized by a constant. The matching function is looking for exact match, partial match (i.e., an exact match but with a different label), or matches between the parents or ancestors of the input vertices. For example, Figure 7.4 illustrates the matching between the text “Yahoo purchased a company” and the hypothesis “Yahoo acquired Overture in 2003.”

Textual entailment usually is evaluated on collections consisting of text–hypothesis pairs manually labeled for entailment (Dagan et al. 2005). On these datasets, the graph-matching algorithm can achieve accuracy figures competitive with other state-of-the-art supervised methods for entailment.
Figure 7.4. Graph matching for entailment recognition.

recognition, and it significantly exceeds the performance of simpler methods such as bag-of-words matching or vector–space similarity.

7.5. Word-Sense Disambiguation

Another topic of interest in lexical semantics is word-sense disambiguation, defined as the problem of identifying the most appropriate meaning of a word given its context. For instance, the English noun plant can mean “green plant” or “factory”; similarly, the French word feuille can mean “leaf” or “paper.” Most of the work in this area assumes the availability of a predefined sense inventory such as WordNet and consists of methods that can be classified broadly as knowledge-based, supervised, or semi-supervised.

One of the earliest knowledge-based methods is the semantic memory developed by (1968), which was proposed as a representational format that enables the storage of the meaning of words or other cognitive units and makes the use of such meaning possible for other applications. The model consists of a mass of nodes connected by associative links, which can be simple associations or more complex attributes of the cognitive units. It was introduced as a representation of “the memory that a person calls upon in his everyday language behavior,” which can allow for associations and configurations that could be as varied and rich as ideas expressed by humans in natural language.
Years later, spreading-activation networks have started to receive significant attention. Spreading activation refers to graph-based models in which activation that was started from one or more source nodes is propagated over the network, activating increasingly more nodes, until a certain termination condition is met (usually the distance from the source nodes). In these models (see Section 2.11), it is the activation that matters; thus, the information recorded at node level is whether the node is active or passive.

Spreading activation was first used as a method for solving lexical ambiguity of words through simultaneous identification of word senses and frame-case slots (Hirst 1988). It was later applied to a large neural network built by relating words through their dictionary definitions (Veronis and Ide 1990): Words are connected to their dictionary senses, which in turn are connected to the words found in their definitions. Interesting in this method is the use of both activatory links (between words and senses) and inhibitory links (between competing senses of the same word). Figure 7.5 illustrates a sample graph with activated nodes, adapted from Veronis and Ide (1990).

Early work on spreading activation was evaluated on only a few ambiguous words, mainly due to the prohibitive computational costs associated with the construction of the activation network. A large-scale evaluation of the spreading-activation method was done only recently (Tsatsaronis et al. 2007), when the method introduced in Veronis and Ide (1990) was applied to a graph built from WordNet, with edges between senses drawn by using semantic relationships such as synonymy and antonymy. On a dataset consisting of word senses from Senseval-2, the method was found to exceed the random baseline by a large margin and to be competitive with other graph-based algorithms (Mihalcea et al. 2004).
Random-walk methods also have been used for knowledge-based word-sense disambiguation. Using starting-point resources such as WordNet, a graph is built for an input text by adding all possible senses for words in the text as nodes in the graph. The nodes then are connected on the basis of the semantic relationships available in the resource, such as synonymy and antonymy (Mihalcea et al. 2004).

A random walk applied to this graph results in a set of scores that reflects the “importance” of each word sense in the given text. Consequently, the word senses with the highest score are selected as potentially correct. An evaluation of sense-annotated data showed that this graph-based algorithm was superior to alternative knowledge-based methods that did not make use of such rich representations of word-sense relationships.

The method also can be generalized so that it does not require the availability of semantic relationships such as those defined in WordNet. Instead, weighted edges can be derived by using a measure of lexical similarity among word-sense definitions, followed by the application of a weighted random walk, as described in Section 5.4 (Mihalcea 2005; Sinha and Mihalcea 2007). This results in generality because the method is not restricted to semantic networks such as WordNet, and it can be used on any electronic dictionary. For instance, Figure 7.6 is an example of a graph constructed for the sentence “The church bells ring on Sunday,” with edges weighted by the degree of similarity between word senses. The score obtained after running the random walk is indicated next to each word sense.
In addition to random walks, other graph-centrality algorithms can be applied to word-sense graphs derived from WordNet or other resources (Navigli and Lapata 2007; Sinha and Mihalcea 2007). Depending on the method used to construct the graph, among the centrality methods described in Section 3.11, comparative evaluations found that the best word-sense disambiguation accuracy is achieved by using a closeness measure (in which the graph consists of the entire WordNet network) or an in-degree metric (in which the graph is built from the text to be disambiguated, with edges based on WordNet similarity).

Another graph-based word-sense disambiguation algorithm, which provided the best results to date, is based on a topic-sensitive random walk, as described in Section 5.6 (Agirre and Soroa 2009). The walk is biased with respect to the words in the input text, followed by a topic-sensitive PageRank on a graph built by using WordNet semantic relationships, which also includes relationships between disambiguated definitions and word senses. The evaluations that were run on the SENSEVAL-2 and SENSEVAL-3 datasets showed that this method can outperform the basic PageRank algorithm (Mihalcea 2005; Sinha and Mihalcea 2007) by accounting more closely for the input text.

Finally, when annotated data are available, a graph-based method that has been used successfully for semi-supervised word-sense disambiguation is the label-propagation algorithm (Niu et al. 2005). In this method, a graph is constructed from all of the labeled and unlabeled examples provided for a given ambiguous word. The word-sense examples are used as nodes in the graph, and weighted edges are drawn by using a pairwise metric of similarity. On this graph, all of the known labeled examples (i.e., the seed set) are assigned with their correct labels, which then are propagated throughout the graph across the weighted links. In this way, all of the nodes are assigned with a set of labels, each with a certain probability. The algorithm is repeated through convergence, with the known labeled examples being reassigned with their correct label at each iteration. In an evaluation carried out on a standard word-sense disambiguation dataset, the performance of the algorithm was found to exceed the one obtained using monolingual or bilingual bootstrapping and even with support vector machine learning algorithms when only a few labeled examples are available.

7.6. Name Disambiguation

A task closely related to the problem of word-sense disambiguation is name disambiguation, which targets the resolution of name ambiguity in
text. For instance, “George Miller” could refer to the Princeton University psychologist, the U.S. congressman, or the movie producer, among others.

Although most of the supervised or unsupervised word-sense disambiguation techniques also can be applied to name disambiguation, a method that has been used successfully used for the contextual disambiguation of names in emails relies on random walks on graph structures (Minkov et al. 2006). In this method, the graphs are built to encode email-specific items, including the name of the sender, the name of the receiver, the date, the subject, the content, and so forth. The nodes are linked by edges reflecting relationships among email fields such as “sent-by,” “date-of,” and “has-subject-term.” Given a graph built from a corpus of emails and an ambiguous name in the context of an input email, a “lazy” topic-sensitive random walk is performed to identify the most likely name from a set of possible candidates. The random walk is similar to the model described in Section 5.6, the main difference being the “lazy” component, which adds a probability that the random walk would stop at a given node.

For example, consider again the name “George Miller” appearing in the context of an email, which is unambiguous to the recipient of the email but can be ambiguous otherwise. Running a random walk biased toward the text of the given email, the name is resolved to the correct reference by selecting the person node that has the highest score after the convergence of the walk.

When applied to the disambiguation of names in several email corpora, the method provided an accuracy higher than 75 percent in all of the datasets, which is significantly better than the performance of a strong baseline consisting of a cosine similarity applied between the input and other emails in the dataset.

7.7. Sentiment and Subjectivity

Sentiment and subjectivity analysis is an area related to both semantics and pragmatics, which focuses on the automatic identification of private states such as opinions, emotions, sentiments, evaluations, beliefs, and speculations in natural language. Whereas subjectivity classification labels text as either subjective or objective, sentiment classification adds another level of granularity by further classifying subjective text as either positive, negative, or neutral.

Numerous subjectivity- and sentiment-analysis tools that have been developed to date rely on subjectivity and polarity lexicons, which can be constructed manually or compiled automatically.
Among the methods for automatic lexicon construction, clustering over graphs has been used successfully to identify the polarity of adjectives (Hatzivassiloglou and McKeown 1997). In this method, the input graph is constructed from large corpora by identifying adjectives that have the same orientation as in “good and friendly,” as well as adjectives with an opposite orientation as in “beautiful but selfish.” The adjectives form the vertices in the graph and they are linked with edges weighted by a dissimilarity score reflecting their orientation: A low dissimilarity value indicates adjectives that share the same orientation, whereas a high dissimilarity connects adjectives with different orientation. Adjective pairs that have no connecting links are assigned a default neutral dissimilarity of 0.5. The dissimilarity score is learned automatically by using rules based on conjunctions (e.g., “and” implies the same orientation, whereas “but” implies an opposite orientation), along with a log-linear model trained on a small manually annotated dataset.

When the graph is constructed, a clustering algorithm applied to it assigns the adjectives into two clusters, corresponding to a positive or a negative orientation. On datasets with different degrees of link sparseness – ranging from 730 nodes with 7 links per node to 230 nodes with 10 links per node – the classification accuracy is between 78 and 92 percent, exceeding by a large margin the random and the most common baselines.

The sentiment orientation of adjectives also can be measured by using graphs built from lexical resources such as WordNet (Kamps et al. 2004). Starting with the synonymy relationship encoded in the WordNet synsets, a large graph is built over all of the words defined in the resource. The affective load of a word then is measured by using Charles Osgood’s theory of semantic differentiation. Specifically, given an adjective $A$, its semantic orientation is calculated as follows:

$$SO(A) = \frac{D(A, \text{bad}) - D(A, \text{good})}{D(\text{bad}, \text{good})}$$

(7.6)

where $D(X, Y)$ represents the geodesic distance, calculated by using the shortest path in this graph between nodes $X$ and $Y$. Figure 7.7 shows the distances for a few sample adjectives.

When compared to the list of 1,600 adjectives from the General Inquirer lexicon with manually assigned polarity labels (Stone 2000), the accuracy of the automatically generated lexicon was measured at 68 percent, which is reasonable given the efficiency of the method.

In addition to the construction of lexicons, another important direction of research in subjectivity and sentiment analysis is the automatic labeling of corpora at the sentence or document level.
A graph-based method that has been used effectively for sentence-level subjectivity analysis is a min-cut algorithm that generates subjective extracts of movie reviews, which in turn can be used to improve methods for sentiment analysis (Pang and Lee 2004). First, a graph is constructed by adding all of the sentences in a review as nodes and by drawing edges based on sentence proximity. Each node in the graph is initially assigned with a score indicating the probability of the corresponding sentence being subjective or objective, based on an estimate provided by a supervised subjectivity classifier trained on manually labeled data. A min-cut algorithm then is applied to the graph and used to separate subjective and objective sentences from one another. Figure 7.8 illustrates the graph constructed over the sentences in a text in which the min-cut algorithm is applied to identify and extract the subjective sentences. The precision of this graph-based subjectivity classifier was found to be better than the labeling obtained with the initial supervised classifier. Moreover, a sentiment-analysis classifier relying on the min-cut subjective extracts was found to be more accurate than one that was applied to entire reviews.

Figure 7.8. Subjectivity classification using a min-cut algorithm. The dotted line represents the split between subjective and objective sentences, as obtained with the min-cut algorithm.
The automatic labeling of the polarity of entire documents can be achieved effectively using label propagation with graph-based semi-supervised learning, using graphs built on both labeled and unlabeled texts (Goldberg and Zhu 2006). The graph consists of documents connected among them using edges weighted by the interdocument similarity and also connected to the polarity classes using either a priori known labels (for labeled data) or the prediction of an automatic classifier (for unlabeled data). The semi-supervised learning then is achieved by smoothing the graph with respect to the polarity classes, meaning that an edge with a large weight (which indicates similar documents) should correspond to a small difference in polarity and vice versa. The smoothing task is formulated as an optimization problem, and the solution is found by learning the optimal parameters on a development dataset. When tested on a dataset of movie reviews, the graph-based semi-supervised method had significantly better accuracy as compared to learning methods that ignore the unlabeled examples during training. Moreover, the graph method was found to perform well with minimal training data and also have resistance to noise in the data.

Recent research on sentiment and subjectivity analysis also has considered the relationship between word senses and subjectivity (Wiebe and Mihalcea 2006). In work targeting the assignment of subjectivity and polarity labels to WordNet senses, a topic-sensitive random-walk algorithm was applied to the entire WordNet graph (Esuli and Sebastiani 2007). Similar to some extent to the label-propagation method discussed in Section 7.5, the random-walk algorithm is seeded with nodes labeled for subjectivity and polarity, followed by a ranking of all of the senses with respect to their relevance for given subjectivity (i.e., subjective/objective) or polarity (i.e., positive/negative) labels. When compared to a simpler classification method, the random walk was found to result in more accurate annotations of subjectivity and polarity of word senses.

With the same goal of assigning subjectivity labels to word senses, experiments also were run using semi-supervised min-cut algorithms applied to a graph built from WordNet (Su and Markert 2009). The graph was initialized with two vertices – source and sink – corresponding to the subjective and objective label and using a representation similar to the one shown in Figure 7.8. The training word senses, with preassigned subjectivity labels, are associated with the corresponding source or sink node using a high constant weight. The test and unlabeled senses also are associated with the source and sink nodes, using a supervised classifier to determine the edge weights. Finally, all of the senses inside the graph are connected...
using the existing WordNet semantic relationships, such as hypernymy and synonymy, with edge weights identified based on their prevalence among subjective or objective senses inside the training data. An evaluation of the min-cut algorithm on a subset of WordNet senses showed that the method can outperform a supervised Naive Bayes classifier applied to the same dataset.

Further Reading

There is a growing body of work on the use of graph representations and algorithms for lexical semantics. In addition to the material addressed in this chapter, the following papers are recommended: conceptual graphs (Sowa 1979); lexical chaining (Dias et al. 2006; Medelyan 2007); semantic relationships (Nastase and Szpakowicz 2006); learning of semantic classes (Kozareva et al. 2008); semantic-role labeling (Fürstenau and Lapata 2009); and learning of textual entailments (Zanzotto and Moschitti 2006).
This chapter covers the use of graph-theoretical algorithms for tasks in syntax, including part-of-speech tagging using graphs that encode word and tag dependencies; dependency parsing using minimum spanning trees; prepositional attachment using word-dependency distributions induced from random-walk models; and co-reference resolution using graph-clustering and min-cut algorithms.

8.1. Part-of-Speech Tagging

Part-of-speech tagging is defined as the task of automatically assigning parts of speech to words. For instance, given the text, “This is a book,” a part-of-speech tagger identifies that “this” is a pronoun, “is” is a verb, “a” is a determiner, and “book” is a noun. Part-of-speech tagging is required by almost any text-processing task, including word-sense disambiguation, parsing, and semantic analysis. As one of the first processing steps in any such application, the accuracy of the part-of-speech tagger directly impacts the accuracy of any subsequent text-processing steps.

Although most of the part-of-speech taggers developed to date rely on machine-learning algorithms with features drawn from the surrounding text of an ambiguous word, there are also other approaches such as unsupervised tagging using clustering algorithms. In particular, Biemann’s part-of-speech tagger (2006c) is based on the idea of word co-occurrence. First, a bipartite graph of words that appear next to one another is built, followed by the calculation of the second power of that graph’s connectivity matrix, thereby connecting words that appear in the same context. Hence, words like red, green, and blue appear in the same cluster because they are distributionally similar (Lee 1997).

The clustering method is called Chinese Whispers. It was introduced by Biemann and Teresniak (2005) and later discussed in Biemann (2006a). The Chinese Whispers algorithm works as follows: First, each node is
assigned a separate label. Then, at each iteration, each node is assigned the most popular label in its neighborhood. For example, Figure 8.1 shows two iterations of the Chinese Whispers algorithm for clustering a graph with eleven nodes.

When applied to the British National Corpus, after excluding the most frequent two thousand words and drawing edges between words if they shared at least four left and right neighbors, the algorithm produced 282 clusters. The following table illustrates the largest clusters as obtained by the algorithm:

<table>
<thead>
<tr>
<th>size</th>
<th>tags</th>
<th>sample words</th>
</tr>
</thead>
<tbody>
<tr>
<td>18432</td>
<td>NN(17120), AJ(631)</td>
<td>secret, officials, transport</td>
</tr>
<tr>
<td></td>
<td></td>
<td>unemployment, farm, country</td>
</tr>
<tr>
<td></td>
<td></td>
<td>wood, procedure, officials</td>
</tr>
<tr>
<td>4916</td>
<td>AJ(4208), V(343)</td>
<td>busy, grey, tiny, sufficient</td>
</tr>
<tr>
<td></td>
<td></td>
<td>attractive, vital</td>
</tr>
<tr>
<td>4192</td>
<td>V(3784), AJ(286)</td>
<td>filled, revealed, experienced</td>
</tr>
<tr>
<td></td>
<td></td>
<td>learned, pushed, occurred</td>
</tr>
<tr>
<td>3515</td>
<td>NP(3198), NN(255)</td>
<td>White, Green, Jones, Hill</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Brown, Lee, Lewis, Young</td>
</tr>
<tr>
<td>2211</td>
<td>NP(1980), NN(174)</td>
<td>Ian, Alan, Matrin, Tony, Prince</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Chris, Brian, Harry, Andrew</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Christ, Steve</td>
</tr>
<tr>
<td>1855</td>
<td>NP(1670), NN(148)</td>
<td>Central, Leeds, Manchester</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Australia, Yorkshire, Belfast</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Glasgow, Middlesbrough</td>
</tr>
</tbody>
</table>

The largest clusters; from Biemann’s thesis, page 136, Table 5.4.

Of the 282 clusters, 26 had 100 or more elements. The clusters were checked against a lexicon with part-of-speech tags, and their purity was determined to be 88.8 percent.

8.2. Dependency Parsing

After the part-of-speech tagging task is performed, the next layer of annotation usually consists of syntactic parsing, which identifies syntactic relationships among the words in the text. Among the various parsing methods that have been proposed so far, dependency parsing – which is concerned with the identification of dependency structures in the text – has recently gained attention. Figure 8.2 is an example of a dependency parse as obtained
Figure 8.1. Clustering with the Chinese Whispers algorithms. Different shades of gray indicate different classes.

for the sentence “The monthly sales have been setting records every month since March.”

An unconventional approach to dependency parsing has been proposed recently, using graph structures and minimum spanning trees (MST).
The algorithm builds on the observation that each dependency tree (Hudson 1984) of a sentence is a directed subgraph of the full graph linking all words in the sentence. It is interesting that such an approach would not work on the more widely known constituent trees because they contain nonterminals. In dependency parsing, each sentence is represented as a tree, the root of which is typically the main predicate of the sentence (or it is a dummy node-labeled root of which the main predicate is the sole child) and in which edges are used to connect each word to its dependency parent. For example, in the sentence, “John likes green apples,” the predicate is *likes*, which takes two arguments: the *liker* (John) and the *liked* (apples). Finally, because *green* modifies *apples*, it is added to the tree as a child of *apples*. The final tree looks like this:

\[
\text{[likes [John] [apples [green]]].}
\]

The McDonald parser builds a full graph from the input sentence and then associates a score with each potential directed subtree of that graph, which is equal to the sum of the scores of all edges included. The score for each edge in the original graph is the product of a weight vector \( w \) and a feature representation of the edge \( f(i, j) \). The goal of the parser is to find the tree with the highest score. Note that punctuation, including the final period of a sentence, is used in the parsing process.

Next, the Chu–Liu–Edmonds algorithm (Chu and Liu 1965; Edmonds 1967) is used to find maximum spanning trees in directed graphs. The algorithm involves the following procedure: Each node chooses the neighbor with the highest score. The result is either a spanning tree or it has cycles. The Chu–Liu–Edmonds method collapses each such cycle into a single node and recomputes the scores of each edge incident on such a cycle. It can be shown that the minimum spanning tree built on the collapsed graph is the same as the minimum spanning tree computed on the original graph; the algorithm can be run in \( O(n^2) \).

To illustrate, consider an example consistent with McDonald et al. (2005). The sentence that needs to be parsed is “John likes green apples.” The corresponding graph is shown in the leftmost graph in Figure 8.3, and each node in the graph corresponds to a word in the sentence. After the first iteration of the algorithm, no tree is found that covers all nodes; therefore, the two closest nodes are collapsed, resulting in the second graph in Figure 8.3. The process continues until the entire graph is reduced to a single node through a series of iterations.

After all nodes have been collapsed into one, the minimum spanning tree is constructed by reversing the procedure and expanding all nodes
Figure 8.3. The graphs produced by the intermediate steps of the minimum-spanning-tree algorithm.

into their constituents. The end result for this example is shown in Figure 8.4.

Using this parser, state-of-the-art results are obtained in a standard English dataset and better than state-of-the-art results for Czech, which is a free-word-order language (McDonald et al. 2005).

8.3. Prepositional-Phrase Attachment

Prepositional-phrase attachment is one of the most challenging problems in parsing. English grammar allows a prepositional phrase such as with to

Figure 8.4. Output of the minimum-spanning-tree parser.
Pierre Vinken, 61 years old, will join the board as a nonexecutive director Nov. 29. Mr. Vinken is chairman of Elsevier N.V., the Dutch publishing group. Rudolph Agnew, 55 years old and former chairman of Consolidated Gold Fields PLC, was named a nonexecutive director of this British industrial conglomerate.

A form of asbestos once used to make Kent cigarette filters has caused a high percentage of cancer deaths among a group of workers exposed to it more than 30 years ago, researchers reported. The asbestos fiber, crocidolite, is unusually resilient once it enters the lungs, with even brief exposures to it causing symptoms that show up decades later, researchers said. Lorillard Inc., the unit of New York-based Loews Corp. that makes Kent cigarettes, stopped using crocidolite in its Micronite cigarette filters in 1956. Although preliminary findings were reported more than a year ago, the latest results appear in today’s New England Journal of Medicine, a forum likely to bring new attention to the problem.

Figure 8.5. The first two paragraphs from the Penn Treebank.

attach to either the main predicate of a sentence or the noun phrase immediately preceding it. For example, “I ate pizza with olives” is an example of low (nominal) attachment, whereas “I ate pizza with a knife” and “I ate pizza with John” are examples of high (verbal) attachment. In naturally occurring English text, both types of attachment are common. Thus, the prepositional-phrase-attachment problem consists of automatically identifying the phrase to which the prepositional phrase should be linked to inside the parse tree.

The text in Figure 8.5 illustrates the extent of the prepositional-phrase-attachment problem. In two paragraphs of fewer than two hundred words, there are more than ten prepositions whose attachments need to be solved.

One way to address the prepositional-phrase-attachment problem consists of casting it as a semi-supervised learning process on graphs (Toutanova et al. 2004). In this formulation, each node of the graph corresponds to a verb or a noun (see Figure 8.6). Two nodes are connected if they
Figure 8.6. Labeled data for prepositional-phrase attachment. By convention, x02, x01, x0, and x11 are used to indicate the position of the word in the quadruple representing the ambiguous case. V stands for high (verb) attachment and N stands for low (noun) attachment. Note that all feature words are stemmed.

appear in the same context – for example, the verb nodes *hang* and *fasten* are connected because they both appear in the expression *with a nail*. In a similar way, the noun nodes *nail* and *rivet* are connected to one another. Many other types of connections (i.e., more than ten types, including links among words with the same root form, synonyms, and others) also can be added. The algorithm then proceeds with a random walk on the graph until convergence. The evaluation is performed on a standard test set consisting of 3,097 (verb, noun, preposition, and noun) qudruples extracted from the Penn Treebank (see Figure 8.7). The results reported in the paper show a performance of 87.54 percent classification accuracy – which is close to the upper bound corresponding to human performance (i.e., 88.20 percent) – as measured using the agreement of two human annotators performing the prepositional-phrase-attachment task by hand.

8.4. Co-Reference Resolution

In addition to part-of-speech tagging and syntactic parsing, co-reference resolution is another annotation task often used in applications. It is defined

Figure 8.7. Test data for prepositional-phrase attachment.
as the problem of identifying relationships between entity references in a text, whether they are represented by nouns or pronouns. Typical algorithms for co-reference resolution attempt to identify chains of references by using rule-based systems or machine-learning classifiers.

In recent work, graph-based approaches to co-reference resolution were proposed (Nicolae and Nicolae 2006; Ng 2009) that attempt to approximate the correct assignment of references to entities in a text by using a graph-cut algorithm.

In the method presented by Nicolae and Nicolae (2006), a separate graph is created for each entity type as defined in the evaluations run by the National Institute of Standards and Technology, including Person, Organization, Location, Facility, and Geographical/Social/Political Entities. Next, weighted edges are drawn between the entity references, in which the weights correspond to the confidence of a co-reference relationship. Finally, a partitioning method based on min-cut is applied to these graphs, which separates the references corresponding to the same entity.

The following example is used in Nicolae and Nicolae (2006) to illustrate the algorithm. Assume that the co-reference links must be resolved in the following two sentences: Mary(1) has a brother(2), John(3). The boy(4) is older than the girl(5). The entities of interest each are labeled individually. The initial graph produced by the algorithm is shown in Figure 8.8. Several cuts are possible on this graph; eventually, however, the BESTCUT algorithm proposed in Nicolae and Nicolae (2006) will select the correct one, as shown in Figure 8.9.
Figure 8.9. Optimal cut on a graph that correctly solves the co-reference (Nicolae and Nicolae 2006).

When evaluated on standard benchmarks for co-reference resolution, the graph-based algorithm was found to lead to state-of-the-art performance, improving considerably over previous algorithms.

Further Reading

Methods based on graphs for syntactic parsing have been addressed further in Nivre and McDonald (2008) and Zhang and Clark (2008); compression of dependencies graphs has been shown to be useful for sentence fusion in Filippova and Strube (2008); and graph walks on parsed text were used for word similarity in Minkov and Cohen (2008).
Applications

This chapter addresses graph-theoretical methods for text-processing applications. The discussion includes topic identification, text summarization using graph-centrality methods; keyword extraction using random-walk language models; text segmentation using normalized-cut criteria for graph partitioning; graph structures to encode discourse relationships; word graphs for decoding in machine translation and speech processing; random-walk algorithms for translation selection in cross-language information retrieval; and graph representations and patterns on graphs for information extraction and question answering.

9.1. Summarization

Automatic summarization has received attention from the natural language processing community ever since the early approaches to automatic abstraction that laid the foundations of the current text-summarization techniques (Luhn 1958; Edmunson 1969). The literature typically distinguishes between extraction, which is concerned with identification of the information important in the input text, and abstraction, which involves a generation step to add fluency to a previously compressed text.

Most efforts to date have focused on the extraction step, which is perhaps the most critical component in a successful summarization algorithm. Among these efforts, some of the most promising approaches are based on graph representations of the text, which enable the application of graph-theoretical algorithms to identify the most salient elements in the text.

One of the first summarization techniques based on graphs is a method that creates graph representations for encyclopedic articles, in which nodes correspond to paragraphs and edges connect lexically similar paragraphs (Salton et al. 1994, 1997). In this method, a summary is produced by starting at the first paragraph of a document and following paths defined by different algorithms that cover as much of the graph contents as possible.
The idea of graph-based summarization only recently was taken further by introducing the concept of **lexical centrality** (Erkan and Radev 2004a and 2004b; Mihalcea and Tarau 2004; Wolf and Gibson 2004a). Lexical centrality is a measure of importance of the nodes in a graph formed by linking lexically related sentences or documents. A random walk then is executed on the graph and the nodes visited most frequently are selected as the summary of the input graph. Additionally, to avoid nodes with duplicate or near duplicate content, the final decision about including a node in the summary also can rely on its maximal marginal relevance, as defined in Carbonell and Goldstein (1998).

To illustrate, Figure 9.1 is an example drawn from Erkan and Radev (2004b): The input consists of eleven sentences from several news stories on related topics, with the matrix of cosine similarities for all sentence pairs shown in Figure 9.2. It is interesting that the cosine matrix potentially could be expanded into an infinite number of graphs for different values of a cosine cutoff. This is illustrated in the next two figures, which show two graphs obtained for two different threshold values. For example, if the threshold is lowered too much, the graph is almost fully connected (Figure 9.3). Conversely, raising the threshold eventually turns the graph into a set of disconnected components (Figure 9.4). The random walk is typically performed at a threshold value at which approximately half of the node pairs are connected via edges. For instance, Figure 9.5 is the weighted graph built for the text in Figure 9.1.

The random-walk summarization method can be applied to both single- and multi-document summarization because the graph can be built based on information drawn from one or multiple documents. When evaluated on standard datasets from the Document Understanding Conferences,1 the random-walk summarization method was found to be competitive with other more complex supervised systems. More important, the improvements were consistent across different datasets, covering single-document (Mihalcea and Tarau 2004) and multi-document summarization (Erkan and Radev 2004b), as well as the summarization of long documents such as books (Mihalcea and Ceylan 2007).

### 9.2. Semi-supervised Passage Retrieval

The idea of random-walk summarization can be taken further by introducing the concept of a **biased random walk** to address the task of question-focused

---

1 Available at http://duc.nist.gov/.
<table>
<thead>
<tr>
<th>SNo</th>
<th>ID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>d1s1</td>
<td>Iraqi Vice President Taha Yassin Ramadan announced today, Sunday, that Iraq refuses to back down from its decision to stop cooperating with disarmament inspectors before its demands are met.</td>
</tr>
<tr>
<td>2</td>
<td>d2s1</td>
<td>Iraqi Vice president Taha Yassin Ramadan announced today, Thursday, that Iraq rejects cooperating with the United Nations except on the issue of lifting the blockade imposed upon it since the year 1990.</td>
</tr>
<tr>
<td>3</td>
<td>d2s2</td>
<td>Ramadan told reporters in Baghdad that Iraq cannot deal positively with whoever represents the Security Council unless there was a clear stance on the issue of lifting the blockade off of it.</td>
</tr>
<tr>
<td>4</td>
<td>d2s3</td>
<td>Baghdad had decided late last October to completely cease cooperating with the inspectors of the United Nations Special Commission (UNSCOM), in charge of disarming Iraq's weapons, and whose work became very limited since the fifth of August, and announced it will not resume its cooperation with the Commission even if it were subjected to a military operation.</td>
</tr>
<tr>
<td>5</td>
<td>d3s1</td>
<td>The Russian Foreign Minister, Igor Ivanov, warned today, Wednesday, against using force against Iraq, which will destroy, according to him, seven years of difficult diplomatic work and will complicate the regional situation in the area.</td>
</tr>
<tr>
<td>6</td>
<td>d3s2</td>
<td>Ivanov contended that carrying out air strikes against Iraq, who refuses to cooperate with the United Nations inspectors, “will end the tremendous work achieved by the international group during the past seven years and will complicate the situation in the region.”</td>
</tr>
<tr>
<td>7</td>
<td>d3s3</td>
<td>Nevertheless, Ivanov stressed that Baghdad must resume working with the Special Commission in charge of disarming the Iraqi weapons of mass destruction (UNSCOM).</td>
</tr>
<tr>
<td>8</td>
<td>d4s1</td>
<td>The Special Representative of the United Nations Secretary-General in Baghdad, Prakash Shah, announced today, Wednesday, after meeting with the Iraqi Deputy Prime Minister Tariq Aziz, that Iraq refuses to back down from its decision to cut off cooperation with the disarmament inspectors.</td>
</tr>
<tr>
<td>9</td>
<td>d5s1</td>
<td>British Prime Minister Tony Blair said today, Sunday, that the crisis between the international community and Iraq “did not end” and that Britain is still “ready, prepared, and able to strike Iraq.”</td>
</tr>
<tr>
<td>10</td>
<td>d5s2</td>
<td>In a gathering with the press held at the Prime Minister’s office, Blair contended that the crisis with Iraq “will not end until Iraq has absolutely and unconditionally respected its commitments” towards the United Nations.</td>
</tr>
<tr>
<td>11</td>
<td>d5s3</td>
<td>A spokesman for Tony Blair had indicated that the British Prime Minister gave permission to British Air Force Tornado planes stationed in Kuwait to join the aerial bombardment against Iraq.</td>
</tr>
</tbody>
</table>

Figure 9.1. A cluster of 11 related sentences.
passage retrieval (Otterbacher et al. 2005). In this task, the user issues a query in the form of a natural language question and expects to obtain a set of passages from the input documents that contain the answer to that question. The biased random walk is performed on a graph that is already seeded with known positive and negative examples. Each node then is labeled in proportion to the percentage of times a random walk on the graph ends at that node. Given the presence of the initially labeled nodes, the nodes with the highest score are eventually those that are both similar to the seed nodes and central to the document set. In other words, they are chosen as the answer set by a mixture model that takes into account the known seeds (positive or negative) and the centrality score, as described in

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.00</td>
<td>0.45</td>
<td>0.02</td>
<td>0.17</td>
<td>0.03</td>
<td>0.22</td>
<td>0.03</td>
<td>0.28</td>
<td>0.06</td>
<td>0.06</td>
<td>0.00</td>
</tr>
<tr>
<td>2</td>
<td>0.45</td>
<td>1.00</td>
<td>0.16</td>
<td>0.27</td>
<td>0.03</td>
<td>0.19</td>
<td>0.03</td>
<td>0.21</td>
<td>0.03</td>
<td>0.15</td>
<td>0.00</td>
</tr>
<tr>
<td>3</td>
<td>0.02</td>
<td>0.16</td>
<td>1.00</td>
<td>0.03</td>
<td>0.00</td>
<td>0.01</td>
<td>0.03</td>
<td>0.04</td>
<td>0.00</td>
<td>0.01</td>
<td>0.00</td>
</tr>
<tr>
<td>4</td>
<td>0.17</td>
<td>0.27</td>
<td>0.03</td>
<td>1.00</td>
<td>0.01</td>
<td>0.16</td>
<td>0.28</td>
<td>0.17</td>
<td>0.00</td>
<td>0.09</td>
<td>0.01</td>
</tr>
<tr>
<td>5</td>
<td>0.03</td>
<td>0.03</td>
<td>0.00</td>
<td>0.01</td>
<td>1.00</td>
<td>0.29</td>
<td>0.05</td>
<td>0.15</td>
<td>0.20</td>
<td>0.04</td>
<td>0.18</td>
</tr>
<tr>
<td>6</td>
<td>0.22</td>
<td>0.19</td>
<td>0.01</td>
<td>0.16</td>
<td>0.29</td>
<td>1.00</td>
<td>0.05</td>
<td>0.29</td>
<td>0.04</td>
<td>0.20</td>
<td>0.03</td>
</tr>
<tr>
<td>7</td>
<td>0.03</td>
<td>0.03</td>
<td>0.03</td>
<td>0.28</td>
<td>0.05</td>
<td>0.05</td>
<td>1.00</td>
<td>0.06</td>
<td>0.00</td>
<td>0.00</td>
<td>0.01</td>
</tr>
<tr>
<td>8</td>
<td>0.28</td>
<td>0.21</td>
<td>0.04</td>
<td>0.17</td>
<td>0.15</td>
<td>0.29</td>
<td>0.06</td>
<td>1.00</td>
<td>0.25</td>
<td>0.20</td>
<td>0.17</td>
</tr>
<tr>
<td>9</td>
<td>0.06</td>
<td>0.03</td>
<td>0.00</td>
<td>0.00</td>
<td>0.20</td>
<td>0.04</td>
<td>0.00</td>
<td>0.25</td>
<td>1.00</td>
<td>0.26</td>
<td>0.38</td>
</tr>
<tr>
<td>10</td>
<td>0.06</td>
<td>0.15</td>
<td>0.01</td>
<td>0.09</td>
<td>0.04</td>
<td>0.20</td>
<td>0.00</td>
<td>0.20</td>
<td>0.26</td>
<td>1.00</td>
<td>0.12</td>
</tr>
<tr>
<td>11</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.01</td>
<td>0.18</td>
<td>0.03</td>
<td>0.01</td>
<td>0.17</td>
<td>0.38</td>
<td>0.12</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Figure 9.2. Cosine similarities across all sentence pairs in a cluster of eleven sentences.

Figure 9.3. Sample summarization graph, with high connectivity.
the previous section. The graph consists of both sentences (or paragraphs) and features (i.e., content words that appear in these sentences). The graph is bipartite because a sentence can link only to a feature and vice versa.

For instance, in the examples shown in Figures 9.6 and 9.7, the top right-hand node is initially labeled as positive (dark), whereas the bottom right-hand node is labeled as negative (clear node). During the labeling (performed using the method of relaxations), each node’s shading changes

Figure 9.4. Sample summarization graph, with low connectivity.

Figure 9.5. Weighted cosine similarity graph for the cluster in Figure 9.1.
until the process converges. At the end, darker nodes are returned as relevant to the user question.

9.3. Keyword Extraction

Related to text summarization is the task of keyword extraction, which also identifies important information in a text. However, rather than selecting entire sentences as in automatic summarization, it selects the words or phrases that best describe a document. Such keywords may constitute useful entries for building an automatic index for a document collection, can be used to classify a text, or can serve as a concise summary for a given document. For instance, research papers often are assigned key phrases by their authors as a way to facilitate browsing or searching. Similarly, books often come with listings of keywords in the form of back-of-the-book indexes, which can be used as an overview of the main concepts addressed by the book. On the Web, blog or image repositories also use key phrases to tag and organize content. Keywords also can be used for
9.3 Keyword Extraction

Figure 9.7. Biased LexRank as used for semi-supervised passage retrieval.

The problem of terminology extraction, and for the construction of domain-specific dictionaries.

The simplest possible approach is perhaps the use of a frequency criterion to select the “important” words and phrases in a document. However, this method generally was found to lead to poor results; consequently, other methods were explored. The state-of-the-art in this area is currently represented by supervised learning methods, in which a system is trained to recognize keywords in a text, based on lexical and syntactic features (Hulth 2003). This approach was first suggested by Turney (1999), in which parametrized heuristic rules were combined with a genetic algorithm into a system that automatically identified keywords in a document. A different learning algorithm was used by Frank et al. (1999), where a Naive Bayes learning scheme was applied to the document collection, with improved results observed on the same dataset as used by Turney (1999).

The best results to date in keyword extraction have been obtained with random-walk algorithms, which can lead to a performance comparable to supervised methods (Mihalcea and Tarau 2004). Because the expected end
result for this task is a set of words or phrases that are representative of a given natural language text, a graph is formed by adding sequences of one or more words as vertices in the graph. Any relationship that can be defined between two lexical units is a potentially useful connection (i.e., edge) that can be added between two such nodes. One possible relationship is co-occurrence inside the text, which also can be controlled by the distance between word occurrences: Two vertices are connected if their corresponding lexical units co-occur within a window of maximum $N$ words, where $N$ can be set to any value (typically between two and ten words).

After a graph is constructed, a random walk run on this graph leads to a ranking of the importance of the words in the text; consequently, the top-ranked words can be used as keywords. Additionally, words selected as important by the ranking algorithm and found next to one another in the text can be merged into a single phrase, thus forming multiword key phrases. For example, Figure 9.8 is a sample graph built for a short scientific text. After running the random-walk algorithm, the lexical units found to have higher scores and thus higher “importance” are numbers (1.46), inequations (1.45), linear (1.29), diophantine (1.28), upper (0.99), bounds (0.99), and strict (0.77).

It is interesting that experiments comparing this random-walk ranking with the traditional frequency scores showed that the scores assigned by the random walk can differ significantly. For the same text, a frequency approach provides the following top-ranked lexical units: systems (4), types (3), solutions (3), minimal (3), linear (2), inequations (2), and algorithms (2).

Evaluations on a dataset of scientific abstracts showed that the random-walk method can lead to precision and recall scores of up to 31.2 and 43.1 percent, respectively, which are superior to the $tf.idf$ method for keyword extraction and also are comparable to state-of-the-art supervised methods for keyword extraction.

9.4. Topic Identification

Keyword extraction assumes that the important terms describing a document can be found within the document itself. Topic identification moves beyond this assumption and also allows for the identification of important information that cannot be extracted from the given text.

In addition to the content of the document itself, other relevant information about a document such as related topics often can enable a faster and more effective search or classification. Document topics have long been
Compatibility of systems of linear constraints over the set of natural numbers. Criteria of compatibility of a system of linear diophantine equations, strict inequations, and nonstrict inequations are considered. Upper bounds for components of a minimal set of solutions and algorithms of construction of minimal generating sets of solutions for all types of systems are given. These criteria and the corresponding algorithms for constructing a minimal supporting set of solutions can be used in solving all the considered types of systems and systems of mixed types.

Keywords assigned by TextRank:
linear constraints; linear diophantine equations; natural numbers; nonstrict inequations; strict inequations; upper bounds

Keywords assigned by human annotators:
linear constraints; linear diophantine equations; minimal generating sets; non-strict inequations; set of natural numbers; strict inequations; upper bounds

Figure 9.8. Sample graph built for key-phrase extraction.

used by librarians to improve the retrieval of a document and to provide background or associated information for browsing by human users. They also can assist search, background-information-gathering and contextualization tasks, and enhanced relevancy measures.

Topic identification is the task of automatically finding topics that are relevant to an input document (Medelyan and Witten 2008). For instance, starting with a document on “United States in the Cold War,” we want to identify relevant topics such as “history,” “global conflicts,” “Soviet Union,” and so forth. The task goes beyond keyword extraction because relevant
Applications

topics may not be mentioned necessarily in the document and instead must be obtained from repositories of external knowledge. It is also different from text classification because the topics are either not known in advance or provided in the form of a controlled vocabulary with thousands of entries; thus, no classification can be performed. Instead, with topic identification, the aim is to find topics (or categories) that are relevant to the document at hand, which can be used to enrich the content of the document with relevant external knowledge.

In addition to other methods such as supervised topic learning (Medelyan and Witten 2008) and spreading activation (Syed et al. 2008), graph-based algorithms have been applied successfully to the task of topic identification using centrality algorithms (see Section 3.11) applied to a large graph built from Wikipedia (Coursey et al. 2009).

The method consists of two main steps. In the first step, a knowledge graph of encyclopedic concepts is built based on Wikipedia, in which the nodes in the graph are represented by the entities and categories defined in this encyclopedia. The edges between the nodes are represented by their relationship of proximity inside the Wikipedia articles. The graph is built once and then stored offline so that it can be used efficiently to identify topics in new documents. Using the entire English Wikipedia, the graph constructed in this way contains as many as 5.8 million nodes and 65.5 million edges. Figure 9.9 shows a small section of the knowledge graph, as built starting with the article on “Corpus Linguistics.”

In the second step, for each input document, the important encyclopedic concepts are identified using the Wikify system introduced by Mihalcea and Csomai (2007). The Wikify system automatically converts input text into a Wikipedia-like format, with important words and phrases being linked to Wikipedia articles, thus connecting the content of the input document to the external encyclopedic graph. Next, a biased-graph-centrality algorithm is run on the entire graph so that all of the nodes in the external knowledge repository are ranked based on their relevance to the input document. The algorithm is similar to the topic-sensitive random walk described in Section 5.6, with a bias set to the important concepts identified in the input document. By integrating the document and the encyclopedic graph under a common framework, the model accounts for the relationship between the nodes in the document and the encyclopedic graph as well as the relationship between the nodes in the encyclopedic graph itself.

2 As of July 2008, 2.75 million articles, 390,000 categories, and 2.65 million redirection articles.
To illustrate the topic-identification process, consider as an example the following sentence: “The United States was involved in the Cold War.” First, the text is passed through the Wikify system, which returns the articles “United States” and “Cold War.” Considering their importance score as calculated by Wikify, the selections are given an initial bias of 0.5492 (“United States”) and 0.4508 (“Cold War”).

After the first iteration, the initial activation spreads out into the encyclopedic graph, the nodes find a direct connection to one another, and their scores are changed correspondingly to 0.3786 (“United States”) and 0.3107 (“Cold War”). After the second iteration, new nodes are identified from the encyclopedic graph, a subset of which is shown in Figure 9.10. The process eventually continues for several iterations until the scores of the nodes do not change. The nodes with the highest scores in the final graph are considered the most closely related to the input sentence and thus are selected as relevant topics.

To see the effect of the initial bias, consider as an example the ranking of nodes in the encyclopedic graph when biased with the sentence “The United States was involved in the Cold War” versus the sentence “Microsoft
Table 9.1. Node-ranking differences when the encyclopedic graph is biased with different inputs: (1) “United States” and “Cold War” (US/CW) versus (2) “Microsoft” and “Computer Science” (MS/CS). The nodes are either article pages (A) or category pages (C). (Coursey et al. 2009)

<table>
<thead>
<tr>
<th>Wikipedia Entry</th>
<th>US/CW</th>
<th>MS/CS</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: United States</td>
<td>0.393636</td>
<td>0.006578</td>
<td>0.387058</td>
</tr>
<tr>
<td>C: Computer Science</td>
<td>0.000004</td>
<td>0.003576</td>
<td>-0.003571</td>
</tr>
<tr>
<td>A: World War II</td>
<td>0.007102</td>
<td>0.003674</td>
<td>0.003428</td>
</tr>
<tr>
<td>A: United Kingdom</td>
<td>0.005346</td>
<td>0.002670</td>
<td>0.002676</td>
</tr>
<tr>
<td>C: Microsoft</td>
<td>0.000001</td>
<td>0.001839</td>
<td>-0.001837</td>
</tr>
<tr>
<td>C: Cold War</td>
<td>0.001695</td>
<td>0.000006</td>
<td>0.001689</td>
</tr>
<tr>
<td>C: Living People</td>
<td>0.000835</td>
<td>0.002223</td>
<td>-0.001387</td>
</tr>
<tr>
<td>C: Mathematics</td>
<td>0.000029</td>
<td>0.001337</td>
<td>-0.001307</td>
</tr>
<tr>
<td>C: Computing</td>
<td>0.000008</td>
<td>0.001289</td>
<td>-0.001280</td>
</tr>
<tr>
<td>C: Computer Pioneers</td>
<td>0.000002</td>
<td>0.001238</td>
<td>-0.001235</td>
</tr>
</tbody>
</table>

applies Computer Science.” A comparison between the scores of the nodes when activated by each sentence is shown in Table 9.1.

To measure the effectiveness of the topic-identification process, the algorithm was evaluated in three experiments aimed at measuring the relevancy...
of the automatically identified topics with respect to manually annotated gold-standard datasets.

In the first and second experiments, the task consists of rediscovering the relevant categories for each page, which is not trivial, because there are approximately 390,000 categories from which to choose. In the first experiment, the identification of important concepts in the input text (used to bias the topic-ranking process) is performed manually by Wikipedia users. The dataset consists of 150 articles from Wikipedia, which explicitly have been removed from the encyclopedic graph. All of the articles include manual annotations of the relevant categories, as assigned by Wikipedia users. The best F-measure obtained for this dataset is 17 percent, which compares favorably with the F-measure of 11 percent obtained for a baseline system that selects by default all the categories listed in the articles linked from the original document.

In the second experiment, the identification of important concepts in the input document is automatic by the Wikify system. Using the same dataset of 150 Wikipedia articles, the F-measure picks at 11 percent, which is significantly higher than the baseline of 7 percent.

Finally, the third experiment uses a dataset introduced by Medelyan and Witten (2008), in which twenty documents in the field of computer science were annotated independently by fifteen teams of two computer-science undergraduates. The best F-measure obtained with the graph-based algorithm for this dataset is 27 percent, compared to only 14 percent obtained by the baseline system.

The experiments demonstrate that the integration of encyclopedic knowledge consistently adds useful information when compared to baselines that rely exclusively on the text at hand. The graph-based-centrality algorithm provides a means to generate a dynamic ranking of topics in Wikipedia within a framework that has the potential to utilize knowledge or heuristics through additional resources (e.g., encyclopedias) converted to graph form.

9.5. Topic Segmentation

After relevant topics have been identified in a text, an often-needed next step is to divide the text into segments. Topic segmentation is defined as the task of dividing text into coherent chunks, based on the topics being addressed in the text. For instance, considering an article that covers three different news stories, the task would be to identify automatically the stories and delineate the boundaries among them. The division of a text into topics can be accomplished at different levels of granularity, starting with the
division of a book into chapters to the identification of subtopics in a short one-page article. Many applications can benefit from topic segmentation, including text summarization, question answering, keyword extraction, and information retrieval.

Most segmentation algorithms rely on the lexical variation idea introduced by Hearst (1994), which assumes that topic changes are reflected by differences in lexical distribution. Although this is true most of the time, there are cases when the shift between topics is smooth and the lexical variations cannot be caught easily with simple linear models. This drawback can be addressed by using a graph-based representation of the text, which has the capability to account for both local and long-range changes in lexical distribution, thereby allowing for a global analysis of the text.

A graph that is meaningful for the task of text segmentation can be constructed by adding all of the sentences in a text as nodes in the graph and drawing weighted edges that reflect the pairwise sentence similarity. Sentence similarity can be calculated using cosine similarity or other metrics that have been successful in measuring text-to-text similarity (Mihalcea et al. 2006). For instance, given the text shown in Figure 9.11 and using a cosine similarity with term weights calculated using a $tf.idf$ measure, the resulting graph is shown in Figure 9.12.

To partition a text into segments, a graph algorithm that can be used is the normalized-cut criterion described in Section 2.5, which has the ability to measure both the similarity within a partition and the dissimilarity across partitions. In this way, unlike previous segmentation approaches that relied primarily on the local cohesion within a segment, the similarity relationship to other segments in the text also can be accounted for and minimized (Malioutov and Barzilay 2006).

Evaluated on datasets from different domains, including lecture notes from physics and artificial intelligence, the graph algorithm provides significantly better results than previous methods relying exclusively on local lexical cohesion (Malioutov and Barzilay 2006). Added benefits of the graph-based framework are that it can account for long-range lexical relationships between sentences and that it can integrate seamlessly under the same model both within-topic similarities and across-topic dissimilarities.

9.6. Discourse

Text segments, whether or not they belong to different topics, are usually connected by discourse relationships. Their identification in text is important for several text-processing applications, such as question answering, topic segmentation, and text summarization.
The storm was approaching from the southeast with sustained winds of 75 mph gusting to 92 mph. “There is no need for alarm,” Civil Defense Director Eugenio Cabral said in a television alert shortly after midnight Saturday. Cabral said residents of the province of Barahona should closely follow Gilbert’s movement. An estimated 100,000 people live in the province, including 70,000 in the city of Barahona, about 125 miles west of Santo Domingo. Tropical storm Gilbert formed in the eastern Caribbean and strengthened into a hurricane Saturday night. The National Hurricane Center in Miami reported its position at 2 a.m. Sunday at latitude 16.1 north, longitude 67.5 west, about 140 miles south of Ponce, Puerto Rico, and 200 miles southeast of Santo Domingo. The National Weather Service in San Juan, Puerto Rico, said Gilbert was moving westward at 15 mph with a “broad area of cloudiness and heavy weather” rotating around the center of the storm. The weather service issued a flash-flood watch for Puerto Rico and the Virgin Islands until at least 6 p.m. Sunday. Strong winds associated with Gilbert brought coastal flooding, strong southeast winds, and up to 12 feet to Puerto Rico’s south coast. There were no reports of casualties. San Juan, on the north coast, had heavy rains and gusts Saturday, but they subsided during the night. On Saturday, Hurricane Florence was downgraded to a tropical storm, and its remnants pushed inland from the U.S. Gulf Coast. Residents returned home, happy to find little damage from 90 mph winds and sheets of rain. Florence, the sixth named storm of the 1988 Atlantic storm season, was the second hurricane. The first, Debby, reached minimal hurricane strength briefly before hitting the Mexican coast last month.

Discourse relationships often are represented via conjunctions or other content words, which contribute to the coherence of the text. For instance, words such as before and afterwards indicate a temporal sequence between two statements, as in Before he ate, John drank a glass of water.” Similarly, because indicates a cause–effect relationship (e.g., “The flight was delayed because of weather conditions”); also or furthermore precede an elaboration (e.g., “New edges are added to the graph. A weight is also computed and assigned to each edge”); and but or however indicate a contrastive statement (e.g., “Mary liked Chinese food, but John hated it”).
The data structures most frequently used to describe discourse relationships are the tree structures, which are easy to derive and formalize and can encode the text segments involved in a relationship and their discourse connectors (Marcu 2000).

However, it has been argued recently that graphs are even more appropriate for discourse representation because they allow for crossed dependencies among statements (Wolf and Gibson 2004b). To illustrate, consider the text shown in Figure 9.13, which has the discourse relationships shown in Figure 9.14. In this example, there is a crossed dependency between the pair of segments 0–2 and 1–3, thus, for this particular case, a graph representation more naturally matches the relationships found in the text.

0. John had to rent a car
1. and he also had to reserve a hotel
2. because he had to travel to attend a conference in Hawaii.
3. Finding a hotel in Hawaii at this time of the year was quite difficult.

Figure 9.13. Sample discourse text.
Figure 9.14. Discourse relationships for the sample text.

Annotation experiments carried out on 135 texts revealed the fact that such crossed dependencies are a prevalent phenomenon (Wolf and Gibson 2004b). In fact, it was found that as many as 12.5 percent of the discourse relationships should be removed to avoid crossed dependencies so that a tree structure could be used. Moreover, numerous nodes were found to have multiple parents, which is another reason that the tree representations are not appropriate. The annotation experiments and the statistical analysis demonstrated that graph structures are powerful representations for discourse relationships and have the ability to encode richer sets of dependencies, as compared to the alternative tree structures.

9.7. Machine Translation

Machine translation is an important and growing area within the field of natural language processing, targeting the automatic translation of natural language texts from one language to another. Early approaches to machine translation consisted primarily of rule-based systems, which – although accurate whenever they were able to produce a translation – had very low coverage and were not transferable across domains. These drawbacks have been addressed by the more recent statistical methods, which rely on large parallel texts to learn automatically translation models that then can be applied to the translation of new text.

In general, a statistical machine translation produces a single-best translation for a given text. However, there are certain applications in which it is desirable to have access to a larger number of alternative translations, which often range from hundreds to thousands of candidate translations, thereby posing storage and processing difficulties.

A possible solution for a more compact representation of the set of candidate translations is the use of word graphs, which avoid the redundancies often found among the candidate translations (e.g., there are candidates that differ by just one word).
A word graph consists of a directed acyclic graph, which has one designated root node. The edges are labeled with words, and a path through a translation graph represents a candidate translation. Optionally, the graph also can include scores associated with the words on the edges, which can be used to calculate the score for a translation path. To illustrate, Figure 9.15 shows a sample word graph built from the French–English translation of the sentence “Il l’aime” (“He loves her”).

Although initially proposed for the single-word translation models (Ueffing et al. 2002), word graphs also were used successfully for phrase-based translations (Zens and Ney 2005). The main changes from the word-level to the phrase-level translations consist of three additional steps. The first step is adding phrases to the graph by introducing new edges between words. These edges also can be regarded as “shortcuts” that connect all of the words belonging to a phrase. The second step translates the phrases in the graph by introducing an extra edge for each possible phrase translation. The third step converts the phrase-edge labels to word-edge labels by replacing each phrase edge with a sequence of single-word edges.

Experiments run on a Chinese–English translation task showed that using word graphs resulted in a significant reduction in the word-graph error rate, from 64 percent to 23 percent even for small word-graph densities. In addition, the word-graph representations are very efficient because they reduce significantly the size of the translation search space.

9.8. Cross-Language Information Retrieval

In addition to machine translation, another important task in language processing that deals with multilingual documents is cross-lingual information
retrieval. Given the accelerated growth of the number of multilingual documents on the Web and elsewhere, the need for effective multilingual and cross-lingual text-processing techniques is becoming increasingly important. Cross-language information retrieval (also known as multilingual or translingual information retrieval) is an area in information retrieval that addresses the retrieval of information in languages other than the language of a user query. For example, given a query in English, a cross-language information retrieval system could retrieve relevant documents written in Spanish or Chinese. Several important applications can benefit from such a system, including the search of monolingual or multilingual collections in languages that a user cannot read and the retrieval of images with captions in an unfamiliar language.

The two main techniques typically used for cross-language information retrieval are (1) the automatic translation of the documents into the language of the query, and (2) the automatic translation of the query into the language of the documents. Of the two, the translation of the query is used most frequently because it is less time consuming and does not require the re-indexing of the document collection.

To produce the translation of a query, the most widely applicable approach is the use of a bilingual dictionary. Although machine translation and translation based on parallel texts also are viable techniques, they usually require the availability of large parallel corpora, which exist only for a few languages. Instead, bilingual dictionaries are available for many languages and can be used efficiently to translate the words in an input query into the language of the target documents.

However, there is a drawback associated with the dictionary-based translation. Most of the words carry more than one meaning and thus can have more than one translation in a second language. For instance, depending on its context, the English word *plant* can be translated into Italian as either *pianta* or *fabbrica*. In cross-language information retrieval, this ambiguity can have the effect of bringing up irrelevant documents if the words are not correctly translated.

To address this problem, the correct translations can be identified by enforcing that they are related semantically to one another. The simplest “brute-force” approach consists of selecting the combination of translations that co-occur more frequently in a collection of texts. However, this is not always feasible due to data sparseness as well as the numerous of potential translations that can lead to a combinatorial explosion. Instead, a graph-based representation can be used effectively to identify the correct translations by using random walks on a co-occurrence graph (Monz and Dorr 2005).
To construct the co-occurrence graph, all the candidate translations are added as nodes, linked by weighted edges reflecting the association strength as determined on a large corpus in the target language. For instance, assuming a query with four words with each word having a number of translations in the target language leads to the graph illustrated in Figure 9.16.

Given this graph, the translations that are more closely related to one another can be determined using the weighted random walk discussed in Section 5.4, which uses an iterative process to assign a score to each translation. Briefly, each node is first initialized with a value corresponding to its translation probability. For instance, each translation for a word with four translations has an initial probability of $\frac{1}{4} = 25\%$. Next, the scores are iteratively updated by adding to the score from the previous iteration the summation of scores of all of the neighboring translations, each multiplied by the weight of the corresponding edge. On convergence, this process results in a set of scores that can be used to select the correct translations.

To illustrate, consider the query “Trade Unions in Europe,” which is the example used by Monz and Dorr (2005). Assuming an English–German cross-language information retrieval system, Table 9.2 shows the candidate translations along with the scores determined during three iterations of the random-walk process. The translations selected, which have the highest assigned score, are shown in boldface.

In an evaluation of a large bilingual dataset from CLEF 2003, consisting of English queries run against a collection of 700 MB of German

---

3 Available at http://www.clef-campaign.org/.
Table 9.2. Translation scores for a query posed to an English–German cross-language information retrieval system (Monz and Dorr 2005).

<table>
<thead>
<tr>
<th>Source Language</th>
<th>Target Language</th>
<th>Iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>europe</td>
<td>europa</td>
<td>1.0000</td>
</tr>
<tr>
<td>trade</td>
<td>branche</td>
<td>0.0833</td>
</tr>
<tr>
<td>trade</td>
<td>handel</td>
<td>0.0833</td>
</tr>
<tr>
<td>trade</td>
<td>gewerbe</td>
<td>0.0833</td>
</tr>
<tr>
<td>trade</td>
<td>geschaeft</td>
<td>0.0833</td>
</tr>
<tr>
<td>trade</td>
<td>abschluss</td>
<td>0.0833</td>
</tr>
<tr>
<td>trade</td>
<td>handeln</td>
<td>0.0833</td>
</tr>
<tr>
<td>trade</td>
<td>beruf</td>
<td>0.0833</td>
</tr>
<tr>
<td>trade</td>
<td>trade</td>
<td>0.0833</td>
</tr>
<tr>
<td>trade</td>
<td>handwerk</td>
<td>0.0833</td>
</tr>
<tr>
<td>trade</td>
<td>eintauschen</td>
<td>0.0833</td>
</tr>
<tr>
<td>trade</td>
<td>handel treiben</td>
<td>0.0833</td>
</tr>
<tr>
<td>trade</td>
<td>schachern mit etw.</td>
<td>0.0833</td>
</tr>
<tr>
<td>union</td>
<td>union</td>
<td>0.2000</td>
</tr>
<tr>
<td>union</td>
<td>gewerkschaft</td>
<td>0.2000</td>
</tr>
<tr>
<td>union</td>
<td>vereinigung</td>
<td>0.2000</td>
</tr>
<tr>
<td>union</td>
<td>verbindung</td>
<td>0.2000</td>
</tr>
<tr>
<td>union</td>
<td>verein</td>
<td>0.2000</td>
</tr>
<tr>
<td>trade union</td>
<td>gewerkschaftlich</td>
<td>0.5000</td>
</tr>
<tr>
<td>trade union</td>
<td>gewerkschaft</td>
<td>0.5000</td>
</tr>
</tbody>
</table>

Texts, the random-walk model was found to improve significantly over a baseline in which query terms were translated using a bilingual dictionary. Moreover, the random-walk cross-lingual information retrieval system reached 63.5 percent of the upper-bound accuracy that could be obtained on a monolingual dataset, which is a remarkable performance.

9.9. Information Extraction

Information retrieval systems usually return entire documents that are relevant to a given query. When the information need is more specific (e.g., it requires exact numbers or people’s names), it is often more beneficial to use an information-extraction system.

Information extraction is concerned with the identification of knowledge in unstructured, raw text. An example is the identification and extraction of all the titles of books authored by García Márquez. The identification of names (e.g., persons, locations, and organizations) is also an instance of information extraction.
The most successful approaches to date are based on lexico-syntactic patterns extracted starting with a set of seeds representative of the given information request (Riloff and Jones 1999). For instance, starting with the seeds *London* and *Paris* (assuming the task is to extract all country capitals), patterns such as “*X* is a capital” and “capitals such as *X*” can be identified automatically. In turn, the patterns can be used to extract new capitals, such as *Berlin* and *Bucharest*, and the bootstrapping process continues.

An important aspect of any bootstrapping algorithm is represented by the constraints used to steer the bootstrapping process. This is important because errors among the patterns are propagated into the extracted instances, which in turn fire back and lead to more erroneous patterns.

A simple yet efficient strategy to address this aspect is the use of mutual reinforcement, which is based on the idea that good patterns lead to good instances and good instances are followed by good patterns. This reinforcement can be modeled naturally using bipartite graphs, coupled with a centrality algorithm such as the HITS algorithm described in Section 5.8 (Hassan et al. 2006). In this representation, both patterns and instances are represented as vertices in the graph, whereas edges reflect a generation link between instances and patterns (e.g., instances generated by certain patterns or patterns generated by instances). Figure 9.17 illustrates the construction of this bipartite graph.

After the graph is constructed, a centrality measure such as the HITS algorithm (see Section 5.8) is used iteratively to determine a score for each node. This score represents a confidence level associated with both patterns and seeds, which can be used as a filter to avoid the introduction of errors in the bootstrapping process. Thus, the selection of patterns and
instances at each bootstrapping step is determined based on global information iteratively drawn from the entire graph of patterns and seeds (which, in fact, could be a few iterations away) rather than simply local information reflecting the quality of the immediate seeds they generate.

Experiments run on a standard dataset from the Automatic Content Extraction evaluations – using a large unstructured corpus of almost 100,000 documents and targeting the extraction of relations and entities – showed an F-measure of 51 percent, obtained with four hundred generated patterns. This is within a reasonable margin from the performance of 59 percent that can be obtained using a supervised system that requires annotated data.

9.10. Question Answering

Sharing ground with both information retrieval and information extraction, question answering is defined as the task of automatically answering a question posed in natural language. For instance, given the question, “What is the capital of Romania?”, a question answering would sift through documents to extract automatically the correct answer, “Bucharest.” Although several question-answering systems were developed at the beginning of artificial intelligence, the task only recently has received interest because of the Text Retrieval Conferences (Voorhees and Tice 2000).

One of the most successful approaches to question answering entails the use of rule-based systems that rely on patterns that can identify the correct answer in a text (Soubbotin and Soubbotin 2001; Hovy et al. 2001). For example, assuming the question, “When was Mozart born?” and an answer that matches the pattern \(<person> was born in <number>\), then \(<number>\) will be identified as the answer.

The main drawback associated with rule-based approaches is that even if manually built patterns can lead to accurate systems, these systems are not portable across domains. When the system is moved to a new domain, a new set of patterns must be developed, which is both costly and time-consuming.

An alternative solution is to learn automatically the patterns from text. One way to do this is via graph matching, in which question-and-answer graphs learned from training data are used to build graph rules that can identify answers for new questions (Molla 2006).

In this method, starting with a collection of questions and manually annotated answers, rules are learned automatically by matching their corresponding graphs. The graphs are constructed based on the output of a
dependency parser, which captures the syntactic relationships between the words and their associated roles. For instance, Figure 9.18 shows the graph built for the question, “What song did Vivaldi compose in 1731?” The next step is to find a maximum common subgraph between the graphs of the question and the answer, which then is used to generate a pattern. Following the same example and assuming the answer, “In 1731, Vivaldi composed ‘La Notte’” the graph that is generated is shown in Figure 9.19.

From these two graphs, the common subgraph that can be found using a maximum common subgraph algorithm – such as the one proposed in Myaeng and Lopez-Lopez (1992) – is used to build a pattern for the rule-based system. A potential problem with these patterns is their limited coverage due to constraints placed by the question graph. For instance,
in the previous example, the common subgraph also contains the words *concert* and *compose*. The solution is to generalize selected vertices in the graph to variables so that the resulting rule has a higher coverage. The generalization is done for all of the vertices except those appearing on a list of "stop vertices" – a concept analogous to the stop words used in information retrieval. Following these transformations, the resulting rule is shown in Figure 9.20, where two vertices were transformed into variables and the answer is represented by thick lines.

Provided a new question – for example, “What song did Beethoven write in 1801?” – the corresponding question graph obtained from the dependency parser is extended with the graph rule, resulting in the graph shown in Figure 9.21. Then, given a candidate-answer statement, such as

**Figure 9.21. Extended graph for a new test question, “What song did Beethoven write in 1801?”**
“In 1801, Beethoven wrote the ‘Moonlight Sonata’,” the overlap computed between the question and the candidate-answer graphs suggests “Moonlight sonata” as the correct answer, as shown in Figure 9.22.

Although the overall performance of this question-answering method based on graphs does not compete with the state-of-the-art – that is, an accuracy of 21.44 percent as measured on 560 questions from the Text Retrieval Conference dataset – the method nonetheless is appealing because it is easily portable to a new domain, it does not require a separate question-classification step, and it does not use a named entity recognizer that may not be readily available for languages other than English.

9.11. Term Weighting

Term frequency has long been used as a major factor for estimating the probabilistic distribution of features in a document, and it has been employed in a broad spectrum of tasks including language modeling, feature selection, and term weighting. The main drawback associated with the term-frequency method is the fact that it relies on a bag-of-words approach. It implies feature independence and disregards any dependencies that may exist among words in the text. In other words, it defines a “random choice” in which the weight of the term is proportional to the probability of choosing the term randomly from the set of terms that constitute the text. Such an approach might be effective for capturing the relevance of a term in a local context,
London-based sugar operator Kaines Ltd confirmed it sold two cargoes of white sugar to India out of an estimated overall sales total of four or five cargoes in which other brokers participated. The sugar, for April/May and April/June shipment, was sold at between 214 and 218 dlrs a tonne cif, it said.

but it fails to account for the global effect that the term’s existence exerts on the entire text segment.

Instead, given that relationships could be preserved by maintaining the structural representation of the text, a method that considers the structural properties of the context could lead to a better term-weighting scheme. In fact, a method that models the weighting problem as a “random-walk” rather than “random-choice” was found to lead to significant improvements with respect to the term-frequency approach when applied to a text-classification task (Hassan et al. 2007). The random-walk method assumes an imaginary reader (or “walker”) who steps through the text on a term-by-term basis. In this setting, the importance of the term is determined by the probability of the random walker to encounter the target term in the text during the walk.

First, given a document, a graph is built by adding all of the terms in the text as vertices. Then a co-occurrence scanner is applied to the text to relate the terms that co-occur within a given window size. For a given term, all of the terms that fall in its vicinity are considered dependent terms, which are represented by a set of edges that connect the term to all other terms in the window. Finally, a random-walk algorithm (see Section 5.2) is applied on the graph, resulting in a set of scores associated with the terms in the documents. These scores then can be used as weights for text classification. Other applications that rely on language models also can benefit from this random-walk term-weighting method.

To illustrate, consider the example in Figure 9.23, which models a sample document. Starting with this text, a graph is constructed as follows: If a term has not been seen previously, then a node is added to the graph to represent it. A term can be represented only by one node in the graph. An undirected edge is drawn between two nodes if they co-occur within a certain window size. Figure 9.24 shows the graph constructed for this text, assuming a window size of 2 and corresponding to two consecutive terms in the text (e.g., London is linked to based).
Table 9.3. Random-walk scores for a sample text.

<table>
<thead>
<tr>
<th>Term</th>
<th>rw</th>
<th>tf</th>
<th>Term</th>
<th>rw</th>
<th>tf</th>
</tr>
</thead>
<tbody>
<tr>
<td>sugar</td>
<td>16.88</td>
<td>3</td>
<td>participated</td>
<td>3.87</td>
<td>1</td>
</tr>
<tr>
<td>sold</td>
<td>14.15</td>
<td>2</td>
<td>april</td>
<td>3.87</td>
<td>2</td>
</tr>
<tr>
<td>based</td>
<td>7.39</td>
<td>1</td>
<td>india</td>
<td>1.00</td>
<td>1</td>
</tr>
<tr>
<td>confirmed</td>
<td>6.90</td>
<td>1</td>
<td>estimated</td>
<td>1.00</td>
<td>1</td>
</tr>
<tr>
<td>Kaines</td>
<td>6.81</td>
<td>1</td>
<td>sales</td>
<td>1.00</td>
<td>1</td>
</tr>
<tr>
<td>operator</td>
<td>6.76</td>
<td>1</td>
<td>total</td>
<td>1.00</td>
<td>1</td>
</tr>
<tr>
<td>London</td>
<td>4.14</td>
<td>1</td>
<td>brokers</td>
<td>1.00</td>
<td>1</td>
</tr>
<tr>
<td>cargoes</td>
<td>4.01</td>
<td>2</td>
<td>may</td>
<td>1.00</td>
<td>1</td>
</tr>
<tr>
<td>shipment</td>
<td>4.01</td>
<td>1</td>
<td>june</td>
<td>1.00</td>
<td>1</td>
</tr>
<tr>
<td>dhrs</td>
<td>4.01</td>
<td>1</td>
<td>tonne</td>
<td>1.00</td>
<td>1</td>
</tr>
<tr>
<td>white</td>
<td>3.87</td>
<td>1</td>
<td>cif</td>
<td>1.00</td>
<td>1</td>
</tr>
</tbody>
</table>

After the graph is constructed, the random-walk model is applied to the graph, resulting in a set of scores associated with the vertices (i.e., words). Table 9.3 shows the term frequency and the random-walk weights. By analyzing the random-walk weights, one can observe a nonlinear correlation with the tf weights, with emphasis on terms surrounding important key

Figure 9.24. Sample graph representation from Hassan et al. 2007.
terms such as sugar and cargoes. This spatial locality resulted in higher ranks for terms like operator compared to other terms like London.\footnote{All of the missing words (e.g., Ltd and it) not shown in the graph are common words that were eliminated during preprocessing.}

To evaluate the random-walk–based approach to feature weighting, the model was integrated in a text-classification algorithm and its performance was evaluated on several standard text-classification datasets. Experiments showed that the random-walk model can achieve relative error-rate reductions of up to 84.3 percent, as compared to the traditional term-frequency–based approach, which demonstrates that graphs are useful representations for capturing the importance of terms in a text.

Further Reading

Many other natural-language applications that make use of graph representations and algorithms could not be covered in this chapter. For further reading, we recommend the following papers: language classification (Biemann 2006b); construction of email networks (McCallum et al. 2007); word graphs for speech recognition (Ney et al. 1997); event-centric summarization (Vanderwende et al. 2004); temporal graphs (Bramsen et al. 2006); relation extraction (Chen et al. 2006); and novelty detection (Gamon 2006).
Some of the conference proceedings below are available from the Association for Computational Linguistics, P.O. Box 6090, Somerset, NJ 08875 USA, acl@aclweb.org, http://www.aclweb.org; the Association for Computing Machinery, acmhelp@acm.org, http://www.acm.org; and the Computation and Language subject area of the Computing Research Repository e-print archive, a part of the xxx.lanl.gov e-print archive on the World Wide Web.


Bibliography


Bibliography


Bibliography


Bibliography


Soubbotin, M., and S. Soubbotin. 2001. Patterns of potential answer expressions as clues to the right answer. In Proceedings of the TREC-10 Conference, pp. 175–182, Gaithersburg, MD, USA.


Index

adjacent vertices, 12
algebraic connectivity, 112
associative network, 82
assortative mixing, 76
assortativity, 65
authorities, 101
average shortest path, 13

Barabasi–Albert model, 61
betweenness centrality, 69
betweenness-based clustering, 115
bipartite graph, 14
breadth-first search, 23
breadth-first traversal, 22

Chinese Whispers algorithm, 140
Chu–Liu–Edmonds algorithm, 143
clique, 13
closeness centrality, 68
clustering coefficient, 13, 14, 62
co-occurrence network, 78
cycle, 13
cluster, 15
co-reference resolution, 146
cross-lingual information retrieval, 166
giant component, 62

depth-first search, 23
depth-first traversal, 20
diameter, 13
digraph, 11
Dijkstra's algorithm, 27
dimensionality reduction, 32
directed graphs, 11
discourse analysis, 159
electrical network, 42
entropy, 106
Erdős number, 67
Erdős-Rényi model, 54
Fiedler algorithm, 113
Ford–Fulkerson theorem, 30
graph centrality algorithm, 67
graph clustering, 136
graph definition, 11
graph edit distance, 31
graph matching, 171
graph theory, 11
graph-cut algorithm, 147
harmonic function, 38
hierarchical clustering, 106, 120

HITS, 101, 170
hubs, 101

in-degree, 12
incidence, 12
induced links, 119
information content, 127
information extraction, 169
intercluster density, 109
index