Progress in Computer Science and Applied Logic

Horst Bunke, Peter J. Dickinson, Miro Kraetzl, and Walter D. Wallis

A Graph-Theoretic Approach to Enterprise Network Dynamics

Birkhäuser

We dedicate this book to our parents

Progress in Computer Science and Applied Logic
Volume 24

Editor

John C. Cherniavsky, National Science Foundation

Associate Editors

Robert Constable, Cornell University
Jean Gallier, University of Pennsylvania
Richard Platek, Cornell University
Richard Statman, Carnegie-Mellon University

Horst Bunke
Peter J. Dickinson
Miro Kraetzl
Walter D. Wallis

A Graph-Theoretic Approach to Enterprise Network Dynamics

Birkhäuser
Boston • Basel • Berlin

Horst Bunke
Universität Bern
Institute of Computer Science and
Applied Mathematics (IAM/FKI)
CH-3012 Bern
Switzerland
bunke@iam.unibe.ch

Peter J. Dickinson and Miro Kraetzl
Australian Department of Defence
Defence Science and Technology
Organisation (DSTO)
ISR Division–CA Group
Edinburgh SA 5111
Australia
peter.dickinson@dsto.defence.gov.au
miro.kraetzl@dsto.defence.gov.au

Walter D. Wallis
Southern Illinois University
Department of Mathematics
Carbondale, IL 62901
USA
wdwallis@math.siu.edu

Mathematics Subject Classification (2000): 68R19, 94C15, 05C90, 68T10

Library of Congress Control Number: 2006928981

ISBN-10: 0-8176-4485-7 e-ISBN-10: 0-8176-4519-5
ISBN-13: 978-0-8176-4485-7 e-ISBN-13: 978-0-8176-4519-9

Printed on acid-free paper.

©2007 Birkhäuser Boston *Birkhäuser* ®

All rights reserved. This work may not be translated or copied in whole or in part without the written permission of the publisher (Birkhäuser Boston, c/o Springer Science+Business Media LLC, 233 Spring Street, New York, NY 10013, USA), except for brief excerpts in connection with reviews or scholarly analysis. Use in connection with any form of information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed is forbidden.
The use in this publication of trade names, trademarks, service marks and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

9 8 7 6 5 4 3 2 1

www.birkhauser.com (TXQ/MP)

Preface

In this monograph, we describe the application of many graph-theoretic algorithms to a comprehensive environment of analysis of dynamic enterprise networks. Networks are ubiquitous, increasingly complex, and dynamic. Since they are part of all aspects of human life, their support of modern enterprise environments is paramount. Enterprises in general are becoming more information-based, and proper networking support depends on optimal performance management of all intranets involved in populating information and knowledge databases. Among other parameters, network dynamics analysis yields valuable information about network performance, efficiency, fault prediction, cost optimization, indicators, and warnings. After many years of applied research of generic network dynamics, we have decided to write a chronicle of our investigations to date with emphasis on enterprise networks. The motivation was two-fold: first, we wanted to convey to practitioners involved in network analysis a number of elegant applications of traditional graph-theoretic algorithms and techniques to computationally-tractable network dynamics analysis; second, we wanted to motivate researchers in other areas of mathematics, statistics, and computer science to apply similar reasoning in implementation of their approaches to analysis of dynamic enterprise networks. This monograph is also suitable for various graduate-level courses addressing state-of-the art applications of graph theory in analysis of dynamic communication networks, dynamic databasing, knowledge management, and many related applications of network dynamics.

The exposition is organized in four parts. They are relatively self-contained and describe in detail the main phases of our up-to-date investigations of enterprise network dynamics.

Part I serves as an introduction to the monograph. Chapter 1 is a basic overview of typical enterprise networks, such as intranets, and their management. An overview of enterprise intranets is given, together with the most important aspects of network monitoring and detection of anomalous network behavior. Parts of this chapter and the majority of the remaining chapters in this monograph are also based on the thesis of P. Dickinson [58]. Chapter 2 is devoted to introduction of all graph-theoretic prerequisites for the algorithms used later in the monograph.

Part II is an in-depth treatise on the use of various graph distances for event detection in modern enterprise networks. Chapter 3 covers graph matching for networks with unique node labeling; this work has also been described in [62]. Parts of reference [62] are used in this monograph with kind permission of Springer Science and Business Media. Chapter 4 introduces the most important graph similarity measures for abnormal change detection in networks, as reported in [26, 158]. Median graphs and their most important applications in detection of anomalous changes in dynamic networks are outlined in Chapter 5. Many experimental results are given as well. This work was also reported in [59, 63]. Chapter 6 addresses the important problem of clustering in the graph domain of time series of graphs. Most important types of clustering are given and applications to network analysis are outlined. Graph distances based on intra-graph clustering are covered in Chapter 7; this work has been reported also in [61]. Chapter 8 outlines possible applications of matching sequences of graphs to network dynamics investigations. Some applications to incomplete network knowledge are given. Some of this work was reported in [22].

Part III is dedicated to the exploration of properties of underlying graphs in dynamic enterprise networks. Chapter 9 introduces graph dynamics measures using path lengths and clustering coefficients. Relationships to networks of small-world type and general enterprise networks is given. In Chapter 10, a new set of measures utilizing Kendall–Wei ranking of graph tournaments is applied to network dynamics modeling and to ranking of enterprise network nodes by their importance in overall communication.

Part IV deals with theory and applications of network behavior inferencing and forecasting using sequences of graphs. Moreover, in this part, graph distances based on the hierarchical graph abstractions are introduced. Chapter 11 describes the reconstruction of missing network data using context in time and also machine learning and decision tree classifiers applied to network prediction. In this chapter, a detailed examination of the algorithms implemented is given, along with an extensive set of computational results. Some of the new results described in this chapter have been reported elsewhere [23]. In Chapter 12, network dynamics measures involving hierarchical graph abstractions are explored, together with their most important applications to enterprise network monitoring. Bounding techniques are implemented for graph contractions, resulting in favorable speedups of anomalous change computations; the main results of this chapter have been reported elsewhere [64].[1] We acknowledge the permission of World Scientific to use the material from that publication.

A monograph of this size and scope would not be possible without the help and support of many people. The first author wants to acknowledge contributions from his students at the University of Bern, especially Christophe Irniger, Michel Neuhaus, and Florian Thalmann. The second and third authors would like to thank the many people from the Intelligence, Surveillance and Reconnaissance Division of DSTO for their support during the development of this book. All four authors would also like to acknowledge the contribution of Peter Shoubridge to many theoretical and experimental aspects of our work on network dynamics investigations. Any views stated within this

[1]Also available at http://www.worldscinet.com/ijprai/18/1803/S0218001404180З.html.

book are completely our own and are not related in any way to the Australian Department of Defence.

Bern (Switzerland) *Horst Bunke*
Adelaide (Australia) *Peter J. Dickinson, Miro Kraetzl*
Carbondale (U.S.A.) *Walter D. Wallis*
March 2006

Contents

Part I

Introduction

1

Intranets and Network Management

1.1 Introduction

The origin of the Internet and TCP/IP protocol suite date back to 1969, when the Advanced Research Projects Agency (ARPA) funded a research and development project to create a packet-switched network, named the ARPANET. The aim was to demonstrate techniques to provide a heterogeneous, robust, and reliable data communications network. The ARPANET grew steadily as many research and educational institutions implemented the open protocols and connected to the network. The ARPANET has since evolved into the global network of networks that we know as the Internet and has continued to grow rapidly.

The Transmission Control Protocol/Internet Protocol (TCP/IP) is a suite of protocols that form the basic foundation of the Internet. The Internet Protocol (IP) is central to the architecture of the Internet. In terms of the OSI (Open Systems Interconnection) seven-layer reference model, it provides the data link and network layer services. The function of IP is to ensure that packets injected at any point in the network are routed to the intended destination. It is a connectionless protocol; hence it provides no guarantee that packets will be successfully delivered to the destination node. The TCP is a reliable connection-oriented transport layer protocol. Its function is to fragment a byte stream into discrete messages and then use IP to route these messages to the destination. At the destination, TCP sorts packets into the correct order and requests the sender to retransmit lost packets. A comprehensive coverage of TCP/IP and layered protocols, such as the OSI reference model, can be found in [166].

The popularity and growth of the Internet, in conjunction with the variety of applications that make use of it (e.g., World Wide Web and email), has led to the widespread usage of the TCP/IP protocol suite in networks not connected, or indirectly connected, to the Internet. These networks are called intranets and provide data communications for internal use by organizations. Within such organizations, the trend has been toward larger and more complex networks that support numerous corporate activities. Not surprisingly, this has led to an increased reliance on the intranet for daily business functions. In addition, the larger and more complex the network becomes, the greater the risk of performance degradations and faults. To maintain reliable network operations, it is important that network management processes be employed [69].

Network management comprises five key functional areas. Of these, fault management, performance management, and security management are central to maintaining a high level of service. Fault management is responsible for detecting and identifying network faults. A fault is usually indicated by a failure to operate correctly or through excessive errors. Performance management is concerned with how well the network or its parts are working. Security management ensures that only selected users have access to network resources. It encompasses functions such as user authorization and intrusion detection. In all of these areas of network management the early detection of network anomalies can greatly assist in minimizing or preventing a problem from occurring. Anomaly detection can be used to identify abnormal network behavior by statistical modeling of data collected from the network. To improve the effectiveness of network anomaly detection, more sophisticated techniques are required to collect and process raw network measurements in order to produce new data that has greater sensitivity to anomalous behavior.

Section 1.2 will extend the definition of an intranet by discussing typical network configurations and applications that they utilize. A general description of computer network management will be provided in 1.3. The underlying architecture for implementation of network management will be addressed in Section 1.4. In Section 1.5 the general description of network management will be refined to specifically address management of TCP/IP networks. A discussion of the two major protocols used, namely SNMP and RMON, will be given. Network monitoring, which is defined in Section 1.6, provides common implementations, techniques to minimize the volume of network measures collected, and new methods to synthesize improved measures from existing measures. A summary of the chapter is given in Section 1.8.

1.2 Enterprise Intranets

An intranet is a private network inside a company or organization that uses technologies developed for the public Internet, but that is for internal use only. Intranets are growing rapidly in popularity because they provide platform independence, and are less expensive to build and manage than private networks that use proprietary protocols and software. The importance of the intranet to an enterprise is increasing as enterprises become more information-based. The intranet provides powerful capabilities to an enterprise for dissemination of information (e.g., WWW and web browsers), efficient information retrieval (e.g., search engines) and interactive information exchange (e.g., email, newsgroups) [17]. These capabilities equate to reduced costs in operation and maintenance of infrastructure, and increased productivity from employees [14, 85].

Intranets can take advantage of many types of networking technologies in the same manner as that of the Internet. The technologies used for any given intranet will depend on many factors, including the number of users that need to access the network and the geographical area that the network must span. A small organization located in a single office, building, or group of buildings that are in close proximity to one another may require only a single network technology. A local area network (LAN) using Ethernet technology would suffice. On the other hand, a large organization that has offices globally may use numerous networking technologies. This global intranet may comprise of many Ethernet LANs, to service the individual offices, that are interconnected using a high-speed wide area networking (WAN) technology, such as asynchronous transfer mode (ATM) or frame relay. Network technologies are dealt with more thoroughly in [169]. The difficulty in managing networks increases with size and complexity of the network. While the management of a single LAN segment requires only basic network management tools, a large heterogeneous network requires a powerful network management system.

It is often a requirement for an organization to connect its intranet to the public Internet. Under these circumstances a firewall is deployed to keep unauthorized Internet traffic off the intranet. Figure 1.1 shows the interconnectivity between an intranet and the Internet. Likewise, if a partnership is formed between two organizations, a need may arise to connect their individual intranets to share information. While this poses less risk than connection to the Internet, firewalls are again deployed, however using less-stringent filters. When two or more enterprise intranets are connected to one another they are referred to as an extranet (see Figure 1.1). Organizations are faced with supporting a broader range of communications among a wider range of sites and at the same time reducing the cost of the communications infrastructure. When an urgent need arises to provide connectivity to remote offices, the past solutions to wide area networking, such as dedicated leased lines, have proven to be inflexible and expensive. Many of these problems have been solved following the advent of virtual private network (VPN) technology. VPNs use the open distributed infrastructure of the Internet to transfer encrypted data from the corporate intranet to remote sites. An example of this can be seen in Figure 1.1. The use of VPNs is not limited to corporate sites only. They can be used to provide secure connectivity to mobile users and to provide interconnections to extranets. VPNs provide a significant cost reduction over dedicated leased lines. The tradeoff for having greater flexibility in connections to extranets and the Internet is increased security risks. The management of security of an intranet is vital and should consider both internal and external vulnerabilities. While firewalls help to secure interconnections to external networks, including the Internet, and VPN technology provides intranets with a secure method of data transfer that utilizes the public infrastructure, additional methods are required to ensure that intranets, and the sensitive information they carry, remain protected from unauthorized access and malicious attacks. These techniques require the ability to detect network anomalies and identify network intrusion in order to identify and isolate incursions. Figure 1.1 also depicts how a VPN is used to connect to remote intranets via the Internet.

As mentioned, the main objective of an intranet is to provide a mechanism for information exchange. The applications that are most prevalent in intranets for information exchange are email, web browsers, ftp, and telnet. More recently, voice over IP (VoIP) and video conferencing have grown in popularity. All of these applications can generate a large volume of dynamic traffic on the underlying physical network. For most intranets, web traffic makes up the bulk of this traffic and arises from communications between distributed web servers and web browsers on user workstations. Unlike the

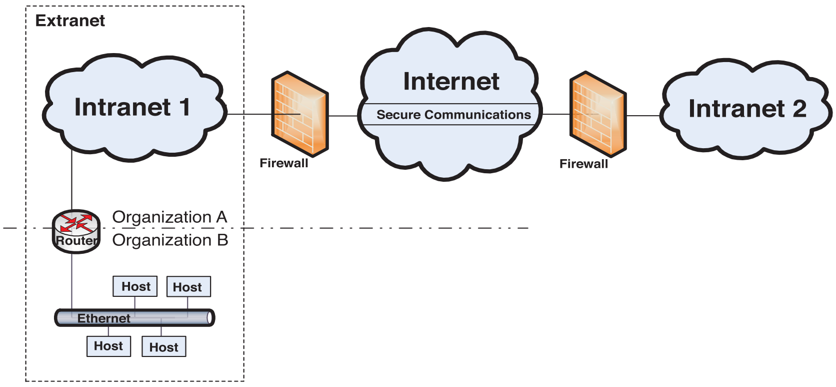


Fig. 1.1. Intranets, extranets, and the Internet.

traditional client/server models, where traffic patterns are somewhat predictable, the new web-centric model leads to unpredictable traffic patterns. These unusual patterns result from a large number of users accessing a variety of web pages that reside on different web servers distributed across the intranet. Under some circumstances this traffic can lead to significant network problems. Flash crowds [95], whereby a recently published web site is accessed concurrently by a large number of users, is one such example that results in network congestion. The ensemble behavior of web traffic is thus largely driven by the activity of its users. Web traffic, and traffic resulting from many other TCP/IP-based applications, poses significant challenges to network managers if they are to provide acceptable performance and availability of intranet applications to end users. To maintain desired levels of service to end users it is critical to exercise effective network management of web resources, bandwidth, and traffic. Network management tools that can help to identify abnormal behavior before it leads to performance degradation or network failure are still very immature. Better techniques are required to detect network anomalies in order to identify problems early so that corrective action can be taken.

The protocols used in the TCP/IP suite to control network routing (e.g., open shortest path first (OSPF) protocol), can also result in significant dynamic network behavior. It is important for network managers to know when this behavior is normal or abnormal so that faults, such as misconfiguration of routers, can be identified early. In order to derive models of normal and abnormal behavior, it is necessary for metrics to be collected from various points in the network. The network management system is used to retrieve such information. Fortunately, intranets are generally owned by one organization and thus have the advantage of a single management entity. This makes it possible to access network devices directly to acquire management data. Conversely, when one is performing network monitoring functions in the Internet domain, ownership of certain parts of the network may not always be so clear, and access to management data may be prohibited by the owner. Where complete ownership is not guaranteed, other techniques, such as network tomography [50], are required to derive information about the network. These techniques may also prove valuable for intranets. Tools that can make use of varied sources of network metrics in order to distinguish abnormal from normal behavior, and thus identify anomalous events, are valuable to network managers.

Intranets have proven to be a very useful facilitator of communication of information within an organization and as such have become relied upon for performing the daily functions of the organization. The size and complexity of these networks, in combination with the flexible nature of applications available to end users, can lead to serious network faults. This has the potential to cause major disruption to the operations of an organization. The impact of such faults can suspend an organization's activities until the problem is rectified. This could result in a large amount of lost revenue for the organization. It is very important that an organization has a network management system in place and that the capability of that system is commensurate with the risk that the organization is willing to accept if the network were to fail. Performance, fault, and security management of a network are vital components of an overall network management solution. They minimize the risk of network failure and network intrusion, and ensure that the network is providing the desired quality of service to its end users.

1.3 Network Management

In the last section we emphasized that intranets are heavily relied upon by organizations, and as a consequence it has become mandatory that network services be maintained to ensure that business operations are not disrupted. Network management is the discipline that attempts to deliver this outcome. The goal of network management is to guarantee an agreed upon level of service to users of the network. In many enterprises a service level agreement (SLA) is established with users. In general, network management is a service that employs a variety of tools, applications, and devices to assist human network managers in monitoring and maintaining networks. A more thorough coverage of network management principles can be found in [165].

Over the past twenty years there has been a huge growth in network deployment. At the same time, networking technologies have been evolving at a rapid rate. As networking requirements within organizations have grown, it has been necessary to augment the existing networking infrastructure with new infrastructure. At the time of a network upgrade it was common for an organization to use the latest technology available. This generally provided the greatest improvement in capability as a function of cost. As a result, the enterprise networks that emerged comprised a range of different networking technologies, each requiring its own set of skilled network managers. In the early 1980s, the cost of network management required to manage these large, heterogeneous networks created a crisis for many organizations. An urgent need arose for automated network management that was integrated across diverse environments.

The International Organization for Standardization (ISO) has developed the OSI network management reference model, which comprises five functional areas. This model has become the primary means for understanding the major functions of a network management system and has gained broad acceptance by vendors of both standardized and proprietary network management tools. The five functional areas are performance management, fault management, security management, configuration management, and accounting management.

The goal of performance management is to measure and make available various aspects of network performance so that internetworking performance can be maintained at an acceptable level. Examples of performance variables are network throughput, reliability, availability, latency, user response times, and resource utilization. There are two main functions of performance management, namely monitoring and control. Monitoring is used to probe network resources to acquire data that can be analyzed by performance management tools. It is important that the chosen measurement variables capture information that is suitable for identifying performance degradations. Network monitoring is discussed at length in Section 1.6. Control is the reactive component of performance management that makes adjustments to the network in response to unacceptable levels of network performance.

Fault management is used to detect, identify, locate, and if possible repair network problems. It is probably the most widely implemented function of the OSI network management components due to the impact a fault can have on critical business operations. Faults in networks include network misconfigurations, link failure, hardware interface failure, and traffic anomalies (for example, broadcast storms [5,71,82,133]). Open standards, associated with the TCP/IP protocols, have led to networks that contain hardware and software of different vendors. This complexity coupled with the growing size of networks has made network faults increasingly difficult to detect and diagnose [193].

Security management is concerned with physically securing the network and controlling access to network resources. Network access control is required to protect sensitive information from unauthorized users and to avoid malicious network attacks. It is important that sensitive information, and access points to this information, be identified and secured. Access points that are most sensitive include end user devices, backbone networks, and web servers. In large organizations it is usual that users have different levels of access to information. Users that access the network via an extranet will have very little access to sensitive information. Internal network users would likely have access to general information; however, access may be denied to information originating from a particular department. Access to human resource files, for example, would be inappropriate for most users outside the human resources department. Access-related issues have largely been managed by maintaining logs of network access and examination of audit records. The protection of an intranet from external attacks is performed by a firewall. It monitors and controls traffic into and out of an intranet. Firewalls are commonly used at the gateway between an intranet and an extranet, or the Internet. Most security risks are common to all types of networks. With the constant evolution of intranets and TCP/IP protocols, new security risks are likely to emerge as new web sites, network servers, and services are added to the network. Some of the highest-level risks to intranets include brute-force attacks, vulnerabilities in web server software, anonymous ftp, and overriding buffers [168]. Network anomaly detection can aid in network security management by detecting and identifying abnormal network events associated with breaches of security. Denial-of-service attacks and network intrusion are examples of such breaches.

The goal of configuration management is to maintain a record of the relationship among systems and network components and the status of these components during network operation. Monitoring of the network and system configuration allows the impact on network operations, of various versions of hardware and software elements, to be tracked and managed.

In many organizations there is a need to measure network usage so that individual groups, cost centers, or projects that use the corporate intranet can be charged accordingly. These charges are usually internal accounting transactions and do not require cash transfers. The measurement of network resource utilization is also important for reasons such as discovering inefficient usage or abuse of the network by end users and planning for network growth. Collection of data on network utilization, setting of usage quotas for service-level agreements, and billing users on their usage is the objective of accounting management.

Several standards bodies including IEEE, ANSI, ISO, and IETF are involved in developing new, and enhancing existing, network management standards for computer networks [13]. This responsibility includes finding solutions to known deficiencies and shortcomings in standards already in practice. The evolution of the Simple Network Management Protocol (SNMP), including the security capability introduced in SNMPv3, is a classic example of how standards are being adapted to overcome problems [160,162].

The bulk of this monograph addresses techniques that can assist with aspects of performance, fault, and security management in large dynamic intranets. In particular, a network-wide approach for detection of network anomalies is addressed. While accounting management and configuration management are important functions of an overall network management strategy, they will not be given any further attention.

1.4 Network Management System

The underlying architecture for implementation of network management is the network management system (NMS). An NMS manages all components that are connected to a network. It is made up of management stations and network elements (see Figure 1.2). A management station is generally a centralized resource, providing a user interface to enable a network manager to observe and control the behavior of the network. Furthermore, it contains a set of applications to perform management functions such as performance monitoring, fault detection, and configuration control. Network elements are resources such as workstations, routers, and bridges. Network resources that are to be managed must include a management agent. The agent module is responsible for collecting management information from the network resource and passing it on to a management station for further analysis.

In addition to the agents and managers, an NMS contains a management information base (MIB) and network management protocol. The MIB is used to store current and historical information relevant to a network resource. Some attributes contained in the MIB are related to local configuration information, such as retransmit timers, while others are related to traffic flow data, such as the number of packets in and out of a

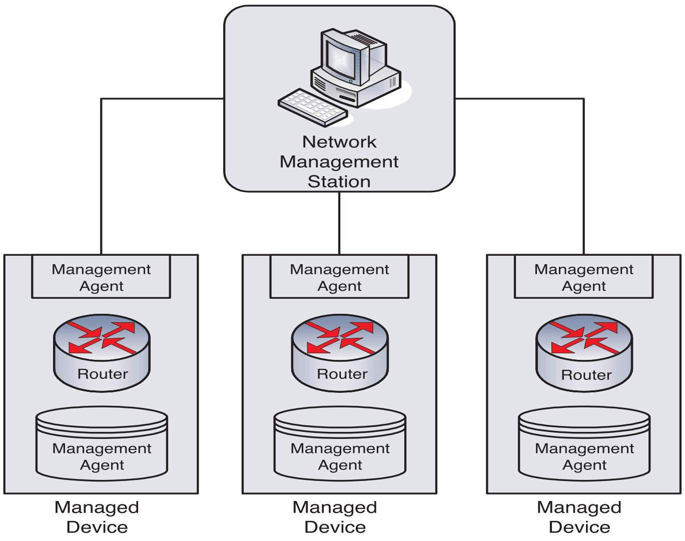


Fig. 1.2. Network management system.

device. The management agent associated with the network resource is responsible for maintaining the MIB. The management station accesses information from the MIB of managed resources to determine their status. The management station can also change the characteristics of a network element by controlling values of attributes in the MIB that relate to configuration aspects of the resource. Interaction between a manager station and management agents is provided by the network management protocol. A network management protocol defines functions for retrieving management information from agents and for issuing commands to agents. The SNMP is widely used in TCP/IP networks and hence is commonly used to manage intranets.

It is quite straightforward for a single vendor to produce an NMS to manage a network constructed solely from its own products. In practice, it is seldom the case that an organization would choose network components from a single vendor. Hence the installation of multiple network management systems may be required by an organization [159]. The use of multivendor network management systems is a problem, not only in intranets, but in network management systems in general. Much effort has been made to develop standards for the purpose of providing a common management system and to enable interoperability among different NMSs. Some attention has been given to solving internetworking problems [42, 190]. In addition, commercial products, such as HP Openview, provide an open network management platform on top of which can be built network management applications. Openview provides common management services that can be accessed through standard application interfaces (APIs). The APIs enable third-party vendors to develop their own network management systems that conform to Openview. Thus an enterprise can deploy an integrated multivendor NMS.

The two main standards for network management are those of the Internet and OSI. The OSI protocols, developed by ISO, comprise Common Management Information

Protocols (CMIP) and Common Management Information Service Element (CMISE) [159, 192]. CMIP/CMISE are very comprehensive and address all seven layers of the OSI reference model. In addition to specifying a management protocol, they also address network management applications. OSI network management is very complex, and hence implementation is difficult. The Internet protocols developed by IETF comprise the Simple Network Management Protocol (SNMP) and Remote Network-Monitoring (RMON) standard [162]. As the name implies, these protocols are simple. Because of the relative simplicity of SNMP, compared to the OSI network management standards, the TCP/IP standards have become the accepted protocol for network management. Most vendors of networking equipment support SNMP. While SNMP was developed to manage networking resources in the Internet, it has become very popular for managing other types of networks, including telecommunication networks and intranets. A thorough discussion of SNMP and RMON will be given in Sections 1.5.1 and 1.5.2 respectively, due to their widespread use in the management of intranets.

1.5 Network Management in TCP/IP Networks

In the early years of TCP/IP development little attention was given to network management. Management problems that arose were fixed by protocol experts involved in ARPANET research, using basic tools such as Internet Control Message Protocol (ICMP). As the number of hosts and subnetworks connected to the Internet exploded, so too did the number of administrative domains responsible for the health of the network. No longer could a small group of experts solve all management problems. In addition, there was a need for remote monitoring and configuring of gateways. Thus a standard protocol for network management was required, and this resulted in the development of the Simple Gateway Monitoring Protocol (SGMP). To enhance the capability provided by SGMP, the Internet Advisory Board (IAB) recommended the development of SNMP. Currently, SNMP remains at the center of network management of TCP/IP networks. The Remote Network Monitoring (RMON) standards were later developed so that statistics of network performance from subnetworks be produced locally and then passed to the central monitoring station. The IETF is responsible for all TCP/IP standards, including those for network management. It publishes standards in a form known as Request for Comments (RFCs).

Most effort in the development of TCP/IP network management standards has focused on transport protocols for accessing management information across the network and in the development of the management information base. It was envisaged that OSI network management, namely CMIP/CMISE, would provide a longer-term solution to network management; however, the simplicity of SNMP has led to it becoming the preferred standard. It is likely that SNMP will remain the primary network management protocol for many years. Beyond this, SNMP may be replaced by a more advanced network management model. It is uncertain which protocol will replace SNMP; however, OSI remains unlikely to be adopted due to its complexity [159]. Despite this, OSI network management will continue to provide a framework for the development of new standards for network management.

The following two sections provide an overview of SNMP and RMON. For a more comprehensive description see [162].

1.5.1 Simple Network Management Protocol (SNMP)

SNMP is a set of standards used for network management of TCP/IP networks. Not only does it define a protocol for exchanging network management information between an agent and manager, but it also contains a framework for the definition of management information relating to network devices. For a device to be managed by SNMP it must be capable of running an SNMP management agent. Currently, most devices designed for use in TCP/IP networks, such as bridges and routers, meet this requirement. This makes SNMP a cost-effective method for monitoring network functions. In contrast to CMIP, SNMP does not provide a definition for network management functions and services. Instead, it provides a set of primitives from which network management applications can be built. The network management applications are performed at the management station on information retrieved from management agents. Many vendors produce proprietary network management applications that use SNMP [160].

The Management Information Base (MIB) is a database used by SNMP to define characteristics of the managed resource (e.g., server, bridge, router). Each resource to be managed is represented in the database by an object. The MIB is a structured collection of such objects. Objects within the MIB must be either a scalar or tabular quantity. A tabular variable is used where multiple object instances are defined. The total number of packets on a router interface would be represented by a scalar variable, whereas a list of interface entries would be tabular. The SNMP standards describe in considerable detail the information that must be maintained by each type of management agent. This information is rigidly specified to ensure that a management station will be compatible with network resources produced by a variety of vendors. Management agents are not required to maintain all objects defined in the MIB. Depending on the type of device, a set of objects relevant to the operations of that device will be populated. For example, a device that does not require the implementation of TCP will not have to manage MIB objects relating to TCP. Most of the information stored in the MIB of a device either represents the status of the device (e.g., operational status of an interface on a router), or is an aggregated traffic-related parameter (e.g., number of packets into an interface on a router). The management station also maintains a MIB. The contents of this database reflects the contents of MIBs of network devices that it manages, and hence the objects that these devices support.

The objects defined for the MIB of SNMP are arranged hierarchically as a tree and are clustered into groups of related areas. The tree structure facilitates this logical grouping, with each managed object occupying a leaf in the tree. MIB-II is the current standard for SNMP and is specified in RFC1213 [128]. There are eleven groups defined for MIB-II. These groups contain information relating to such areas as system, interface, TCP, and UDP (User Datagram Protocol). The system group defines objects related to system administration, such as system name, contact person, and physical locality. It is the most accessed group in MIB-II. Other groups, such as interfaces and TCP, comprise objects that control the behavior of the network resource (e.g., the maximum number of allowable TCP connections) and provide counts for certain traffic-related variables (e.g., total number of input octets received by an interface). In total there are 175 objects (or variables) specified in MIB-II [128]. An important criterion used by developers of MIB-II when selecting object groups was that each object must assist in fault or configuration management. Further criteria are given in the RFC. A *private* subtree has been included in the MIB to cater for vendor specific objects.

The SNMP protocol is used to provide communication of information contained in the MIB between the managed agent and management station, and between two manager processes. SNMP was designed to operate over the UDP for robustness. Since UDP is connectionless, no connections are maintained between a management station and its agents. The use of UDP can result in some SNMP messages not reaching their intended destination node. This problem is exacerbated when SNMP is used to manage larger networks and when regular polling is relied upon for management information. If SNMP had used TCP, then SNMP exchanges may have been lost whenever one or more links in the path between the management station and managed agent failed.

SNMP was designed to be easy to implement and to consume minimal resources. As a result, the capabilities it offers are very simple. The protocol provides four basic functions to allow a manager station to communicate with an agent. These functions include *get*, *set*, *trap*, and *traversal* (e.g., get-next-request) operations. In summary, the *get* command is used to retrieve management information from agents and hence monitor managed devices; the *set* command is used to control managed devices by changing the values of certain MIB objects. The *trap* command is used to send alarms asynchronously from an agent to a management station whenever an abnormal event is detected by the agent. In the event of an alarm, a network manager would respond by polling for management information from the agent in order to ascertain the cause of the alarm. A manager station can also send an alarm to another manager station. Finally, *traversal* operations are used to determine which objects are supported by a device on the network and to sequentially retrieve information from tabular objects, such as routing tables.

The first version of SNMP, referred to as SNMP version 1, was developed as an interim management protocol, ahead of OSI management. The specification is given in RFC1157 [32]. Since OSI management was never realized, further development of SNMP was undertaken to address a number of deficiencies with SNMPv1. This resulted in SNMPv2 [33–40]. The main change to the protocol in SNMPv2 was the introduction of two new messages. The first message, *get-bulk*, provided a bulk data transfer capability for improving retrieval speed of data from tables. The second, *inform-request*, provided improved communication of management information between management stations. SNMPv2 is currently the most widely used version of SNMP. Many security issues that were identified in SNMPv1 were unable to be fixed during the development cycle of SNMPv2. The final revision of SNMPv2 was termed community-based SNMPv2 or SNMPV2C, since it uses the concept of community name for authentication purposes. Recent work on SNMPv3 has addressed security issues in greater detail [161].

An important limitation of SNMP is the inability to derive information about traffic between two subnetworks separated by two or more routers. If the traffic between such subnetworks were to increase, then the cause of such an increase could not be identified using MIB-II alone. In these circumstances RMON can be used. RMON is discussed in Section 1.5.2 below. SNMP is primarily a capability for collecting and reporting management information about the network. Further processing of information is required to perform network anomaly detection.

1.5.2 Remote Network Monitoring (RMON) Protocol

The success of SNMP management is reflected in its widespread usage in TCP/IP-based networks along with its availability in most vendors' networking equipment. This success has resulted in growth of the number of managed network resources in computer networks. SNMPv1 provided the first capability for implementing remote monitoring of a network. This capability enabled a centralized Network Operation Center (NOC) to remotely configure network resources and to detect faults. It is common for large enterprise networks to comprise many thousands of hosts and subnetworks. Unfortunately, SNMPv1 alone could not provide adequate capability for monitoring the performance of such networks. The RMON standard is an enhancement to SNMP to provide a network management system with the ability to monitor a subnetwork as a whole rather than having to monitor individual devices connected to the subnetwork. Since the characterization of network performance is statistical in nature, it was logical that such statistics be produced locally and later transmitted to a central network management station. The process of remote network monitoring involves three steps. The first step requires access to the transmission medium so that packets flowing on the network can be viewed. This activity is performed by a network monitor (or probe) attached to the subnetwork. Network monitors can be stand-alone devices or can be embedded into existing equipment, such as routers. Network monitoring employed by RMON is a passive operation and hence does not disrupt the flow of data on the network. Figure 1.3 shows a typical network configuration employing RMON probes at each LAN segment. The second step in remote monitoring is to produce summary information from data collected from a probe. This may include error statistics, such as number of collisions, and performance statistics, such as throughput and packet size distribution. The production of statistics is performed within the network monitor. The final step requires communication of the summarized information to a remote network management station.

There are several advantages in using RMON devices for network monitoring. It is not always practical to monitor subnetworks using SNMP due to the additional traffic generated by the protocol. A single RMON device can devote all of its resources to monitor a network segment and relay summarized information, characterizing the behavior of the subnetwork, to the management station. The information can be sent upon request by a management station or as a result of an abnormal event being detected by the management agent. The SNMP protocol is still used to access information from RMON devices; however, the overall effect is a marked reduction of SNMP traffic on the network, especially in the segment where the network management station resides. In addition, RMON is able to provide greater detail about the nature of traffic on a subnetwork. This can be used to deduce information such as the host within a LAN segment that is generating the most errors. It would not be possible to do this without remote monitoring unless the network management station were connected directly to

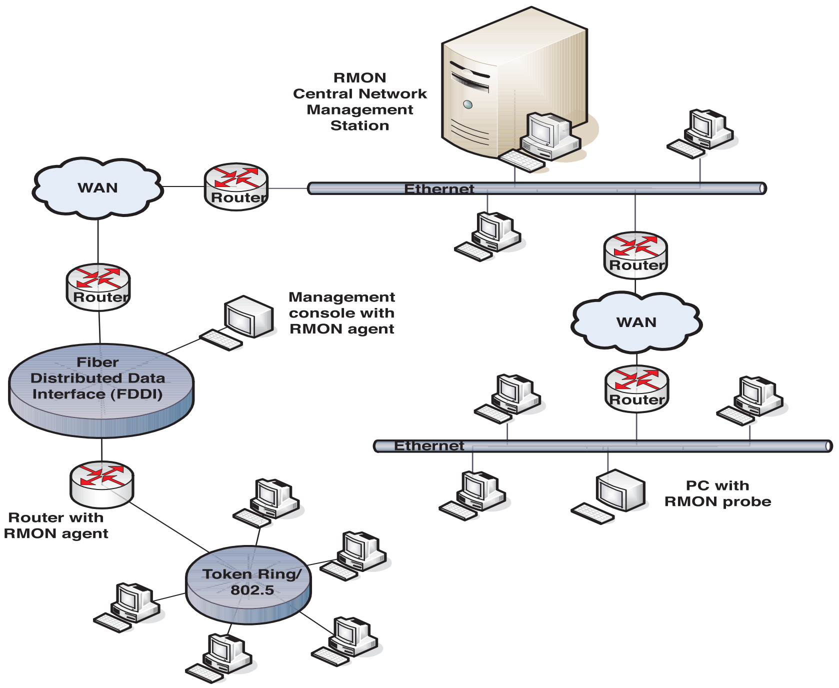


Fig. 1.3. Typical use of RMON probes for network monitoring.

the subnetwork. Since SNMP uses unreliable transport of packets, it is more likely for packet loss to occur across a large network. The local probing of a subnetwork, performed by an RMON device, is thus more reliable than that which could be achieved by regular polling of information across a large network using SNMP. If the network management station is unable to communicate with a device due to link failure, the RMON device can continue to collect statistics about the local subnetwork and report back to the network management station when connectivity resumes. It is possible to perform near-continuous monitoring of a subnetwork using RMON; hence proactive fault detection is a possibility. At the least, network problems can be identified more quickly and reported to the network management station.

In order to implement RMON, it was necessary to add new MIB variables to supplement MIB-II. No changes were required to the SNMP protocol to support RMON. A device that implements the RMON MIB is known as an RMON probe. RMON is described in detail in a number of RFCs published by the IETF. The earliest implementation of RMON, now referred to as RMON1, is given in RFC 1757 [177]. It is capable of monitoring all traffic on the LAN segment to which it is attached. RMON1 operates at the data link layer; hence it can capture MAC (medium access control) level frames and read source and destination MAC addresses in those frames. If a router is attached to the LAN, RMON1 can monitor total traffic only into and out of that router. It is not capable of determining source and destination addresses beyond the router. The

RMON1 MIB is divided into ten groups of variables. Each group provides storage of data specific to that group. An example is the statistics group used to store information on utilization of the network.

RMON2 is defined by RFC2021 and RFC2074 [11,178]. It was developed to provide a capability of monitoring protocol traffic above the MAC level. RMON2 operates upward from the network layer to the application layer. It can monitor traffic at the network layer, including IP addressing, and at the application level, such as email, ftp, and web. As a result, RMON2 can determine source or destination addresses beyond a router. This additional capability enables a network manager to determine such things as which nodes are contributing to the bulk of traffic that is incoming or outgoing to the LAN. It also enables a breakdown of traffic by protocol or application. The RMON2 MIB introduces an additional nine groups of variables to that of the RMON1 MIB. These hold information related to higher-layer activities, such as statistics of traffic carried between specific host pairs for a given application. Of most importance to the study of anomaly detection in intranets, especially the graph-theoretic techniques developed in this monograph, are the matrix groups. These groups provide statistics on the amount of traffic between pairs of hosts, and contain statistics relating to the network layer and the application layer. The network-layer matrix (nlMatrix) group provides statistics for the aggregated traffic between host pairs, while the application-layer (alMatrix) group provides statistics on the basis of application-level address. This information can be used to describe the network topology and traffic flow at the network layer for any given time interval.

The finer-grained detail of network management information provided by RMON-II comes at the cost of greater processing requirements at the management agent. This has led vendors to produce stand-alone RMON probes that are hosted on high-end servers. At present, the standards for RMON2 are being extended to support high-capacity networks.

1.6 Network Monitoring

In the last section, TCP/IP network management was discussed due to its importance in the management of intranets. Two of the most common protocols used in TCP/IP management, namely SNMP and RMON, were described in some detail. As a result of the widespread deployment of devices supporting the SNMP standards [15, 31, 90], many network anomaly detection systems are based on measurement of variables derived from such devices. In this section a general overview of network monitoring is given.

Network monitoring is an essential component of managing TCP/IP networks and is important for network anomaly detection. It is the process of gathering useful information pertaining to operations of the network. Network monitoring contributes to all of the five functional areas of the OSI network management model. In configuration management it is necessary not only to know the static configuration of a network but also to monitor its dynamically changing topology. In terms of accounting management, information is required to attribute usage of network resources to groups or divisions within an organization and to identify misappropriation of network resources by legitimate users. From a security management perspective, network monitoring is used to collect data for network intrusion detection and to identify malicious attacks, such as denial of service and web-based attacks [111]. Possibly the two most important management functions that rely on network monitoring functions are performance and fault management. Information relating to traffic statistics, network delay, and errors are required to produce indicators of network performance and to detect and identify faults. Network anomaly detection plays an important role in improving the overall capability of the OSI management functions. An important aspect of being able to detect network anomalies lies with the ability to characterize the dynamic behavior of a network. The aim is to be able to distinguish between times when the network is behaving normally and when it is behaving abnormally. To achieve this it is imperative that network behavior be quantified in some manner. Network measurement variables that are sensitive to the types of anomalies of interest to network managers must be produced from information collected from the network to provide this quantification. The underlying network monitoring system is responsible for collecting, refining, and disseminating this information to anomaly detection algorithms.

Network monitoring can be broken down into three stages. The first of these stages involves the collection of information about the network. Both active and passive monitoring techniques are employed for this function. In TCP/IP-based networks such information is commonly derived from the MIB variables of SNMP and RMON. The second stage of network monitoring transforms the collected information into useful detection metrics. The new metrics should capture information about the behavior of the network. This stage is normally referred to as the information processing phase. Finally, the third stage of network monitoring assesses the ensemble behavior of the network in order to determine abnormal events. This function is referred to as anomaly detection. Anomaly detection will be discussed separately in Section 1.7.

The major thrust of this monograph is to develop graph-based concepts that transform measurements, collected from network devices, into new measures that are more sensitive to detecting changes in network topology. This forms an important part of the second stage of network monitoring. The techniques require regular information about network-wide traffic flows in order to produce measures of network change and hence identify network anomalies.

1.6.1 Active and Passive Monitoring

Network monitoring techniques can be categorized as either active or passive. Active network monitoring techniques are generally path-oriented and include tools such as traceroute and ping [1]. Active approaches operate by injecting test traffic into the network in order to measure performance metrics such as network end-to-end delay and packet loss. These techniques are often used for characterization of the Internet, since they can be used when administrative control of the network is not centralized, and hence direct access to network elements is not possible. Conversely, passive monitoring techniques are node oriented. Passive techniques do not disrupt the flow of packets, nor add test traffic to the network. Passive monitors are used to gauge traffic flows in and out of a single device and can examine encapsulated headers to derive behavior related to the network layer and above. Devices such as routers containing SNMP agents and network monitors that implement the RMON MIB are the most commonly used passive monitors. While active monitoring techniques are useful for producing certain performance metrics, such as network latency measurements, passive techniques that collect information relating to origin–destination (OD) traffic flows are of prime interest to this monograph. The information from OD traffic flows can be represented as a graph. Graph-based techniques can then be used to produce measures that are sensitive to network change and hence be used in network anomaly detection. Such and other techniques applied to analysis of enterprise network dynamics indeed represent the main topics of this monograph.

1.6.2 Common Monitoring Solutions for Intranets

SNMP-based polling systems have proven to be cost-effective means for network monitoring due to the widespread deployment of devices that are SNMP enabled. MIB variables are a very good source of aggregated network data and are hence often used for passive network monitoring [15,31,90]. These polling systems, however, have an inherent overhead in terms of the processing load on the network devices required to compute traffic statistics, and on network bandwidth consumed when the management station retrieves data from managed agents. In some extreme cases, where a poorly designed network management system is in operation, the SNMP traffic can be responsible for disrupting the very services that it aims to maintain. In order to be capable of rapidly detecting network anomalies and faults, the rate of polling of each SNMP agent in a network has to be at least of the same order of time as that of the fault; otherwise, the fault will go undetected. This can lead to very short polling intervals on the order of a few minutes. This increased polling frequency further accentuates the processing demand on network devices and bandwidth overhead. Also, as network management systems become more focused on application-level management, the network monitoring system is required to collect more data. An increased processing load on network devices can lead to lost packets. Some research has recently been undertaken to improve the efficiency of polling [43] and data access [19] for the SNMP protocol. When a centralized measurement system is utilized, bandwidth bottlenecks can occur on links nearest to the central management station. The resource-intensive task of polling can be overcome using a distributed measurement system whereby network monitors send important data back to midlevel management stations. A midlevel station will usually oversee approximately ten probes and be located in close proximity to those probes. Its function is to consolidate data from each probe and respond to periodic queries from a higher level, or central management station [175]. In distributed polling systems, the bulk of polling is moved closer to each device, where the corresponding links are less likely to be affected by the additional traffic. The traffic load on links between midlevel management systems and the central management system is thus greatly reduced. In [112] a method to minimize bandwidth utilization using hierarchical network monitoring is addressed. To reduce the hardware requirements, and hence cost, this research sought to find the minimum number and location of midlevel stations in a network to perform polling. Other research into failure-resilient monitoring of link delays [9] and monitoring of bandwidth and latency [18] also deals with the problem of minimizing the cost of measurement infrastructure.

RMON is also commonly used for network monitoring. However, the cost of deploying an RMON solution network-wide is high. RMON was described in detail in Section 1.5.2. Since the function of RMON is to monitor and aggregate traffic in a subnetwork, it provides a similar benefit to that of a distributed management model, in that it reduces the need for regular polling by a management station. RMON is generally limited to monitoring LAN segments. Implementation of RMON for monitoring higher-speed backbone interfaces has proven to be infeasible or prohibitively expensive [77].

Packet monitors are an alternative method for the production of network measurements and are commonly used on high-speed backbone links. Packet monitors can provide very detailed information about traffic traversing a link. They operate by collecting a copy of each packet that traverses a link, recording IP, TCP/IP, or application layer information. In monitoring high-speed links the collection of every packet becomes impractical. To reduce the load on processing elements and volume of data collected, these monitors often collect only a limited number of bytes from each packet. Typically, only the IP header is collected, which contains information such as source and destination addresses and port numbers. Many packet monitoring tools have been developed by research institutions [69, 72, 103] and commercial vendors.

Many commercial tools are available for performing network monitoring functions. These range from personal computers fitted with a network interface card and special monitoring software to custom hardware devices. Examples of popular commercial monitors include NetFlow by Cisco [48], and Ecoscope by Compuware.

1.6.3 Alternative Methods for Network Monitoring

In the management of intranets we assume that a single administrative domain exists. From a network monitoring perspective this means that monitoring techniques that require direct connection to the network to perform active or passive monitoring, or require access to information on network devices, can be employed. Conversely, monitoring of Internet performance is a more difficult problem due to the size, heterogeneity, and lack of centralized administration [49, 86]. Before entering into a peering relationship,[1] the owner of an autonomous system (AS) would generally like to gain some insight into the operations of the AS that it wishes to peer with. Under these conditions it is necessary for monitoring techniques to be able to acquire information about the AS of interest without cooperation from devices within that network. Techniques exist whereby passive monitoring of traffic emanating from that network is used to derive information about the network's internal operations. Network tomography [50], which is based on signal processing, is one such technique that has been adapted for this purpose. Here network inferencing is used to estimate network performance parameters based on traffic measurements taken at a small subset of network nodes. Many of

[1]A bilateral agreement established between two or more ISPs for the purpose of directly exchanging Internet traffic.

the techniques developed specifically for Internet monitoring [44] can also be applied to intranets. While it may seem unnecessary to do so, in some circumstances where limited monitoring infrastructure is available, and network bandwidth is at a premium, such techniques can be invaluable. These methods provide additional sources of data for network anomaly detection.

1.6.4 Sampling Interval and Polling Rate

An important parameter in network monitoring is the period of time between measurements and the sampling (or aggregation) interval. The period of time between measurements, or *polling rate*, defines how often a network measurement is taken. When a fast polling rate is used, the interval between network measurements is short. Likewise, a slow polling rate results in a long interval of time between measurements. The sampling interval determines the length of time that information collected from the network is aggregated to produce any given network measurement. This interval governs the types of network faults or anomalies that can be detected. An example of this is a count of the number of packets into or out of a router interface. A sampling interval of fifteen minutes will enable shorter-duration faults to be detected than those that would be detected by a longer time interval. Aggregation of statistics over a longer time interval would mask the occurrence of the shorter-duration faults or anomalies, but would be better suited for predicting long-term network trends. The performance of a network is also largely influenced by the time of day, or operating period [194]. This is mostly due to traffic resulting from network users.

It is common for a fixed sampling interval to be employed in network monitoring implementations. However, the use of a variable-length sampling interval can be better suited to certain monitoring functions. Instead of time being used to determine the boundaries of a sampling interval, the interval could be determined by using a fixed number of OD traffic flows on a link. Thus the time interval would vary depending on the rate of traffic on the link. In busy periods, when traffic on the network is heavy, the interval would be short. Conversely, when network activity is low, such as in the middle of the night, the time interval would be longer. Obviously, there exist many other ways for determining the length of sampling intervals.

Depending on the network monitoring requirement it may be advantageous to have a short sampling interval and a slow polling rate. This approach is most suited to monitoring networks that are known to be relatively static over time. The sampling interval is set to a value suitable for capturing network anomalies of interest. Aggregation of measurements over the sampling interval is usually performed locally at a router or network probe. A slow polling rate reduces network load resulting from traffic between the management station and network element. The polling rate selected should ensure that any change that may occur will be detected within an acceptable time frame.

Most of the examples given above discuss aggregation of SNMP MIB variables. In this monograph, we address the aggregation of network topology. The aim is to produce a static representation of the network topology observed during the aggregation interval (see Section 4.2). This method is central to the application of graph-based techniques developed throughout this monograph.

In designing a network-monitoring scheme for the purposes of network-anomaly detection, the selection of time-based parameters must be given careful consideration. At present, the selection of appropriate parameters suitable for detecting anomalies of interest relies on expertise of network operators. Issues relating to selection of optimal measurement intervals for purposes of network monitoring are beyond the scope of this monograph.

1.6.5 Minimizing Collection Infrastructure and Reducing Data Volume

Network monitoring requires data to be collected from numerous measurement points across a network. Since it is not practical to collect network data from all links in a network, a selection of links must be chosen. Recent research has developed techniques to determine the best links to monitor in order to provide maximal coverage of the network given a finite set of probes [9, 18, 112]. In monitoring high-speed links in a network, the amount of data that is generated at each measurement point can be quite large. The occurrence of duplicate data is also a possibility due to several probes being on the same path between OD pairs. Where the requirement is to measure the spatial flow of packets through a network, techniques such as trajectory sampling, developed by Duffield and Grossglauser [68], make use of duplicate collection of packets. In other cases, such as the ability to detect change in network topology, duplicate packets provide misleading information. In such situations the removal of duplicate conversations is an imperative and has the additional benefit of both reducing data volume and minimizing the processing demand on network management software. The techniques developed in this monograph require the removal of duplicates.

Sampling and filtering are two other techniques used to minimize the volume of data collected [198]. Sampling involves the selection of a subset of representative packets that allow accurate estimates of the properties of the unsampled traffic to be made. Most sampling techniques aim to reduce the number of packets to be processed and transmitted to a value that is within the capabilities of the network management system. In packet filtering all packets that are not of interest are removed. Only packets that conform to certain criteria (e.g., those using the TCP protocol) are retained for further analysis. Generally, the IP header information is the only portion of the packet retained, especially when OD traffic flow data is all that is required.

Aggregation of data is also very important in reducing the amount of data forwarded by network probes. Aggregation provides compact predefined views of the traffic. Aggregation is used extensively in the MIB variables of SNMP and RMON. An example is the network-layer matrix (nlMatrix) group of RMON, which provides statistics for the aggregated traffic between host pairs.

1.6.6 Synthesis of Improved Network Measures

It has already been stated that in order to perform network anomaly detection, it is critical to collect information about a network that is sensitive to the types of anomalies to be detected. This can help to improve the detection of real network anomalies while at the same time minimizing false alarms [83]. It was also stated that no single variable or metric is capable of identifying all network anomalies. It is therefore necessary to synthesize improved network measures. A common approach is to perform some form of aggregation of two or more metrics, such as those derived from the SNMP MIB. In [172], an operator matrix was used to correlate information from several measurement variables.

In [158], summaries of traffic flows between OD pairs, for a given interval of time, have been used to generate a time series of networks, or graphs. Graph difference algorithms were applied to each consecutive graphs. This resulted in a new time series of numbers whose values represent a measure of network change that occurred between consecutive time intervals. The application of techniques for anomaly detection to the new time series has shown promise in identification of abnormal network events or trends. The research that has been outlined in this monograph builds upon these graph-based techniques for synthesizing network measurement variables for anomaly detection.

It has been discussed how network monitoring is essential to effective management of modern IP networks and that many factors are important in producing improved monitoring systems. Techniques used to transform raw network measurement variables into measures suitable for network anomaly detection are of prime interest to the work explored in this monograph. In particular, graph-based approaches form the basis for this research. Techniques for improvement of other aspects of network monitoring are beyond the scope of this monograph.

1.7 Network Anomaly Detection and Network Anomalies

Anomaly detection is the process of determining when system behavior has deviated from normal behavior. The detection of abnormal events in large dynamic intranets has become increasingly important as networks grow in size and complexity. Current network management systems are, however, unable to perform early detection of network anomalies. They rely upon alarms generated by network resources for detection and processing of network failures [2]. Not surprisingly, anomaly detection research has recently gained much interest. In network management, network administrators are responsible for monitoring network assets, such as routers and switches, for anomalous traffic behavior. While many traffic anomalies are of little or no concern, some anomalies can be indicators of serious problems. Performance bottlenecks due to flash flooding (or flash crowds), network element failure, IP forwarding anomalies, and network intrusions or denial of service attacks are a few examples of problems that cause traffic anomalies. The identification of network anomalies has relied upon ad hoc methods developed over many years by skilled network operators. In enterprise network management today, the methods used by network operators for detecting network anomalies include the use of standard tools such as SNMP traps and syslog messages. These are best suited for detecting problems such as network component failures, which can be isolated to a local area within the network and hence more easily identified. Such problems generally produce only short transient anomalies, while routing protocols reconverge. Often an anomaly can be detected only by observing traffic or topology changes in the network as a whole. The standard techniques are unable to detect such anomalies. A requirement exists for automated techniques to perform early detection of network-wide traffic anomalies to enable timely and rapid correction of a problem before it can result in a failure of service [94].

1.7.1 Anomaly Detection Methods

Anomaly detection has found applications in many of the activities relating to network management including network security [56, 117], fault detection [89, 101, 116, 124], and performance monitoring [86]. Network intrusion detection (i.e., intrusion detection and denial of service attacks) is the greatest driver for this research. Proactive fault detection [87–89, 171] is also a big driver of this work. The type of methods used for implementing anomaly detection fall into two major categories. In this section a description will be given for these two main areas of research. They are the signature-based and statistical approaches.

Signature Method

Signature-based, or rule-based, approaches to anomaly detection have been used considerably in network management [110, 119, 126]. The fundamental characteristic of signature-based approaches is that they can only detect network anomalies that have been observed in the past. A signature for each anomaly is created and stored in a database. When new anomalies are identified, through other means, a new signature is created and added to the database. Techniques in this category perform well against known problems, usually with very low false alarm rates. Signature-based methods can also help to limit the source domain. Information not relevant to signatures of interest can be ignored. Any data that cannot be matched to a known pattern can also be immediately discarded. The major disadvantage is that signature approaches are unable to identify new problems as they arise. A network anomaly that is not represented in the database of signatures of known anomalies will remain undetected. In addition, this method assumes that information is available to build a database of representative signatures. In general, such a database requires substantial time to develop, and demands the attention of network experts. Signature-based techniques are used in network anomaly detection due to the large number of efficient algorithms that have been developed in this area over time. While only a limited number of anomalies can be detected by any system using this method, it is expected that detection of anomalies using such systems is reliable, and able to explicitly identify the type of anomaly that has transpired.

Several variants of signature-based methods have been explored over time. The early work in this area was based on expert systems, where rules defining the behavior of faulty systems or known network intrusions were compiled. The rule-based systems rely heavily upon expertise of network managers and do not adapt well to an evolving network environment. In the detection of network faults, alarms are generated by network resources and sent to a central management station. Alarms arising from multiple network resources must be correlated to determine the cause of a problem. There is considerable research in the domain of alarm correlation [97, 150]. Rule-based systems are used to correlate such alarms. Case-based reasoning [119] is an extension to rule-based systems whereby previous fault scenarios are used to assist in the decision-making process. Adaptive learning techniques can be employed to enable this approach to adapt to the changing network environment. Finite state machines [116] have also been used for anomaly detection in networks. Historical data is used to build a finite state machine model. The sequence of alarms, generated from devices in the network, are modeled as states of the finite state machine. A problem is flagged whenever the sequence of events leads to a state that is known to represent an anomaly. In this way the exact type of anomaly is usually determined. The problem with finite state machines is that there may be an explosion in the number of states as a function of the number and complexity of anomalies to be modeled. Finite state machines are also not highly suited for adaptation to a changing network environment.

Statistical Method

Statistical approaches to anomaly detection [7, 125, 171] offer an alternative to signature-based methods. These methods function by learning normal network behavior from network measures. When network measurement variables deviate away from normal behavior, it represents a possible anomaly. In order to build a model of normal behavior there is a need to know when the network is operating normally. In practice it is rare for this information to be known. Likewise, it is difficult to characterize abnormal behavior [52]. Instead, it is common practice that normal behavior be represented by a period devoted to learning. An assumption is made that it would be unlikely that a network anomaly would occur during a learning period because of the low frequency of occurrence of anomalies. However, if an anomaly does occur during the learning period, the problematic behavior will become part of the model for normal behavior and hence be undetectable in the future. Common techniques used to learn normal network behavior and perform anomaly detection include auto regressive processes [31,83,171], neural networks, hidden Markov models (HMM) [87], wavelets [125], Kalman filters, change point detection [197], and Bayesian networks [89].

Unlike the signature approaches, statistical methods are capable of detecting network anomalies that have not been observed in the past. In addition, as the network evolves, statistical approaches can continuously update their model of normal behavior. Thus they have no need for regular recalibration or retraining. For statistical methods to perform adequately, they require suitable indicators to be selected as input for the decision engine. Choosing the right measures is difficult, since different types of anomalies produce different symptoms. This requires a large range of network measures to be considered. It is critical that the measurement variables selected enable modeling of normal network behavior and be sensitive to network abnormalities of interest. When more than one measure has been deemed suitable, they can be combined to produce a single anomaly measure (see Section 1.6.6) so that single time series analysis can be utilized. Alternatively, techniques based on analysis of multiple time series, or multivariate analysis, such as principal component analysis, may be used.

Statistical techniques are likely to have the disadvantage of producing a larger number of false alarms and may not be able to identify the type of anomaly detected. This increases the difficulty in deciding what action is needed by network operators when anomalous activity is detected. It is important that the network management system be able to determine whether the anomaly represents abnormal activity requiring further action, or whether it represents simply a significant change in normal network behavior. It is also useful to be able to determine whether the anomaly is related to a hardware fault or software error. These tasks are not trivial. An obvious solution might be to flag all anomalies for human attention, but this is far from ideal: frequent reporting of false alarms will lead to the anomaly detection system being ignored by network operators.

A large number of researchers have focused on applying statistical anomaly detection to network intrusion. In [117], a comparative study is given for various methods of anomaly detection based on unsupervised learning and outlier detection. Some of the outlier detection schemes studied included the nearest-neighbor, Mahalanobis distance, and density-based local outliers. Techniques were applied to data obtained by packet filtering techniques, with data aggregated at the 5-tuple IP-flow level (IP address and port number for both source and destination, and protocol type). Results showed that there was no single best solution and that the most successful anomaly detection scheme was dependent upon the type of attack.

Many techniques for anomaly detection are based on modeling the statistical behavior of one or more SNMP MIB variables. This is due to the widespread deployment and standardization of SNMP. The MIB variables selected for this purpose are generally traffic-related. In [172], a statistical signal processing technique based on abrupt change detection is used for anomaly detection in IP networks. An abrupt change is any change in the parameters of a time series that occurs on the order of the sampling interval. Correlation of events between multiple MIB variables is used to discern between anomalous events and random changes relating to normal behavior. The network anomalies detected were file server failure, network access problems, runaway processes, and protocol implementation errors. This type of approach to network anomaly detection is useful for detecting device-level anomalies.

In addition to the two main areas described above, other methods used for anomaly detection include machine learning [41] and graph-based approaches [137]. A desirable network anomaly detection system would be one that employs both a signature-based approach, to detect and identify known anomalies, and a statistical approach, to detect anomalies that have previously not been identified.

1.7.2 Network-Wide Approach to Anomaly Detection

Most work in network anomaly detection usually approaches the summarization task (determining the typical behavior of a network) from a single-link, temporal analysis standpoint. Here either MIB variables, from devices like routers or OD flow information, derived from packet filtering tools, are used as network measures. In contrast to these techniques are those that take a network-wide approach to anomaly detection. The sources of raw network measures can be the same as those used for the single-link techniques. However, the difference is that network measures are collected from numerous points across a network and combined to produce new measures that capture network-wide anomalies. Network-wide measures are suited to modeling the dynamic behavior of a network and detecting topology and traffic anomalies brought about by unusual behavioral patterns of its users. It is important for network operators to be able to identify the causes of change in traffic volumes over physical links caused by changes in user behavior across the network. The work in this monograph takes this approach, employing graph-based techniques to produce measures that are sensitive to changes in network topology.

A technique to detect network-wide anomalies was studied in [113]. Here the subspace method, based on multivariate statistical process control, was applied to OD level traffic flows collected from all routers in a network. Principal component analysis (PCA) was used to decompose OD flows into their constituent eigenflows. The top eigenflows correspond to normal network behavior with the remainder of eigenflows representing abnormal behavior. The original OD flows were reconstructed as the sum of normal and abnormal components. Abnormal events were isolated by inspecting the residual traffic. Results were obtained for three OD flow data sets comprising 5-tuple data and one of the number of bytes, packets, or flows. The anomalies detected using this approach proved to be valid. However, an additional finding showed that each data set led to different anomalies being detected. This suggests that the data sets derived from number of bytes, packets, and flows produce complementary information about network behavior.

1.7.3 Examples of Network Anomalies

There are numerous network anomalies that are of great interest to network operators. Many of these anomalies arise from network device failure, performance degradations, and network security issues. Some common performance anomalies include file server failure, paging across the network, babbling nodes, broadcast storms, and transient congestion [127]. Denial of service attacks and network intrusion are examples of security-related anomalies. Some network anomalies can be detected using techniques that make use of data gathered at the link layer or below. This may require network measurements such as counts of packets into or out of router interfaces. There are, however, many instances whereby a network-wide solution may be better suited to detecting certain kinds of network anomalies. Here, monitoring techniques that gather data at the network layer and above are required. This may entail collecting IP header information at several points across a network. The resulting data sets would include OD flows. However, these could also be further refined by specification of a certain application layer protocol (e.g., OD flows relating to http traffic only). Network-wide anomalies are often the result of unusual patterns of activity caused by user behavior. Below are a number of examples of network anomalies that impact intranets, and would be best detected using a network-wide analysis approach. The examples given aim to provide motivation for research into network monitoring strategies outlined in this monograph. Accordingly, they are prime candidates of network problems where the techniques proposed in this monograph could be applied.

Logical Communities of Interest

Changes in logical communities of interest, or user groups, often occur at times when new projects are conceived or when projects reach fruition. Teams will be formed to provide the necessary skills to achieve project outcomes, or be disbanded once a project has been completed. The enterprise intranet would provide the communications fabric for team members to go about their business. These activities generate network traffic ranging from low-volume email to large-volume data transfer. Problems are likely to occur if the traffic becomes significant. The problem is compunded when the team members access the intranet from across a country or the globe. Until network operators become aware of the impact that these users have on the network, the quality of service for them and other users may be impaired. If the new user groups can be identified early, the network can be redimensioned to provide additional network capacity where it is needed.

Corporate Reorganization

During a corporate reorganization, various job functions may be shifted from one site to another without network operators being informed. An example of this is the relocation of the management function of an email distribution server. This service may be inadvertently located within a stub network where link capacity back to the core of the enterprise network may not be adequately dimensioned to cope with the rise in traffic caused by a major email server. The additional traffic on links into and out of this stub network could lead to network congestion and hence poor quality of service to users of that network. If the stub network is not being monitored, nor links adjacent to this part of the network, then traditional tools would be unlikely to identify the problem. Such a problem could remain undetected until users report a degradation in service. Tools that can detect changes in logical network topology would be able to identify this problem. Early detection of this type of change would allow the period of performance degradation to be minimized.

Flash Crowd or New Web Service

In large-enterprise intranets it is not uncommon for a new web server to be installed onto a LAN segment without network operators being informed. The web server may be required for a new corporate electronic form to replace a dated mainframe-based system. As users access this new server, there will be a rise in traffic flow into and out of links adjacent to the server. Similarly, an existing web server may have new web pages added, or a change in content that is of high interest to the user population across the intranet. This can result in a sudden rise in traffic as the web site is accessed concurrently by a large number of users. Such an anomaly is known as a flash crowd [95]. From a link layer perspective, it would be apparent that the amount of traffic was increasing on links adjacent to the web server. Network layer detection measures are able to detect this behavior as a change in network topology.

Implementation of Custom Applications and Services

When a new service, such as a command and control system in defense applications, is being developed that requires network connectivity, it is not uncommon for the developers to perform tests over the existing network infrastructure. The traffic from these systems may branch out over many physical links to achieve connectivity with all of its constituents. The system may use a nonstandard TCP port number to avoid incompatibility with existing applications. This makes traffic monitoring difficult. In many cases, developers do not take into consideration the impact that these new systems will have on the network during testing phases. Changes in logical network topology and traffic flow would enable network operators to detect these anomalies.

Distributed Denial of Service Attacks

The aim of a distributed denial of service attack (DDoS) is to bring down a target system by flooding the network with artificial traffic. There are usually two stages to the attack. The first stage involves the perpetrator of the attack infiltrating a number of computer systems connected to the network and installing the DDoS tools. In the second stage these computer systems are instructed to start generating large volumes of network traffic. Typical targets of such an attack are routers and web servers. A DDoS could be detected by looking for an abnormally large number of connections being established from multiple nodes to the target. It may also be possible to trace the network node used by the perpetrator by analyzing historical information. Changes in logical network topology and traffic flow could provide early warning of such DDoS attacks.

1.8 Summary

In this chapter, a definition of a typical enterprise intranet was given along with many of the network management functions required to adequately maintain quality of service to users. Intranets are based on technologies developed for the Internet and have recently become invaluable to modern business. Many enterprise business functions now rely upon the intranet, and network failures or intrusions can be costly. Early detection of network anomalies can reduce or eliminate possible failures. Techniques for network anomaly detection can be used to aid network management functions.

A model widely adopted for network management comprises five functional areas, including fault management, performance management, security management, configuration management, and accounting management. While a detailed explanation of each function was given, application to performance, fault, and security management of large dynamic intranets was identified to be the focus of techniques developed in this monograph. TCP/IP network management is used to manage most intranets. The SNMP protocols form the basis of TCP/IP management. The implementation of network management functions is executed by a network management system. Management stations gather information compiled at network elements (e.g., router) by a management agent and use this information to monitor and control the behavior of the entire network.

The database used in SNMP is known as the MIB. The SNMP agent is responsible for populating the MIB with useful network measures. standardization of SNMP has resulted in it being implemented by most vendors of networking equipment. RMON is another important protocol in TCP/IP management, providing the ability to monitor remote parts of a network.

Network monitoring is essential for management of all networks. It is the process of gathering useful information pertaining to operations of the network that facilitates all five functional areas of network management. Network monitoring consists of three steps: data collection, information processing, and anomaly detection. New methods are required to improve both information processing and anomaly detection. Synthesis of new network measures using graph-based techniques can be used to improve the detection of network-wide anomalies, which is the main theme of this monograph.

Network anomaly detection is a growing field of research due to the increasing size and complexity of modern networks and the lack of tools to identify abnormal network behavior. The two main approaches to network anomaly detection are the signature and statistical methods. The signature-based methods rely on a database of known signatures to detect anomalies. While they are unable to detect new types of network anomalies, they generally produce very low false alarm rates and are able to identify the actual anomaly that has occurred. The statistical approaches model normal network behavior and classify deviation away from normal behavior as abnormal. These methods are capable of detecting unknown anomalies. However, they suffer from larger numbers of false alarms and less ability to identify the type of anomaly when one occurs.

2

Graph-Theoretic Concepts

2.1 Introduction

We are going to discuss the structure of several kinds of communications networks. In every case, the network consists of a number of individuals, or *nodes*, and certain relationships between them.

The basic mathematical structure underlying this is a *graph*. A graph consists of undefined objects called *vertices* together with a binary relation called *adjacency*: given any two vertices, either they are adjacent or they are not. Vertices will usually represent nodes or collections of nodes of the network; for example, they might be individuals in an organization, or servers in an intranet. (When vertices represent single nodes, the words "node" and "vertex" are used interchangeably.) Adjacency might represent communication between two nodes, or acquaintanceship, or any other relation.

It is useful to represent a graph in a diagram. The vertices are specially identified points, and a line is drawn between each adjacent pair of vertices, and for this reason adjacent pairs are called *edges*. (The name "graph" derives from this graphic representation.) Numerical measures may be associated with the vertices or with the edges, representing costs, capacities, et cetera.

In this book we shall encounter several generalizations of graphs; for example, it is often (but not always) useful to associate directions with the edges. Numerical measures may be associated with the vertices or with the edges, representing costs, capacities, and so on. The definition of an edge implies that there can be at most one edge between two vertices, but in some representations multiple edges make sense.

Because of the importance of the graphs that underlie networks, we shall start with a formal discussion of graphs and a few standard definitions.

General discussions of graph theory include [10,181,188]. The relation to networks is discussed in [6,135,184].

2.2 Basic Ideas

A *graph* G consists of a finite set $V(G)$ of objects called *vertices* or *nodes* together with a set $E(G)$ of unordered pairs of vertices; the elements of $E(G)$ are called *edges* or *links*. The "vertex–edge" and "node–link" terminologies are used interchangeably; "vertex" and "edge" have traditionally been used in graph-theoretic discussions, while "node" and "link" have been used in applied situations. However, in discussing communications networks, there may be more than one graph associated with a network. For example, one might consider a graph in which a vertex represents an individual user, and another graph in which each vertex stands for a collection of users (those connected to one server, or those in one branch of an organization). In that case "node" and "link" refer to the members and connections in the network, while "vertex" and "edge" are used for the graph elements.

Sometimes labels are attached to the vertices of a graph. Then we can always distinguish between two vertices when their labels are different. If there are no labels, we cannot always distinguish. As an example, consider the graphs in Figure 2.1. If considered unlabeled, the three graphs are identical. If they are considered labeled, the three graphs are different. Moreover, in the first graph, the vertices labeled x and z are different, but in the third graph the two corresponding vertices are indistinguishable.

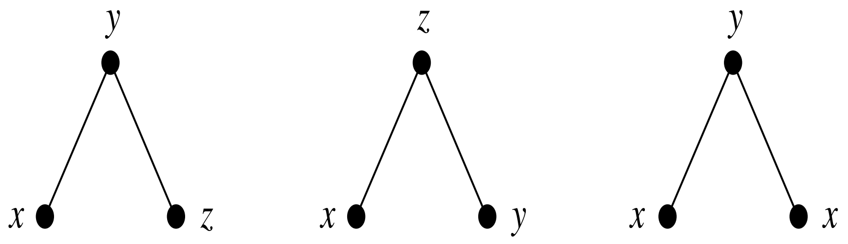


Fig. 2.1. Labeled graphs.

In terms of the more general definitions sometimes used, we can say that "our graphs are finite and contain neither loops nor multiple edges."

We write $v(G)$ and $e(G)$ for the orders of $V(G)$ and $E(G)$, respectively.

The edge containing x and y is written xy or (x, y); x and y are called its *endpoints*. We say that this edge *joins* x to y. $G - xy$ denotes the result of deleting edge xy from G; if x and y were not adjacent, then $G + xy$ is the graph constructed from G by adjoining an edge xy. Similarly $G - x$ is the graph derived from G by deleting one vertex x (and all the edges on which x lies). Similarly, $G - S$ denotes the result of deleting some set S of vertices.

In order to discuss cases in which there may be more than one link between two vertices, we define a *multigraph* in the same way as a graph except that there may be more than one edge corresponding to the same unordered pair of vertices. The *underlying graph* of a multigraph is formed by replacing all edges corresponding to the unordered pair $\{x, y\}$ by a single edge xy. Unless otherwise mentioned, all definitions pertaining to graphs will be applied to multigraphs in the obvious way.

If vertices x and y are endpoints of one edge in a graph or multigraph, then x and y are said to be *adjacent* to each other, and it is often convenient to write $x \sim y$. The set of all vertices adjacent to x is called the *neighborhood* of x, and denoted by $N(x)$. We define the *degree* or *valency* $d(x)$ of the vertex x to be the number of edges that have x as an endpoint. If $d(x) = 0$, x is an *isolated* vertex. A graph is called *regular* if all its vertices have the same degree; in particular, if the common degree is 3, the graph is called *cubic*. We write $\delta(G)$ for the smallest of all degrees of vertices of G, and $\Delta(G)$ for the largest. (One also writes $\Delta(G)$ for the common degree of a regular graph G.) If G has v vertices, so that its vertex set is, say,

$$V(G) = \{x_1, x_2, \dots, x_v\},$$

then its *adjacency matrix* M_G is the $v \times v$ matrix with entries m_{ij} such that

$$m_{ij} = \begin{cases} 1 & \text{if } x_i \sim x_j, \\ 0 & \text{otherwise.} \end{cases}$$

Some authors define the adjacency matrix of a multigraph to be the adjacency matrix of the underlying graph; others set m_{ij} equal to the number of edges joining x_i to x_j. We shall use the former convention.

A vertex and an edge are called *incident* if the vertex is an endpoint of the edge, and two edges are called incident if they have a common endpoint. A set of edges is called *independent* if no two of its members are incident, while a set of vertices is independent if no two of its members are adjacent.

Theorem 2.1. *In any graph or multigraph, the number of edges equals half the sum of the degrees of the vertices.*

Proof. It is convenient to work with the incidence matrix: we sum its entries. The sum of the entries in row i is just $d(x_i)$; the sum of the degrees is then $\sum_{i=1}^{v} d(x_i)$, which equals the sum of the entries in N. The sum of the entries in column j is 2, since each edge is incident with two vertices; the sum over all columns is thus $2e$, so that

$$\sum_{i=1}^{v} d(x_i) = 2e,$$

giving the result.

Corollary 2.2. *In any graph or multigraph, the number of vertices of odd degree is even. In particular, a regular graph of odd degree has an even number of vertices.*

Given a set S of v vertices, the graph formed by joining all pairs of members of S is called the *complete* graph on S, and denoted by K_S. We also write K_v to mean any complete graph with v vertices. The set of all edges of $K_{V(G)}$ that are *not* in a graph G will form a graph with $V(G)$ as vertex set; this new graph is called the *complement* of G, and written $\overline{G}$. More generally, if G is a subgraph of H, then the graph formed by deleting all edges of G from H is called the *complement of* G *in* H, denoted by $H - G$. The complement $\overline{K}_S$ of the complete graph K_S on the vertex set S is called a null graph; we also write $\overline{K}_v$ for a null graph with v vertices.

An *isomorphism* of a graph G onto a graph H is a one-to-one map ϕ from $V(G)$ onto $V(H)$ with the property that a and b are adjacent vertices in G if and only if $a\phi$ and $b\phi$ are adjacent vertices in H; G is isomorphic to H if and only if there is an isomorphism of G onto H. From this definition it follows that all complete graphs on n vertices are isomorphic. The notation K_n can be interpreted as being a generic name for the typical representative of the isomorphism class of all n-vertex complete graphs.

If G is a graph, it is possible to choose some of the vertices and some of the edges of G in such a way that these vertices and edges again form a graph, H say. H is then called a *subgraph* of G; one writes $H \leq G$. Clearly every graph G has itself and the 1-vertex graph (which we shall denote by K_1) as subgraphs; we say that H is a *proper* subgraph of G if it equals neither G nor K_1. If U is any set of vertices of G, then the subgraph consisting of U and all the edges of G that joined two vertices of U is called an *induced* subgraph, the *subgraph induced by* U, and is denoted by $\langle U \rangle$. A subgraph G of a graph H is called a *spanning* subgraph if $V(G) = V(H)$. Clearly any graph G is a spanning subgraph of $K_{V(G)}$.

2.3 Connectivity, Walks, and Paths

A graph is called *disconnected* if its vertex set can be partitioned into two subsets, V_1 and V_2, that have no common element in such a way that there is no edge with one endpoint in V_1 and the other in V_2; if a graph is not disconnected then it is *connected*. A disconnected graph consists of a number of disjoint subgraphs; a maximal connected subgraph is called a *component*.

Among connected graphs, some are connected so slightly that removal of a single vertex or edge will disconnect them. Such vertices and edges are quite important; in network applications, they can represent areas of great vulnerability. A vertex x is called a *cutpoint* in G if $G - x$ contains more components than G does; in particular, if G is connected then a cutpoint is a vertex x such that $G - x$ is disconnected. Similarly, a *bridge* (or *cut-edge*) is an edge whose deletion increases the number of components.

A collection of edges whose deletion disconnects G is called a *cut* in G. A cut partitions the vertex set $V(G)$ into two components, A and B say, such that the edges joining vertices in A to vertices in B are precisely the edges of the cut, and we refer to "the cut (A, B)." (The two sets A and B are not uniquely defined—for example, if there is an isolated vertex in G, it could be allocated to either set—but the cut will be well-defined.) The cut $(A, \{x\})$, consisting of all edges incident with the vertex x, is called a *trivial* cut.

The *complete bipartite graph* on V_1 and V_2 has two disjoint sets of vertices, V_1 and V_2; two vertices are adjacent if and only if they lie in different sets. We write $K_{m,n}$ to mean a complete bipartite graph with m vertices in one set and n in the other. Figure 2.2 shows $K_{4,3}$; $K_{1,n}$ in particular is called an *n-star*. Any subgraph of a complete bipartite graph is called "bipartite." More generally, the *complete r-partite graph* $K_{n_1,n_2,\dots,n_r}$ is a graph with vertex set $V_1 \cup V_2 \cup \cdots \cup V_r$, where the V_i are disjoint sets and V_i has order n_i, in which xy is an edge if and only if x and y are in different sets. Any subgraph of this graph is called an r-partite graph. If $n_1 = n_2 = \cdots = n_r = n$ we use the abbreviation $K_n^{(r)}$.

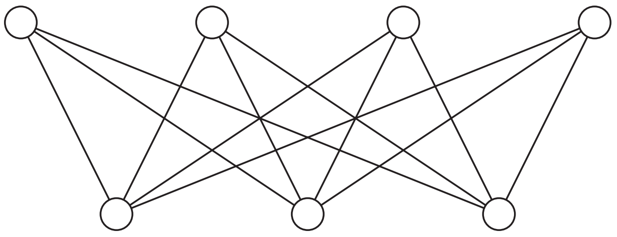


Fig. 2.2. $K_{4,3}$.

A *walk* in a graph G is a finite sequence of vertices $x_0, x_1, \dots, x_n$ and edges $a_1, a_2, \dots, a_n$ of G:

$$x_0, a_1, x_1, a_2, \dots, a_n, x_n,$$

where the endpoints of a_i are x_{i-1} and x_i for each i. A *simple walk* is a walk in which no edge is repeated. A *path* is a walk in which no vertex is repeated; the *length* of a path is its number of edges. A walk is *closed* when the first and last vertices, x_0 and x_n, are equal. A *cycle* of length n is a closed simple walk of length n, $n \geq 3$, in which the vertices $x_0, x_1, \dots, x_{n-1}$ are all different.

The following observation, although very easy to prove, will be useful.

Theorem 2.3. *If there is a walk from vertex y to vertex z in the graph G, where y is not equal to z, then there is a walk in G with first vertex y and last vertex z.*

We say that two vertices are *connected* when there is a walk joining them. (Theorem 2.3 tells us we can replace the word "walk" by "path.") Two vertices of G are connected if and only if they lie in the same component of G; G is a connected graph if and only if all pairs of its vertices are connected. If vertices x and y are connected, then their *distance* is the length of the shortest path joining them.

Cycles give the following useful characterization of bipartite graphs.

Theorem 2.4. *A graph is bipartite if and only if it contains no cycle of odd length.*

The proof is straightforward.

A graph that contains no cycles at all is called *acyclic*; a connected acyclic graph is called a *tree*. Clearly all trees are bipartite graphs. We shall discuss trees in the next section.

It is clear that the set of vertices and edges that constitute a path in a graph is itself a graph. We define a walk P_n to be a graph with n vertices $x_1, x_2, \dots, x_n$ and $n-1$ edges $x_1x_2, x_2x_3, \dots, x_{n-1}x_n$. A cycle C_n is defined similarly, except that the edge x_nx_1 is also included, and (to avoid the triviality of allowing K_2 to be defined as a cycle) n must be at least 3. The latter convention ensures that every C_n has n edges. Figure 2.3 shows P_4 and C_5.

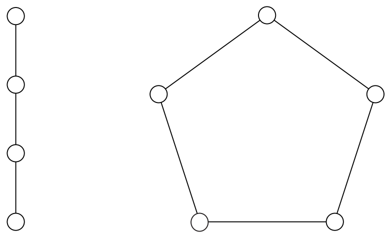


Fig. 2.3. P_4 and C_5.

As an extension of the idea of a proper subgraph, we shall define a *proper tree* to be a tree other than K_1, and similarly define a *proper path*. (No definition of a "proper cycle" is necessary.)

A cycle that passes through every vertex in a graph is called a *Hamilton cycle* and a graph with such a cycle is called *Hamiltonian*. Typically one thinks of a Hamiltonian graph as a cycle with a number of other edges (called *chords* of the cycle). The idea of such a spanning cycle was simultaneously developed by Hamilton [81] in the special case of the icosahedron, and more generally by Kirkman [105].

It is easy to discuss Hamiltonicity in particular cases, and there are a number of small theorems. However, no good necessary and sufficient conditions are known for the existence of Hamilton cycles. The following result is a useful sufficient condition.

Theorem 2.5. *If G is a graph with n vertices, $n \geq 3$, and $d(x) + d(y) \geq n$ whenever x and y are nonadjacent vertices of G, then G is Hamiltonian.*

Proof. Suppose the theorem is false. Choose an n such that there is an n-vertex counterexample, and select a graph G on n vertices that has the maximum number of edges among counterexamples. Choose two nonadjacent vertices v and w: clearly $G + vw$ is Hamiltonian, and vw must be an edge in every Hamilton cycle. By hypothesis, $d(v) + d(w) \geq n$.

Consider any Hamilton cycle in $G + vw$:

$$v, x_1, x_2, \ldots, x_{n-2}, w, v.$$

If x_i is any member of $N(v)$, then x_{i-1} cannot be a member of $N(w)$, because

$$v, x_1, x_2, \ldots, x_{i-1}, w, x_{n-2}, x_{n-3}, \ldots, x_i, v$$

would be a Hamilton cycle in G. So each of the $d(v)$ vertices adjacent to v in G must be preceded in the cycle by vertices not adjacent to w, and none of these vertices can be w itself. So there are at least $d(v) + 1$ vertices in G that are not adjacent to w. So there are at least $d(w) + d(v) + 1$ vertices in G, whence

$$d(v) + d(w) \leq n - 1,$$

a contradiction.

Corollary 2.6. *If G is a graph with n vertices, $n \geq 3$, and every vertex has degree at least $n/2$, then G is Hamiltonian.*

Theorem 2.5 was first proven by Ore [138] and Corollary 2.6 some years earlier by Dirac [65]. Both can in fact be generalized into the following result of Pósa [143]: a graph with n vertices, $n \geq 3$, has a Hamiltonian cycle provided the number of vertices of degree less than or equal to k does not exceed $k - 1$, for each k satisfying $1 \leq k \leq (n - 1)/2$.

2.4 Trees

As we stated in the preceding section, a *tree* is a connected graph that contains no cycle. Figure 2.4 contains three examples of trees. It is also clear that every path is a tree, and the star $K_{1,n}$ is a tree for every n.

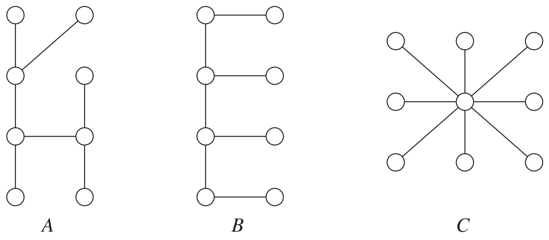


Fig. 2.4. Three trees.

A tree is a minimal connected graph in the following sense: if any vertex of degree at least 2, or any edge, is deleted, then the resulting graph is not connected.

Theorem 2.7. *A connected graph is a tree if and only if every edge is a bridge.*

Trees are also characterized among connected graphs by their number of edges.

Theorem 2.8. *A finite connected graph G with v vertices is a tree if and only if it has exactly $v - 1$ edges.*

From this it follows that every tree other than K_1 has at least two vertices of degree 1. (This does not hold if we allow our graphs to have infinite vertex sets—one elementary example consists of the infinitely many vertices $0, 1, 2, \ldots, n, \ldots$ and the edges $01, 12, 23, \ldots, (n, n + 1), \ldots$—but infinite graphs will not arise in this book.)

Theorem 2.9. *Suppose T is a tree with k edges and G is a graph with minimum degree $\delta(G) \geq k$. Then G has a subgraph isomorphic to T.*

2.5 Factors, or Spanning Subgraphs

If G is any graph, then a *factor* or *spanning subgraph* of G is a subgraph with vertex set $V(G)$. A *factorization* of G is a set of factors of G that are pairwise *edge-disjoint*—no two have a common edge—and whose union is all of G.

Every graph has a factorization, quite trivially: since G is a factor of itself, $\{G\}$ is a factorization of G. However, it is more interesting to consider factorizations in which the factors satisfy certain conditions. In particular a *one-factor* is a factor that is a regular graph of degree 1. In other words, a one-factor is a set of pairwise disjoint edges of G that between them contain every vertex. A *one-factorization* of G is a decomposition of the edge set of G into edge-disjoint one-factors. Similarly, a *two-factor* is a factor that is a regular graph of degree 2, a union of disjoint cycles, and a *two-factorization* of G is a decomposition of the edge set of G into edge-disjoint two-factors.

A *spanning tree* is a spanning subgraph that is a tree when considered as a graph in its own right. Clearly every connected graph has a spanning tree. This also applies to graphs with loops and multiple edges. Moreover, a given graph may have many different spanning trees.

In many of the applications in which each edge of a graph has a weight associated with it, it is desirable to find a spanning tree such that the weight of the tree, the total of the weights of its edges, is minimum. Such a tree is called a *minimum spanning tree*. A finite graph can contain only finitely many spanning trees, so it is possible in theory to list all spanning trees and their weights, and to find a minimum spanning tree by choosing one with minimum weight But this process can take a very long time, particularly in the large graphs arising from enterprise networks. So efficient algorithms to find a minimum spanning tree have been developed. One of the best was given by Prim [144].

There is an extensive literature on matching and factorization problems; see, for example, [120, 179, 180].

2.6 Directed Graphs

A *directed graph* or *digraph* consists of a finite set v of objects called *vertices* together with a finite set of directed edges or *arcs*, which are *ordered pairs* of vertices. It is like a graph except that each edge is allocated a direction; one vertex is designated a *start* and the other is a *finish*. An arc directed from start s to finish t is denoted by (s, t), or simply st. It is important to observe that, unlike a graph, a digraph can have two arcs with the same endpoints, provided they are directed in opposite ways. But we shall not allow *multiple arcs* or *loops*.

The idea of adjacency needs further elaboration in digraphs. Associated with a vertex v are the two sets

$$A(v) = \{x : (v, x) \text{ is an arc}\},$$
$$B(v) = \{x : (x, v) \text{ is an arc}\}.$$

A vertex v is called a *start* in the digraph if $B(x)$ is empty and a *finish* if $A(x)$ is empty. The *indegree* and *outdegree* of a vertex are the numbers of arcs leading into and leading away from that vertex respectively, so if multiple arcs are not allowed, then the indegree and outdegree of v equal $|B(v)|$ and $|A(v)|$ respectively.

Our notation is extended to directed graphs in the obvious way, so that if X and Y are any sets of vertices of G, then $[X, Y]$ consists of all arcs with start in X and finish in Y. If X or Y has only one element, it is usual to omit the set brackets in this notation. Observe that if V is the vertex set of G, then

$$\begin{aligned} [v, A(v)] = [v, V] &= \text{ set of all arcs leading out of } v, \\ [B(v), v] = [V, v] &= \text{ set of all arcs leading into } v. \end{aligned}$$

A *walk* in a directed multigraph is a sequence of arcs such that the finish of one is the start of the next. (This is analogous to the definition of a walk in a graph, but takes into account the direction of each arc. Each arc must be traversed in its proper direction.) A *directed path* is a sequence $(a_0, a_1, \ldots, a_n)$ of vertices, all different, such that $a_{i-1}a_i$ is an arc for every i. Not every path is a directed path. If a directed path is considered as a digraph, then a_0 is a start, and is unique, and a_n is the unique finish, so we call a_0 and a_n the *start* and *finish* of the path. We say a_i *precedes* a_j (and a_j *succeeds* a_i) when $i < j$.

A *directed cycle* $(a_1, a_2, \ldots, a_n)$ is a sequence of two or more vertices in which all of the members are distinct, each consecutive pair $a_{i-1}a_i$ is an arc, and also a_n, a_1 is an arc. (Notice that there can be a directed cycle of length 2, or *digon*, which is impossible in the undirected case.) A digraph is called *acyclic* if it contains no directed cycle.

As an example, consider the digraph of Figure 2.5. It has

$$\begin{aligned} A(a) &= \{b, c\}, & B(a) &= \emptyset, \\ A(b) &= \emptyset, & B(b) &= \{a, c, d\}, \\ A(c) &= \{b, d\}, & B(c) &= \{a\}, \\ A(d) &= \{b\}, & B(d) &= \{c\}; \end{aligned}$$

a is a start and b is a finish. We have $[\{a, c\}, \{b, d\}] = \{ab, cb, cd\}$. There are various directed paths, such as (a, c, d, b), but no directed cycle.

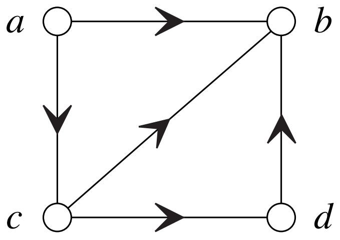


Fig. 2.5. A typical digraph.

Lemma 2.10. *If a digraph contains an infinite sequence of vertices $(a_0, a_1, \ldots)$ such that $a_{i-1}a_i$ is an arc for every i, then the digraph contains a cycle.*

Proof. Any digraph has finitely many vertices, so the sequence $(a_0, a_1, \ldots)$ must contain repetitions. Suppose a_i is repeated; say j is the smallest subscript greater than i such that $a_i = a_j$. Then $(a_i, a_{i+1}, \ldots, a_j)$ is a cycle in g.

In a similar way we can prove the following lemma:

Lemma 2.11. *If a digraph contains an infinite sequence of vertices* $(a_0, a_1, \ldots)$ *such that* $a_{i+1}a_i$ *is an arc for every* i*, then the digraph contains a cycle.*

Theorem 2.12. *Every acyclic digraph has a start and a finish.*

The concept of a complete graph generalizes to the directed case in two ways. The *complete directed graph* on vertex set V, denoted by DK_V, has as its arcs all ordered pairs of distinct members of V, and is uniquely determined by V. On the other hand, one can consider all the different digraphs that can be formed by assigning directions to the edges of the complete graph on V; these are called *tournaments*.

In those cases in which a directed graph is fully determined, up to isomorphism, by its number of vertices, notation is used that is analogous to the undirected case. The directed path, directed cycle, and complete directed graph on v vertices are denoted by DP_v, DC_v, and DK_v respectively.

We shall say that vertex x is *reachable* from vertex y if there is a walk (and consequently a directed path) from y to x. (When x is reachable from y, some authors say "x is a descendant of y" and "y is an ancestor of x.") Two vertices are *strongly connected* if each is reachable from the other, and a digraph (or directed multigraph) is called strongly connected if every vertex is strongly connected to every other vertex. For convenience, every vertex is defined to be strongly connected to itself. We shall say that a directed graph or multigraph is *connected* if the underlying graph is connected, and *disconnected* otherwise. However, some authors reserve the word "connected" for a digraph in which given any pair of vertices x and y, either x is reachable from y or y is reachable from x.

It is clear that strong connectivity is an equivalence relation on the vertex set of any digraph D. The equivalence classes, and the subdigraphs induced by them, are called the *strong components* of D.

Part II

Event Detection Using Graph Distance

3

Matching Graphs with Unique Node Labels

3.1 Introduction

In its most general form, graph matching refers to the problem of finding a mapping f from the nodes of one given graph g_1 to the nodes of another given graph g_2 that satisfies some constraints or optimality criteria. For example, in graph isomorphism detection [130], mapping f is a bijection that preserves all edges and labels. In subgraph isomorphism detection [173], mapping f has to be injective such that all edges of g_1 are included in g_2 and all labels are preserved. Other graph matching problems that require the constructions of a mapping f with particular properties are maximum common subgraph detection [118, 129] and graph edit distance computation [131, 151].

The main problem with graph matching is its high computational complexity, which arises from the fact that it is usually very costly to find mapping f for a pair of given graphs. It is a known fact that the detection of a subgraph isomorphism or a maximum common subgraph and the computation of graph edit distance are NP-complete problems. If the graphs in the application are small, optimal algorithms can be used. These algorithms are usually based on an exhaustive enumeration of all possible mappings f between two graphs. Sometimes application-dependent heuristics can be found that allow us to eliminate significant portions of the search space (i.e., the space of all possible functions f), but still guarantee the correct, or optimal, solution being found. Such heuristics can be used in conjunction with look-ahead techniques and constraint satisfaction [51, 115, 173]. For matching of large graphs, one needs to resort to suboptimal matching strategies. Methods of this type are characterized by an (often low-order) polynomial-time complexity, but they are no longer guaranteed to find the optimal solution for a given problem. A large variety of such suboptimal approaches have been proposed in the literature, based on a multitude of different computational paradigms. Examples include probabilistic relaxation [45, 191], genetic algorithms [54, 183], expectation maximization [122], eigenspace methods [108, 123], and quadratic programming [141].

Another possibility to overcome the problem arising from the exponential complexity of graph matching is to focus on classes of graphs with an inherently lower computational complexity of the matching task. Some examples of such classes are given in [91, 99, 121]. Most recently, in the field of pattern recognition and computer vision, the class of trees has received considerable attention [140, 156].

In this chapter another special class of graphs will be introduced. The graphs belonging to this class are characterized by the existence of unique node labels, which means that each node in a graph possesses a node label that is different from all other node labels in that graph. This condition implies that whenever two graphs are being matched with each other, each node has at most one candidate for possible assignment under function f in the other graph. This candidate is uniquely defined through its node label. Consequently, the most costly step in graph matching, which is the exploration of all possible mappings between the nodes of the two graphs under consideration, is no longer needed. Moreover, we introduce matching algorithms for this special class of graphs and analyze their computational complexity. Particular attention is directed to the computation of graph isomorphism, subgraph isomorphism, maximum common subgraph, graph edit distance, and median graph computation.

If constraints are imposed on a class of graphs, we usually lose some representational power. The class of graphs considered in this chapter is restricted by the requirement of each node label being unique. Despite this restriction, there exist some interesting applications for this class of graphs. From the general point of view, graphs with unique node labels seem to be appropriate whenever the objects from the problem domain, which are modeled through nodes, possess properties that can be used to identify them uniquely. In particular, the condition of unique node labels does not pose a problem when one is dealing with graphs constructed from data collected from computer networks. It is common in these networks that each node, such as a client, server, or router, be uniquely identified. For example, in an intranet employing ethernet technology on a local area network (LAN) segment, either the Media Access Control (MAC), or the Internet Protocol (IP) address could be used to uniquely identify nodes on the local segment. As a consequence, the efficient graph matching algorithms described in Theorem 3.8 can be applied to computer networks to assist in network management functions. Another application of graphs with unique node labels is web document analysis [154].

The remainder of this chapter is organized as follows. In Section 3.2, we introduce our basic concepts and terminology. Graphs with unique node labels and related matching strategies are discussed in Section 3.3. In Section 3.4, we present the results of an experimental study in which the run time of some of the proposed algorithms was measured. Finally, conclusions from this work are drawn in Section 3.5.

3.2 Basic Concepts and Notation

In this section the basic concepts and terminology used throughout this and later chapters of the book will be introduced. We consider directed graphs with labeled vertices (nodes) and edges (links). Let L_V and L_E denote sets of node and edge labels, respectively. A *graph* $g = (V, E, \alpha, \beta)$ is a 4-tuple where V is the finite set of *vertices*, $E \subseteq V \times V$ is the set of *edges*, $\alpha : V \longrightarrow L_V$ is a function assigning labels to the nodes, and $\beta : E \longrightarrow L_E$ is a function assigning labels to edges. Edge $(x, y) \in E$ originates at node $x \in V$ and terminates at node $y \in V$. An undirected graph is obtained as a special case if there exists an edge $(y, x) \in E$ for every edge $(x, y) \in E$ with $\beta(x, y) = \beta(y, x)$.

Let $g = (V, E, \alpha, \beta)$ and $g' = (V', E', \alpha', \beta')$ be graphs; g' is a *subgraph* of g, $g' \subseteq g$, if $V' \subseteq V$, $E' \subseteq E$, $\alpha(x) = \alpha'(x)$ for all $x \in V'$, and $\beta(x, y) = \beta'(x, y)$ for all $(x, y) \in E'$. Let $g \subseteq g'$ and $g \subseteq g''$. Then g is called a *common subgraph* of g' and g''. Furthermore, g is called a *maximum common subgraph* (notation: *mcs*) of g' and g'' if there exists no other common subgraph of g' and g'' that has more nodes and, for a given number of nodes, more edges than g.

For graphs g and g', a *graph isomorphism* is any bijection $f : V \longrightarrow V'$ such that:

(1) $\alpha(x) = \alpha'(x)$ for all $x \in V$; and
(2) for any edge $(x, y) \in E$, there exists $(f(x), f(y)) \in E'$ with $\beta(x, y) = \beta'(f(x), f(y))$, and for any edge $(x', y') \in E'$ there exists an edge $(f^{-1}(x'), f^{-1}(y')) \in E$ with $\beta'(x', y') = \beta(f^{-1}(x'), f^{-1}(y'))$.

If $f : V \longrightarrow V'$ is a graph isomorphism between graphs g and g', and g' is a subgraph of another graph g'', i.e., $g' \subseteq g''$, then f is called a *subgraph isomorphism* from g to g''.

Next we introduce the concept of *graph edit distance* (notation: *ged*), which is based on graph edit operations. We consider six types of edit operations: substitution of a node label, substitution of an edge label, insertion of a node, insertion of an edge, deletion of a node, and deletion of an edge. A cost (i.e., a nonnegative real number) is assigned to each edit operation. Let e be an edit operation and $c(e)$ its cost. The *cost of a sequence of edit operations*, $s = e_1 \dots e_n$, is given by the sum of all its individual costs, i.e., $c(s) = \sum_{i=1}^{n} c(e_i)$. The *edit distance* $d(g_1, g_2)$ of two graphs g_1 and g_2 is equal to the *minimum cost*, taken over all sequences of edit operations, that transform g_1 into g_2. Procedures for *ged* computation are discussed in [131].

Finally, we introduce the median of a set of graphs [100]. Let $G = \{g_1, \dots, g_N\}$ be a set of graphs and U the set of all graphs with labels from L_V and L_E. The *median* $\overline{g}$ of G is a graph that satisfies the condition

$$\sum_{i=1}^{N} d(\overline{g}, g_i) = \min \left\{ \sum_{i=1}^{N} d(g, g_i) \mid g \in U \right\} .$$

It follows that the median is a graph that has the minimum average edit distance to the graphs in set G. It is a useful concept to represent a set of graphs by a single prototype. In many instances the median of a given set G is not unique; nor is it always a member of G. For further details on median graphs see [100].

3.3 Graphs with Unique Node Labels

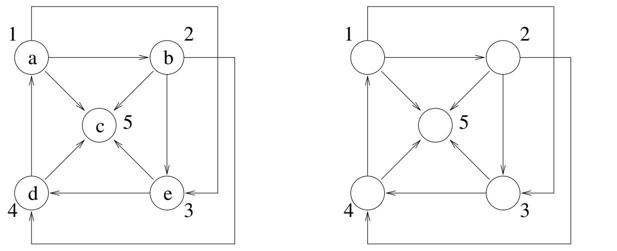
In this section we introduce a special class of graphs that are characterized by the existence of unique node labels. Formally, we require that for any graph g and any pair $x, y \in V$, the condition $\alpha(x) \neq \alpha(y)$ holds if $x \neq y$. Furthermore, we assume that the underlying alphabet of node labels is an ordered set, for example, the integers, i.e., $L_V = \{1, 2, 3, \ldots\}$, or words over an alphabet that can be lexicographically ordered. Throughout this chapter we consider graphs from this class only, unless otherwise mentioned.

Definition 3.1. Let $g = (V, E, \alpha, \beta)$ be a graph. The *label representation* $\rho(g)$ of g is given by $\rho(g) = (L, C, \lambda)$, where:

(1) $L = \{\alpha(x) | x \in V\}$;
(2) $C = \{(\alpha(x), \alpha(y)) | (x, y) \in E\}$; and
(3) $\lambda : C \longrightarrow L_E$ with $\lambda(\alpha(x), \alpha(y)) = \beta(x, y)$ for all $(x, y) \in E$.

According to this definition the label representation of a graph g is obtained by representing each node of g by its (unique) label and dropping the set V. From the formal point of view, $\rho(g)$ defines the equivalence class of all graphs that are isomorphic to g. The individual members of this class are obtained by assigning an arbitrary node, or more precisely an arbitrary node name, to each unique node label, i.e., to each element from L.

Example 3.2. Let $L_V = \{1, 2, 3, 4, 5\}$ and $g = (V, E, \alpha, \beta)$, where $V = \{a, b, c, d, e\}$, $E = \{(a, b), (b, e), (e, d), (d, a), (a, c), (b, c), (d, c), (e, c), (a, e), (b, d)\}$, $\alpha : a \mapsto 1, b \mapsto 2, c \mapsto 5, d \mapsto 4, e \mapsto 3$, $\beta : (x, y) \mapsto 1$ for all $(x, y) \in E$. A graphical illustration of g is shown in Figure 3.1(a), where the node names (i.e., the elements of V) appear inside the nodes and the corresponding labels outside. Because all edge labels are identical, they have been omitted. The label representation $\rho(g)$ of g is then given by the following quantities: $L = \{1, 2, 3, 4, 5\}$, $C = \{(1, 2), (2, 3), (3, 4), (4, 1), (1, 5), (2, 5), (4, 5), (3, 5), (1, 3), (2, 4)\}$, $\lambda : (i, j) \mapsto 1$ for all $(i, j) \in C$.



(a) Example graph g (b) Label representation of g

Fig. 3.1. Example graph g and its label representation.

Intuitively, we can interpret the label representation $\rho(g)$ of any graph g as a graph identical to g up to the fact that all node names are left unspecified. Hence $\rho(g)$ can be conveniently graphically represented in the same way as g is represented. For example, a graphical representation of $\rho(g)$, where g is shown in Figure 3.1(a), is given in Figure 3.1(b).

Lemma 3.3. *Let $g_1 = (V_1, E_1, \alpha_1, \beta_1)$, $g_2 = (V_2, E_2, \alpha_2, \beta_2)$ be two graphs and $\rho(g_1) = (L_1, C_1, \lambda_1)$, $\rho(g_2) = (L_2, C_2, \lambda_2)$ their label representations. Graph g_1 is isomorphic to graph g_2 if and only if $\rho(g_1) = \rho(g_2)$ (i.e., $L_1 = L_2$, $C_1 = C_2$, and $\lambda_1 = \lambda_2$).*

Proof. Assume that there exists a graph isomorphism $f : V_1 \longrightarrow V_2$. Then $\alpha_1(x) = \alpha_2(f(x))$ for all $x \in V_1$. Since f is bijective, it follows that $L_1 = L_2$. Furthermore, because of the conditions on the edges that are imposed by graph isomorphism f, we conclude that $C_1 = C_2$ and $\lambda_1 = \lambda_2$. Conversely, assume that $\rho(g_1) = \rho(g_2)$. Construct now the mapping $f : V_1 \longrightarrow V_2$ such that $f(x) = y$ if and only if $\alpha_1(x) = \alpha_2(y)$. Because $L_1 = L_2$ and the node labels in both g_1 and g_2 are unique, this mapping is a bijection that satisfies the conditions of graph isomorphism imposed on the edges and edge labels in g_1 and g_2.

Based on this lemma, we can examine two graphs for isomorphism by simply generating their label representations and checking the conditions $L_1 = L_2$, $C_1 = C_2$, and $\lambda_1 = \lambda_2$. Assume $n = \max\{|V_1|, |V_2|\}$. Then $|L_1| = |L_2| = \mathcal{O}(n)$, $|E_1| = |C_1| = \mathcal{O}(n^2)$, and $|E_2| = |C_2| = \mathcal{O}(n^2)$. Testing two ordered sets for identity is an operation that is linear in the number of elements. Hence the computational complexity of testing two graphs with unique node labels for isomorphism amounts to $\mathcal{O}(n^2)$.

Lemma 3.4. *Let g_1, g_2, $\rho(g_1)$, and $\rho(g_2)$ be the same as in Lemma 3.3. Then, g_1 is subgraph isomorphic to g_2 if and only if $L_1 \subseteq L_2$, $C_1 \subseteq C_2$, and $\lambda_1(i, j) = \lambda_2(i, j)$ for all $(i, j) \in C_1$.*

Proof. Firstly, we assume that there exists a subgraph isomorphism $f : V_1 \longrightarrow V_2$. Then $\alpha_1(x) = \alpha_2(f(x))$ for all $x \in V_1$. Since f is injective, it follows that $L_1 \subseteq L_2$. Similarly to the proof of Lemma 3.3, we conclude that $C_1 \subseteq C_2$ and $\lambda_1(i, j) = \lambda_2(i, j)$ for all $(i, j) \in C_1$. Conversely, assume $L_1 \subseteq L_2$, $C_1 \subseteq C_2$, and $\lambda_1(i, j) = \lambda_2(i, j)$ for all $(i, j) \in C_1$. Then we can construct an injective mapping $f : V_1 \longrightarrow V_2$ such that $f(x) = y$ if and only if $\alpha_1(x) = \alpha_2(y)$. Similarly to the proof of Lemma 3.3, it follows that this mapping is a subgraph isomorphism from g_1 to g_2.

Using Lemma 3.4, testing two graphs for subgraph isomorphism reduces to examining the corresponding label representations for the three conditions $L_1 \subseteq L_2$, $C_1 \subseteq C_2$, and $\lambda_1(i, j) = \lambda_2(i, j)$ for all $(i, j) \in C_1$. The third condition can be checked in $\mathcal{O}(|C_1|) = \mathcal{O}(n^2)$ time. Checking whether an ordered set is a subset of another ordered set is linear in the size of the larger of the two sets. Hence the computational complexity of subgraph isomorphism of graphs with unique node labels is $\mathcal{O}(n^2)$.

Lemma 3.5. *Let g_1, g_2, $\rho(g_1)$, and $\rho(g_2)$ be the same as in Lemma 3.3. Let g be a graph with $\rho(g) = (L, C, \lambda)$ such that $L = L_1 \cap L_2$, $C = \{(i, j)|(i, j) \in C_1 \cap C_2$, and $\lambda_1(i, j) = \lambda_2(i, j)\}$, and $\lambda(i, j) = \lambda_1(i, j)$ for all $(i, j) \in C$. Then g is an mcs of g_1 and g_2.*

Proof. First we note that $L \subseteq L_1$ and $L \subseteq L_2$. Hence $V \subseteq V_1$ and $V \subseteq V_2$ for any graph g with label representation $\rho(g) = (L, C, \lambda)$. Similarly, because C includes a pair (i, j) if and only if a corresponding edge with identical labels exists in both g_1 and g_2, we observe that $E \subseteq E_1$ and $E \subseteq E_2$ for any such graph g. Thirdly, the labels of edges (x, y) occurring in both g_1 and g_2 are preserved under λ. Hence g is a subgraph of both g_1 and g_2. Now assume that g is not an *mcs*. In this case there must exist another subgraph g' of both g_1 and g_2 with either more nodes than g or the same number of nodes, but with more edges. The first case contradicts the way set L is constructed; if g' has more nodes than g, then $L \neq L_1 \cap L_2$. The second case is in conflict with the construction of C and λ, i.e., if g' has the same number of nodes as g, but more edges, then C and λ must be different from their values stated in Lemma 3.5. Hence g must be indeed an *mcs* of g_1 and g_2. This concludes the proof.

Possible computational procedures implied by Lemma 3.5 are again based on the intersection of two ordered sets. Hence the complexity of computing the *mcs* of two graphs with unique node labels is $\mathcal{O}(n^2)$.

In [21] a detailed analysis was provided showing how *ged* depends on the costs associated with the individual edit operations. A set of edit operations together with their cost is also called a *cost function*. In this chapter we focus our attention on the following cost function: $c_{nd}(x) = c_{ni}(x) = 1$, $c_{ns}(x) = \infty$, $c_{ed}(x, y) = c_{ei}(x, y) = c_{es}(x, y) = 1$, where $c_{nd}(x)$, $c_{ni}(x)$, and $c_{ns}(x)$ denote the costs associated with the deletion, insertion, and substitution of node x, while $c_{ed}(x, y)$, $c_{ei}(x, y)$, and $c_{es}(x, y)$ denote the cost associated with the deletion, insertion, and substitution of edge (x, y), respectively. This cost function is simple in the sense that each edit operation has a cost equal to one, except for node substitutions, which have infinite cost. It is easy to see that for any two graphs g_1 and g_2, there always exists a sequence of edit operations that transforms g_1 into g_2 with a finite total cost (for example, a sequence that deletes all nodes and edges from g_1, and inserts all nodes and edges in g_2). Hence edit operations with infinite cost will never be applied in the computation of any actual *ged*. This means that node substitutions will never be applied and may be considered inadmissible under the cost function introduced above, while all other edit operation can be applied and have the same cost. The exclusion of node substitutions for graphs with unique node labels makes sense since node label substitutions may generate graphs with nonunique node labels, i.e., graphs that do not belong to the class of graphs under consideration.

Lemma 3.6. *Let g_1, g_2, $\rho(g_1)$, and $\rho(g_2)$ be the same as in Lemma 3.3. Furthermore, let $C_0 = \{(i, j) | (i, j) \in C_1 \cap C_2$ and $\lambda_1(i, j) = \lambda_2(i, j)\}$, and $C_0' = \{(i, j) | (i, j) \in C_1 \cap C_2$ and $\lambda_1(i, j) \neq \lambda_2(i, j)\}$. Then $d(g_1, g_2) = |L_1| + |L_2| - 2|L_1 \cap L_2| + |C_1| + |C_2| - 2|C_0| + |C_0'|$.*

Proof. Because node substitutions can be regarded inadmissible, the minimum cost sequence of edit operations transforming g_1 into g_2 assigns each node $x \in V_1$ with label $\alpha_1(x)$ to node $y \in V_2$ with $\alpha_1(x) = \alpha_2(y)$. If no node $y \in V_2$ exists with this property, node x is deleted from g_1. Similarly, all nodes $y \in V_2$ for which no node $x \in V_1$ exists with $\alpha_1(x) = \alpha_2(y)$ will be inserted in g_2. This leads to $|L_1| - |L_1 \cap L_2|$ node deletions in graph g_1, and $|L_2| - |L_1 \cap L_2|$ node insertions in graph g_2, each having a cost equal to one. Hence the total cost arising from edit operations on the nodes of g_1 and g_2 amounts to $|L_1| - |L_1 \cap L_2| + |L_1| - |L_1 \cap L_2| = |L_1| + |L_2| - 2\,|L_1 \cap L_2|$.

We now consider the edges. There exist $|C_1| - |C_0|$ edges in g_1 that do not occur in g_2, and need to be deleted. Similarly, there exist $|C_2| - |C_0|$ edges in g_2 that do not have a counterpart in g_1, and need to be inserted. Furthermore, there are two types of edges corresponding to set $C_1 \cap C_2$. The first type are edges $(i, j) \in C_0$, for which $\lambda_1(i, j) = \lambda_2(i, j)$. No edit operations are needed for edges of this kind. The second type are edges $(i, j) \in C_0^{'}$, for which $\lambda_1(i, j) \neq \lambda_2(i, j)$. An edge substitution with cost one is needed for each such edge. Hence the total cost of edit operations on the edges of g_1 and g_2 is equal to $|C_1| - |C_0| + |C_2| - |C_0| + |C_0^{'}| = |C_1| + |C_2| - 2\,|C_0| + |C_0^{'}|$. This concludes the proof.

Possible computational procedures for *ged* computation implied by Lemma 3.6 are based again on the intersection of two ordered sets. Hence, similar to all other graph matching procedures considered before, the complexity of edit distance computation of graphs with unique node labels is $\mathcal{O}(n^2)$.

Finally, we turn to the problem of computing a graph g that is the median of a set of graphs, $G = \{g_1, \ldots, g_N\}$, with unique node labels. In the remainder of this section we assume, for the purpose of notational convenience and without restricting generality, that all graphs under consideration are complete. That is, there is an edge $(x, y) \in E$ between any pair of nodes $x, y \in V$ for any considered graph g. "Real" edges can be easily distinguished from "virtual" edges by including a special null symbol in the edge label alphabet L_E and defining $\beta(x, y) = null$ for any virtual edge. The benefit we get from considering complete graphs is that the only necessary edit operations on the edges are substitutions. In other words, any edge deletion or insertion now becomes a substitution that involves the null label. No conflicts will arise from this simplification because the cost of edge substitutions, deletions, and insertions are the same.

Let $\rho(g_1), \ldots, \rho(g_N)$ be the label representations of $g_1, \ldots, g_N$. Define $L_U = \bigcup_{i=n}^{N} L_i$ and $C_U = \bigcup_{i=1}^{N} C_i$. Furthermore, let $\gamma(i)$ be the total number of occurrences of node label $i \in L_U$ in $L_1, \ldots, L_N$. Note that $(1 \leq \gamma(i) \leq N)$. Formally, $\gamma(i)$ can be defined through the following procedure:

```
$\gamma(i) = 0$;
for $k = 1$ to $N$ do
    if $i \in L_k$ then $\gamma(i) = \gamma(i) + 1$
```

Next, we define $\rho(g) = (L, C, \lambda)$ such that

(1) $L = \{i \mid i \in L_U \text{ and } \gamma(i) \geq N/2\}$;
(2) $C = \{(i, j) \mid i, j \in L\}$; and
(3) $\lambda(i, j) = max\_label(i, j)$,

where function $max\_label(i, j)$ returns the label $\lambda_k(i, j) \in L_E$ that has the maximum number of occurrences on edge (i, j) in $C_1, \ldots, C_N$. In case of a tie, any of the competing labels $\lambda_k(i, j)$ may be returned.

Lemma 3.7. *Let G and $\rho(g)$ be as above. Then any graph g with label representation $\rho(g)$ is a median graph of G.*

Proof. The smallest potential median graph candidate is the graph with an empty set of nodes, while the largest potential candidate corresponds to the case $L = L_U$. The second observation is easy to verify, because any graph g^* that includes more node labels will have at least one label k^* that does not occur in any of the L_i's. Hence the node with label k^* will be deleted in all of the distance computations for $d(g^*, g)$, $i = 1, \ldots, N$. Therefore dropping the node with label k^* from g^* will produce a graph with a smaller average edit distance to the members of G. It follows that for any median graph g with node label representation $\rho(g)$, set L must necessarily be a subset of L_U.

If we substitute the expression derived in Lemma 3.6 into the definition of a median graph given in Section 3.2, we recognize that any median graph g with node label representation $\rho(g)$ must minimize the following expression:

$$\begin{aligned}\Delta =& |L| + |L_1| - 2\,|L \cap L_1| + |C| + |C_1| - 2\,|C_{01}| + |C'_{01}| + \cdots \\ &+ |L| + |L_N| - 2\,|L \cap L_N| + |C| + |C_N| - 2\,|C_{0N}| + |C'_{0N}|,\end{aligned}$$

where we use the following notation (see Lemma 3.6):

$$\begin{aligned}C_{0k} &= \{(i, j) \mid (i, j) \in C \cap C_k \text{ and } \lambda(i, j) = \lambda_k(i, j)\}\,, \\ C'_{0k} &= \{(i, j) \mid (i, j) \in C \cap C_k \text{ and } \lambda(i, j) \neq \lambda_k(i, j)\}\,.\end{aligned}$$

Clearly, Δ can be rewritten as

$$\Delta = N\,|L| + \sum_{i=1}^{N} |L_i| - 2 \sum_{i=1}^{N} |L \cap L_i| + N\,|C| + \sum_{i=1}^{N} |C_i| - 2 \sum_{i=1}^{N} |C_{0i}| + \sum_{i=1}^{N} |C'_{0i}|.$$

Note that all quantities are nonnegative integers. Since all L_i's and C_i's are given, minimization of Δ is equivalent to minimizing

$$N\,|L| - 2 \sum_{i=1}^{N} |L \cap L_i| + N\,|C| - 2 \sum_{i=1}^{N} |C_{0i}| + \sum_{i=1}^{N} |C'_{0i}|.$$

First we analyze the term $\Delta_1 = N\,|L| - 2\,\sum_{i=1}^{N} |L \cap L_i|$. It is obvious that $N\,|L|$ will become smaller if we include fewer nodes in the median graph. On the other hand, this will also make the term $2\,\sum_{i=1}^{N} |L \cap L_i|$ smaller, which leads to an increase of Δ_1. To find the optimal number of nodes to be included in the median graph, we consider each element of L individually and decide whether it must be included in the median graph. From the definition of Δ_1 it follows that if a node with label i is included in the median graph, its contribution to Δ_1 will be $N - 2\,\gamma(i)$. Conversely, if that node is not included, its contribution will be zero. Hence, in order to minimize Δ_1, we include a node with label i in the median graph if $N - 2\,\gamma(i) \leq 0$, which is equivalent to $\gamma(i) \geq N/2$.

Now consider the term $\Delta_2 = N\,|C| - 2\,\sum_{i=1}^{N} |C_{0i}| + \sum_{i=1}^{N} |C'_{0i}|$. Assume for the moment that $|C|$ is a constant that is defined through the choice of L. Then we have to minimize $-2 \sum_{i=1}^{N} |C_{0i}| + \sum_{i=1}^{N} |C'_{0i}|$. Since $|C_{0i}| + |C'_{0i}| = |C \cap C_0|$, this is equivalent to maximizing $|C_{0i}|$. However, such a maximization is exactly what is accomplished by function *max_label*. This function chooses, for edge (i, j), the label that most often occurs on edge (i, j) in all the given graphs.

So far we have treated the terms Δ_1 and Δ_2 independently of each other. In fact, they are not independent because the exclusion of a node with label i from the median graph implies exclusion of any of its incident edges (i, j) or (j, i). Therefore the question arises whether this dependency can lead to an inconsistency in the minimization of $\Delta = \Delta_1 + \Delta_2$ in the sense that decreasing Δ_1 leads to an increase of Δ_2 by a larger amount, and vice versa. It is easy to see that such an inconsistency can never happen. First of all, exclusion of an edge (i, j) for the sake of minimizing Δ_2 does not imply any constraints on inclusion or exclusion of any of the incident nodes i and j. Secondly, if node i is not included because $\gamma(i) < N/2$, function *max_label* will surely return the *null* label for any edge (i, j) or (j, i). This is equivalent to not including (i, j) or (j, i) in the median graph. In other words, if a node i is not included in the median graph because $\gamma(i) < N/2$, the dependency between Δ_1 and Δ_2 leads also to not including all incident edges, which is exactly what is required to minimize Δ_2. This concludes the proof of Lemma 3.7.

In order to derive a practical computational procedure for the computation of a median of a set of graphs with unique node labels, we need to implement functions $\gamma(i)$ and *max_label*(i, j). It is easy to verify that the complexities of these two functions are $\mathcal{O}(n\,N)$ and $\mathcal{O}(n^2\,N)$, respectively. It follows that the median graph computation problem can be solved in $\mathcal{O}(n^2\,N)$ time for graphs with unique node labels.

So far, we have assumed that there are $\mathcal{O}(n^2)$ edges in a graph with n nodes. There are, however, applications in which the graphs are of bounded degree, i.e., the maximum number of edges incident to a node is bounded by a constant κ. In this case all of the expressions $\mathcal{O}(n^2)$ reduce to $\mathcal{O}(n)$.

The following theorem summarizes all the results derived in this section.

Theorem 3.8. *For the class of graphs with unique node labels there exist computational procedures that solve the following problems in quadratic time with respect to the number of nodes in the underlying graph:*

(1) graph isomorphism;
(2) subgraph isomorphism;
(3) maximum common subgraph; and
(4) graph edit distance under the cost function introduced earlier in this section.

The median graph computation problem can be solved in $\mathcal{O}(n^2\,N)$ time, where n is the number of nodes in the largest graph and N is the number of given graphs.

3.4 Experimental Results

The aim of the experiments described in this section is to verify the low computational complexity for the theoretical results derived in Section 3.3. The time taken to compute isomorphism, subgraph isomorphism, *mcs*, and *ged* are measured for graphs ranging in size from hundreds of nodes to tens of thousands of nodes, and with different edge densities. In addition, we validate the linear dependency of time taken to compute a median graph to the number of graphs from which the median is derived. Computation times are measured for synthetic data sets and real network data. Real network data were acquired from a link in the core of a wide area computer network. A test for similarity of computational time measurements for real and synthetic data sets is made to verify that results achieved for simulated networks can be repeated for real-world implementations. An experiment was conducted to verify that the times taken to compute algorithms in this chapter are independent of network topology. Two graph generators were used to produce synthetic data sets having different network topologies. The real network data set was used as a third sample having different topology. Graphs in each data set had to be equivalent in number of nodes and links for this test.

The hardware platform used to measure computational times was a SUN Fire V880 with 4 $\times$ 750MHz UltraSparc3 processors and 8GB of RAM. The specific hardware platform used to perform the experiments is not important and has been provided for completeness only. Only relative computational times with respect to graph dimensions are important.

3.4.1 Synthetic Network Data

Synthetic data sets are used to validate the computational complexity of procedures defined in Section 3.3. These data sets are also used to verify that the procedures are independent of network topology.

Two data sets have been produced using normally distributed random edges with edge densities of 2.5% and 10%, respectively. An edge density of 2.5% was used so that graphs with 20000 nodes could be synthesized without exceeding computer memory of the computer platform used for the experiments. The data set with 10% edge density was chosen to mimic the characteristics of the real data network. The maximum number of nodes possible for graphs in this data set was 10000.

An additional single synthetic data set, having edge density of 2.5%, was created using a Fan Chung algorithm [47]. This graph generator produced graphs having vertex degrees with a power-law distribution. The resultant topology of graphs produced using this method is quite different from those of graphs having normally distributed random edges. In fact, graphs having degree distribution that are power laws are characteristic of large networks, such as the Internet [167].

For each synthetic data set we first obtain a series S of graphs, containing 100, 1000, 3000, 5000, 7000, and 10000 nodes. For data sets with edge density of 2.5% we obtain an additional graph in the series that has 20000 nodes. The resulting graphs have directed edges with Poisson distributed edge weights. A second series S' was produced as a counterpart, using the same procedure, for measurements of computational times for *mcs* and *ged*.

A further set of graphs was created to verify the linear increase in computational time with an increase in edge density, for a fixed number of nodes. The graph generator assigned edges using a normal distribution. For this data set, graphs had 5000 vertices and edge densities ranging from 1% to 10% in steps of 1%. A counterpart was created for each graph to be used for *mcs* and *ged* computations.

To compare computational times of algorithms measured for synthetic data against real data sets, we created two randomly distributed graphs having the same number of vertices and edges as each of the real data sets in Section 3.4.2.

Finally, for the validation of computational times for median graphs we created a series of 100 graphs using randomly distributed edges. In this series the average number of vertices and edge density are matched to our business domain network data set (i.e., comprising graphs having on average 70 vertices with edge density of 10%) as described in Section 3.4.2.

3.4.2 Real Network Data

Real network data were acquired from a core link in a large-enterprise data network using network performance monitoring tools. The data network employs static IP addresses, hence its suitability for representation by the class of graphs defined in this chapter. Graphs were produced from traffic traversing the link at intervals of one day. This resulted in a time series of 100 graphs representing 100 days of network traffic.

Two levels of abstraction have been used to produce the time series of real network data. Both have quite different characteristics. The first data set has graph vertices that represent IP addresses, while the second has vertices that represent business domains. In both data sets, edges represent logical links, and edge weights represent the total number of bytes communicated between vertices in one day. The business domain abstraction is created by coalescing IP addresses belonging to a predefined business entity into a single node. This resulted in graphs that contain on average 70 nodes with edge densities of 10%. The IP network abstraction has graphs that have on average 9000 nodes with an edge density of 0.04%. The low edge density is a result of the near bipartite nature of the graphs arising from data collected at a single point in the core of the enterprise data network. The business domain and IP network abstractions are of interest to network administrators because they provide both coarse and fine network performance data, respectively.

Two consecutive graphs were chosen from each of the real network data set abstractions, to be used in comparisons of computational times of algorithms with times measured for synthetic data. The two graphs chosen for the business domain abstraction contained approximately 90 vertices with an edge density of 10%, while the graphs chosen from the IP abstraction contained 9000 vertices with an edge density of 0.04%.

To verify median graph computational times the whole 100-day time series of graphs of business domain data was used.

3.4.3 Verification of $\mathcal{O}(n^2)$ Theoretical Computational Complexity for Isomorphism, Subgraph Isomorphism, MCS, and GED

To measure the time taken to compute a test for graph isomorphism we select the first graph g_1 from S, containing one hundred unique nodes, and make an exact copy g_2. The fact that $g_2 = g_1$ guarantees that the graphs tested are in fact isomorphic to each other.

The computational time measurement does not include the time taken to derive the label representations $\rho(g_1)$ and $\rho(g_2)$ for graphs g_1 and g_2. This is true for all computational times measured for each algorithm. For the measurement of computational time for the subgraph isomorphism test, we use the same graph g_1, together with graph g_3, obtained by removing 20% of the edges from g_1. The graph g_3 is obviously a subgraph of g_1. The measurements of time to compute both *mcs* and *ged* required both graph series S and S'. To measure the time taken to execute these algorithms we again use g_1 from S, and select the equivalent-size graph from S'. The procedures outlined above were repeated for all three synthetic data sets for graph sizes 1000, 3000, 5000, 7000, 10000, and 20000 (where present).

Table 3.1. Computational times for isomorphism.

	Computational times in seconds for graphs with N vertices						
	N=100	1000	3000	5000	7000	10000	20000
Fan Chung 2.5%	0.01	0.55	5.48	17.41	36.77	78.97	343.77
Random 2.5%	0.02	0.5	5.33	17.74	37.94	81.63	383.80
Random 10%	0.02	2.00	25.16	77.75	163.22	357.72	

The results of all computational time measurements are shown in Tables 3.1, 3.2, 3.3, and 3.4. As expected, the measured computational complexity of all matching algorithms is $\mathcal{O}(n^2)$. Figures 3.2, 3.3, 3.4, and 3.5 illustrate this observation for isomorphism, subgraph isomorphism, mcs, and ged, respectively; the x-axis corresponds

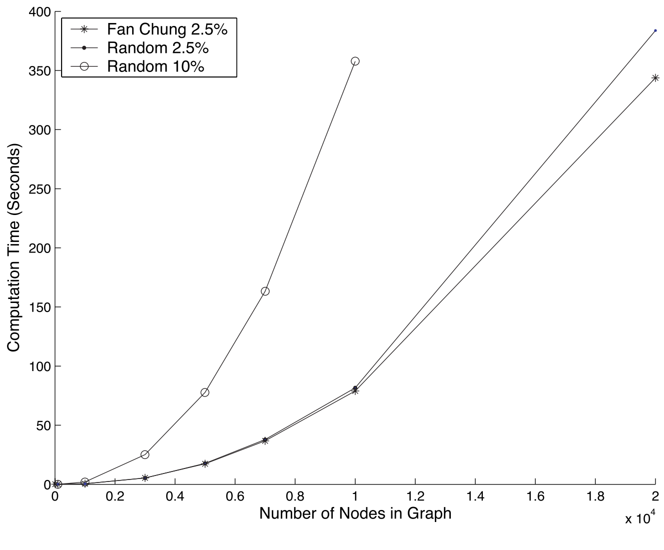


Fig. 3.2. Computational times for Graph Isomorphism.

Table 3.2. Computational times for subgraph isomorphism.

	Computational times in seconds for graphs with N vertices						
	N=100	1000	3000	5000	7000	10000	20000
Fan Chung 2.5%	0.01	0.38	3.04	10.04	21.33	45.60	197.32
Random 2.5%	0.01	0.33	2.82	9.51	20.71	45.03	206.90
Random 10%	0.01	1.17	13.92	43.23	90.23	195.68	

Table 3.3. Computational times for maximum common subgraph.

	Computational times in seconds for graphs with N vertices						
	N=100	1000	3000	5000	7000	10000	20000
Fan Chung 2.5%	0.01	0.38	3.44	11.21	24.28	52.36	230.63
Random 2.5%	0.01	0.40	3.18	10.74	23.63	51.69	237.16
Random 10%	0.01	1.28	16.21	49.40	102.27	221.55	

to the number of nodes in a graph and the y-axis represents the time, in seconds, to compute each graph matching algorithm. Figures show greater computational times for larger edge densities. This result was anticipated due to the dependency on graph elements. Computation times to test for graph isomorphism were the longest. Testing for subgraph isomorphism required the least time to compute. This was a consequence of removing 20% of edges from g_1 to produce a subgraph g_2. The smaller the size

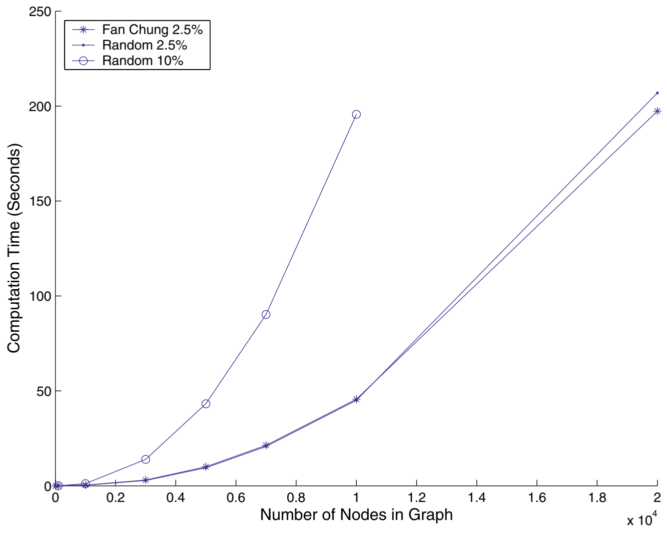


Fig. 3.3. Computational times for subgraph isomorphism.

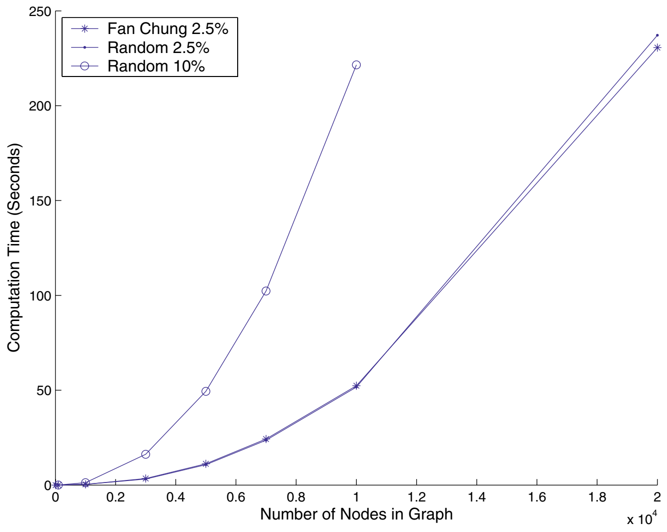


Fig. 3.4. Computational times for maximum common subgraph.

Table 3.4. Computational times for graph edit distance.

	Computational times in seconds for graphs with N vertices						
	N=100	1000	3000	5000	7000	10000	20000
Fan Chung 2.5%	0.01	0.40	3.49	11.34	24.46	52.09	230.51
Random 2.5%	0.01	0.33	3.38	10.90	23.63	51.45	232.09
Random 10%	0.01	1.35	16.30	49.00	101.88	220.26	

of g_2 with respect to g_1, the shorter the time taken to compute the subgraph isomorphism. The computational times for both mcs and ged, as observed in Figures 3.4 and 3.5, are almost indistinguishable. This is not surprising, since the computational steps proposed in Lemmas 3.5 and 3.6 are nearly identical. In all cases the computational times measured for both randomly distributed edges and those with power-law degree distributions, for edge density of 2.5%, are nearly identical. The results would be identical if the numbers of edges in graphs from both data sets were equal. Since the graph generator used to produce graphs with randomly distributed edges create on average graphs with a specified edge density, the actual number of edges can vary. The closeness of results verifies the independence of the algorithms from network topology.

Further experimentation was performed to show the linear dependency of computational complexity in number of edges in a graph with fixed number of vertices. Figure 3.6 shows results for the four graph algorithms. Observation of these results reveals the linear relationship.

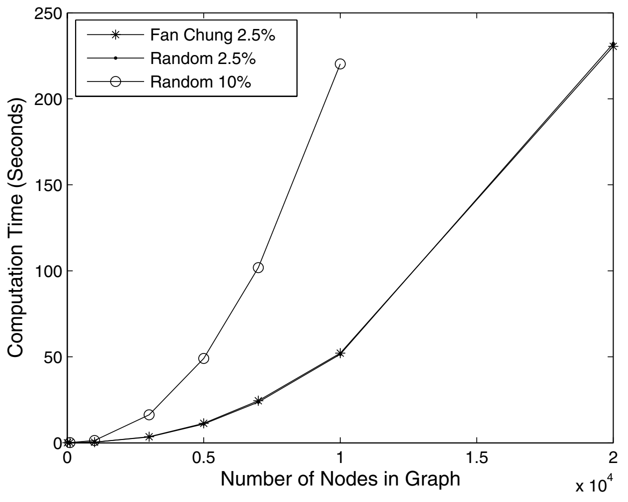


Fig. 3.5. Computational times for graph edit distance.

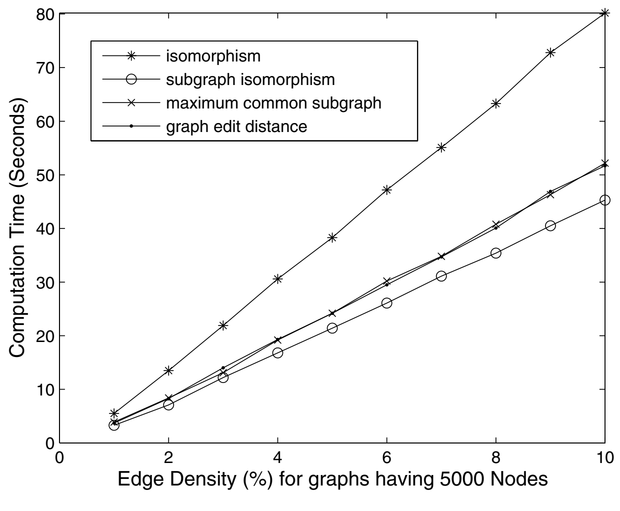


Fig. 3.6. Plot of linear dependency of computational times vs. edge density.

3.4.4 Comparison of Computational Times for Real and Synthetic Data Sets

In this section measurements of computational times of isomorphism, subgraph isomorphism, mcs, and ged on the two real network data sets (i.e., business domain and IP level abstractions) and their synthetic counterparts are described. The aim was to confirm that synthetic data measurements are consistent with those measured for real network data.

The label representation is first derived for the two graphs in each data set. Isomorphism and subgraph isomorphism computation requires only one of the graphs from each set. Both graphs are required for the mcs and ged computations. The results are given in Table 3.5. It can be seen that all measurements between real and equivalent synthetic data sets agree. This implies that results obtained for synthetic data are consistent to those for real data.

Table 3.5. Comparison of computational times for real and synthetic network data.

	Real network data			
	Business domain		IP domain	
	Real	Synthetic	Real	Synthetic
Isomorphism	0.02	0.02	0.60	0.58
Subgraph Isomorphism	0.01	0.01	0.35	0.32
MCS	0.01	0.01	0.38	0.36
GED	0.01	0.01	0.38	0.36

Table 3.6. Computational times for median graph.

Number of graphs in median	Computational time (seconds)	
	Real data	Synthetic data
10	2.21	3.36
20	2.75	4.72
30	3.17	5.37
40	3.53	6.00
50	3.83	6.39
60	4.08	6.76
70	4.27	7.16
80	4.16	7.47
90	4.48	7.79
100	4.93	8.15

3.4.5 Verification of Theoretical Computational Times for Median Graph

The computational time of the median graph algorithm described in Section 3.3 is measured for both real and synthetic data sets. Each contains a time series of one hundred graphs. The sizes of graphs within each time series and between time series are similar. We wish to verify that the time taken to compute a median graph increases linearly as the number of graphs in the median computation increases.

Measurements commenced by taking the first 10 graphs in the time series (i.e., $\{g_1, \ldots, g_{10}\}$) of each data set and computing the median graph. The procedure was repeated using the first 20, 30, 40, ..., 90, and 100 graphs. The results are given in Table 3.6 and Figure 3.7. Both real and synthetic data sets show a linear dependency of computational time with respect to the number of graphs. The plot of computational times for the real data set deviates from a straight line because the edge counts in this data set had a greater standard deviation than those of the synthetic data set. The number of vertices and edges in graphs belonging to real and synthetic data sets can been seen in Figure 3.8.

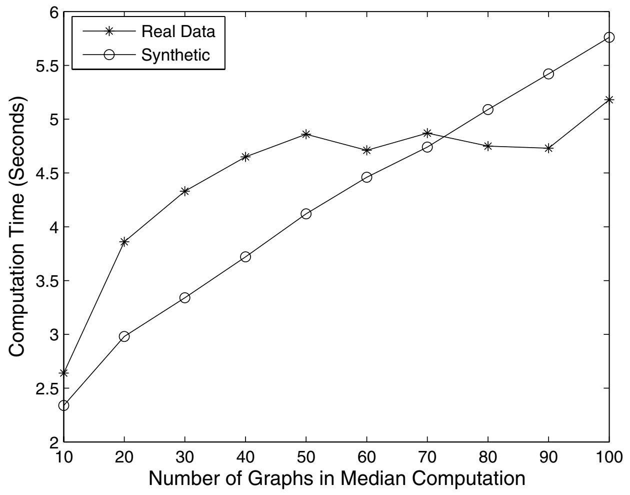


Fig. 3.7. Computational times for median graph algorithm.

3.5 Conclusions

Graph matching is finding many applications in the fields of science and engineering. In this chapter we considered a special class of graphs characterized by unique node labels. A label representation is given for graphs in this class. For a given graph it contains a set of unique vertex labels of the graph, an edge set based on vertex labels, and a

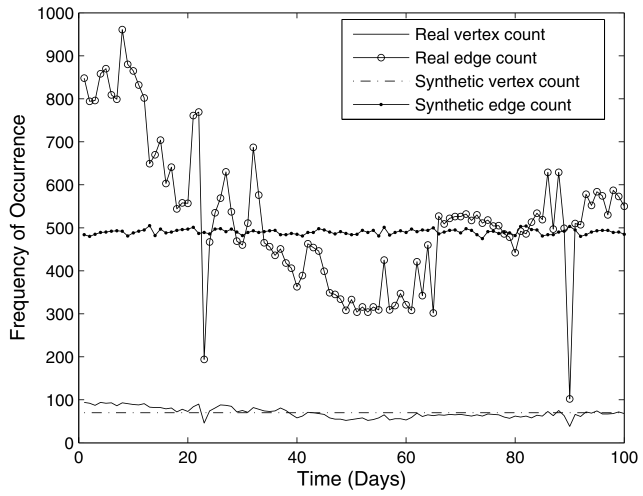


Fig. 3.8. Vertex and edge counts for real and synthetic data sets.

set of edge weights. A number of computationally efficient matching algorithms were derived for this class of graphs. The suitability of applying these matching algorithms to computer network monitoring was addressed.

The matching algorithms that have been derived for graphs having a label representation are detection of graph isomorphism and subgraph isomorphism, computation of maximum common subgraph, graph edit distance, and median graph. The theoretical computational complexity of these algorithms, for graphs having n nodes, is $\mathcal{O}(n^2)$. It was also shown that the time taken to compute a median graph increases linearly with the number of graphs in the set from which the median is computed. Theoretical results were verified using real and synthetic data sets.

It is possible to apply the derived matching algorithms to computer network monitoring since the constraint for unique node labels can be satisfied. In computer networks, nodes can be uniquely identified by means of the media access control or Internet Protocol addresses. The matching algorithms proposed can be used to measure change that occurs in a computer network over time. Measures of network change provide good indicators of when abnormal network events have occurred. Such techniques greatly enhance computer network management, especially in the field of performance monitoring.

The theoretical computational complexity of the matching algorithms were verified through experimentation using synthetic and real network data. Synthetic data sets of graphs with specified numbers of nodes and edge densities were used for this purpose. In addition, synthetic data sets having different network topologies were used to show that the computation times for derived algorithms are independent of network topology. A comparison of results achieved for synthetic data sets with those obtained using data acquired from a large wide area computer network of equal dimension were shown to agree. This outcome, along with the knowledge that the algorithms are independent of network topology, means that simulation of performance of the algorithms on synthetic data can be used to accurately predict the performance that will be achieved for real networks.

In conclusion, graph matching algorithms for uniquely labeled graphs having a label representation provide a significant computational saving compared to the generalized class of graphs, where such matching algorithms have an exponential computational complexity. In this chapter we have shown that for this class of graphs we have been able to apply matching algorithms to graphs having many thousands of nodes.

While this chapter focused on applications to network analysis, it is important to note that the class of graphs described, and the matching algorithms, can be used for any application in which graphs have a unique node labeling. In particular, there are applications in content-based web document analysis [154].

4

Graph Similarity Measures for Abnormal Change Detection

4.1 Introduction

In managing large-enterprise data networks, the ability to measure network changes in order to detect abnormal trends is an important performance monitoring function [17]. The early detection of abnormal network events and trends can provide advance warning of possible fault conditions [171], or at least assist with identifying the causes and locations of known problems.

Network performance monitoring typically uses statistical techniques to analyze variations in traffic distribution [84, 98] or changes in topology [189]. Visualization techniques are also widely used to monitor changes in network performance [8]. To complement these approaches, specific measures of change at the network level in both logical connectivity and traffic variations are useful in highlighting when and where abnormal events may occur in the network [57,59–64,157]. Using these measures, other network management tools may then be focused on problem regions of the network for more detailed analysis.

This chapter examines various measures of network change based on the concept of graph distance. The aim is to identify whether in using these techniques, significant changes in logical connectivity or traffic distributions can be observed between large groups of users communicating over a wide area data network. This data network interconnects some 120, 000 users around Australia. For the purposes of this study, a network management probe was attached to a physical link on the wide area network backbone, and traffic statistics of all data traffic operating over the link were collected. From this information a logical network of users communicating over the physical link is constructed.

Communications between user groups (business domains) within the logical network over any one day is represented as a directed graph. Edge direction indicates the direction of traffic transmitted between two adjacent nodes (business domains) in the network, with edge labels (also called edge-weight) indicating the amount of traffic carried. A subsequent graph can then describe communications within the same network for the following day. This second graph can be compared with the original graph, using a measure of distance between the two graphs to indicate the degree of change occurring in the logical network. The more dissimilar the graphs, the greater the graph distance value. By continuing network observations over subsequent days, the graph distance scores provide a trend of the logical network's relative dynamic behavior as it evolves over time.

This problem becomes one of finding good graph distance measures that are sensitive to abnormal change events but insensitive to typical variations in logical network connectivity or traffic. In addition to graph distance measures, it is also necessary to readily identify where in a network the abnormal change has occurred. This requires the location of regions in the graph that contributed most to the measured change.

The chapter is structured in the following way. Section 4.2 describes how a telecommunication system can be represented as a graph and provides details of how the network traffic was sampled. Graph distance measures suited to changes in logical network connectivity are assessed in Section 4.3, with Section 4.4 examining distance measures aimed at variations in traffic. Measures exploiting graph structures are discussed in Section 4.5, and Section 4.6 applies localization approaches to a particular abnormal event in the sampled network traffic. Concluding remarks are presented in Section 4.7.

4.2 Representing the Communications Network as a Graph

Communications over the data network are represented as graphs. Network nodes (clusters of users in common business domains or individual servers and clients) are represented by vertices in the graph, and edges represent logical links (data transmissions or information transfers) between the nodes. The graph $g = (V, E, \alpha, \beta)$ representing a communications network is vertex-labeled with labeling function $\alpha : V \rightarrow L_V$ assigning vertex identifiers from L_V to individual vertices, such that $\alpha(u) \neq \alpha(v), \forall u, v \in V, u \neq v$, since the graph possesses a unique vertex-labeling (see Chapter 3). Edges are also labeled to indicate the amount of traffic observed over a finite time interval, with labeling function $\beta : E \rightarrow \mathbb{R}^+$. The label of an edge is also called an edge-weight, and edge labeling function β is sometimes referred to as an edge-weighting function. The number of vertices in a graph $g = (V, E, \alpha, \beta)$ is denoted by $|V|$, and likewise, the number of edges is denoted by $|E|$.

A sample of real network traffic was chosen for the purpose of verifying whether such traffic exhibits periods of uncharacteristic behavior, and whether the graph distance measures being considered can highlight such differences. For this investigation, sample traffic statistics were collected from the network management system of the wide area data network. A traffic probe was installed on a single physical link in the network and traffic parameters were logged over 24-hour periods in daily log files. A traffic log file contains information on the logical originators and destinations of traffic (derived from network address information) and the volume of traffic transmitted between OD pairs. To reduce the overall number of OD pairs in the data set, individual users (network addresses) were clustered by the business domain they belonged to on the data network. The aggregated logical flows of traffic between business domains observed over this physical link in a day were then represented as a directed and labeled graph. Vertex-weight identified the business domains of logical nodes communicating over the physical link with edge-weight denoting the total traffic transmitted between corresponding OD pairs over a 24-hour period.

Successive log files collected over subsequent days produced a time series of corresponding directed and labeled graphs representing traffic flows between business domains communicating over the physical link in the network. Log files were collected continuously over the period 9 July 1999 to 24 December 1999. Weekends, public holidays, and days for which probe data were unavailable were removed to produce a final set of 102 log files representing the successive business days' traffic. The graph distance measures examined in this chapter produce a distance score indicating the dissimilarity between two given graphs. Successive graphs derived from the 102 log files of the network data set are compared using the various graph distance measures to produce a set of distance scores representing the change experienced in the network from one day to the next.

Operators in the network management center are interested in identifying the causes of change in traffic volume over the physical link caused by changes in user behavior on the network. Primarily, these effects are caused by the introduction of new applications or services at a local user level, and changes in communities of interest or user groups communicating over particular physical links. Detecting significant changes in logical connectivity is useful in tracking changes in user groups, while logical traffic variations between OD pairs assist with the identification of new applications or services consuming large amounts of network capacity. Distance measures assessing change in topology are required for measuring change in communities of interest between user groups, with traffic-based distance measures required for measurements of change to logical traffic distributions. When significant abnormal change is observed in either connectivity or traffic patterns over successive business days, operators then require identification of the regions within the logical network that contribute most to the overall change observed.

4.3 Graph Topology-Based Distance Measures

Graph distance provides a measure of the difference (or similarity) between graphs. The edit distance, as introduced in Chapter 3, can be used to measure the distance of two given graphs. In this section we introduce a number of additional graph distance measures.

4.3.1 Using Maximum Common Subgraph

Graph distance measures using only graph element parameters, but no labels, provide a measure of difference in graph topology. One such measure relies on the maximum common subgraph (MCS). A distance metric has been defined based on the determination of the maximum common subgraph of two graphs g and g' [30]:

$$d(g, g') = 1 - \frac{|\mathrm{mcs}(g, g')|}{\max\{|g|, |g'|\}} , \tag{4.1}$$

where $\mathrm{mcs}(g, g')$ denotes the maximum common subgraph of g and g', and $|g|$ denotes the number of vertices in the graph g. The number of edges can also be used as $|g|$, or any other measure of problem size in the denominator of equation (4.1) [158, 182].

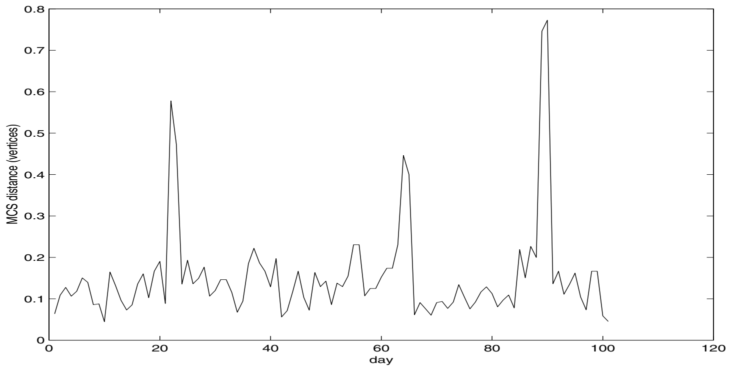


Fig. 4.1. Maximum common subgraph distance (vertices).

In applying this maximum common subgraph distance measure to the network data, subsequent days' logical connectivity between business domains is represented as a series of graphs that are successively compared using the distance metric. The MCS measure produces results depicting the *relative* degree of change experienced. This is shown in Figure 4.1, where $|g| = |V|$, and Figure 4.2, where $|g| = |E|$. In both plots it is clear that there are three prominent events (occurring on days 22, 64, and 89). The three events of greatest change are highlighted more using $|g| = |E|$ as a measure of graph size. This is useful because it is the changes in relationships between logical nodes that are of particular interest to network operators. Verification that these peaks do represent significant change is discussed later.

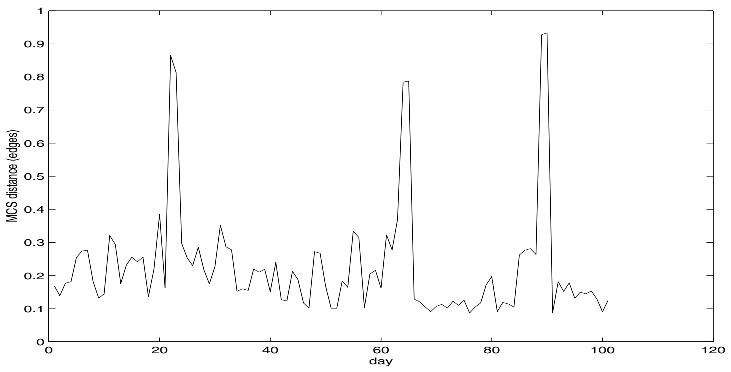


Fig. 4.2. Maximum common subgraph distance (edges).

4.3.2 Using Graph Edit Distance

Another graph distance measure suited to measuring changes in graph topology is graph edit distance (see Chapter 3). Graph edit distance (ged) evaluates the sequence of edit operations required to modify an input graph such that it becomes isomorphic to some reference graph. This can include the possible insertion and deletion of edges and vertices, in addition to edge label substitutions. Generally, ged algorithms assign costs to each of the edit operations and use efficient tree search techniques to identify the sequence of edit operations resulting in the lowest total edit cost [27, 131]. The resultant lowest total edit cost is a measure of the distance between the two graphs.

In general, a unique sequence of edit operations does not exist due to the occurrence of multiple possible vertex mappings. The ged algorithms are required to search for the edit sequence that results in a minimum edit cost. However, in communications performance monitoring, vertex-label value substitution is not a valid edit operation because vertex labels reference unique physical or logical nodes within a network. As a result, the combinatorial search reduces to the simple identification of elements (vertices and edges) inserted or deleted from one graph g to produce the other graph g'. The implementation requires linear time in the size of the problem, as described in Chapter 3.

If the cost associated with the insertion or deletion of individual elements is one, and edge-weight value substitution is not considered (i.e., we consider the topology only), the edit sequence cost becomes the difference between the total number of elements in both graphs and all graph elements in common.

Using the above cost function, let the graph $g = (V, E, \alpha, \beta)$ represent the communication network's connectivity observed over a single business day, and let $g' = (V', E', \alpha', \beta')$ describe the same network's connectivity over a subsequent business day. The *graph edit distance* $d(g, g')$ describing topological change experienced by the network over successive days then becomes[1]

$$d(g, g') = |V| + |V'| - 2|V \cap V'| + |E| + |E'| - 2|E \cap E'| \,. \tag{4.2}$$

Clearly the edit distance, as a measure of topology change, increases with increasing degree of change experienced by the network over successive time intervals. Edit distance $d(g, g')$ is bounded below by $d(g, g') = 0$ when g and g' are isomorphic (i.e., there is no change), and above by $d(g, g') = |V| + |V'| + |E| + |E'|$ when $g \cap g' = \emptyset$, the case in which the networks are completely different. Cost functions can also be designed to place greater significance on the more important vertices or edges within the graph representation of the network. It is interesting to note that this measure, ged, is related to the previous maximum common subgraph distance metric given in equation (4.1) in [20].

Results plotted in Figure 4.3 show that the edit distance produces peaks (indicating significant change) that correlate with the maximum common subgraph distance measures, in addition to indicating further possible events of interest. Note that edit

[1] Note that equation (4.2) is a simplification of the formula derived in Lemma 3.6 because no edge label substitutions are considered here.

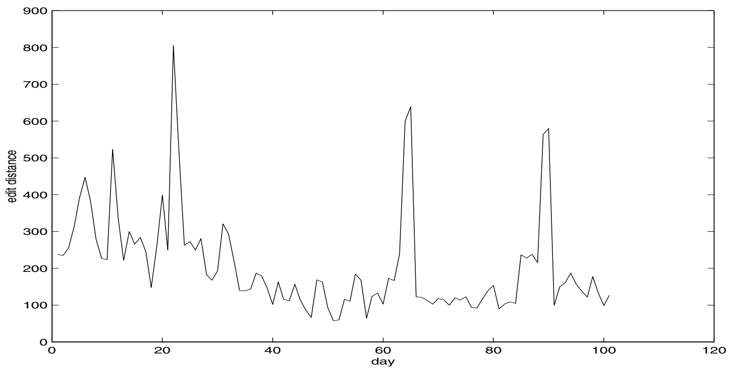


Fig. 4.3. Edit distance.

distance measures the *absolute change* between consecutive network observations in the time series, while the MCS approach is relative. This absolute measure is useful in the detection of large increases (or decreases) in graph size. Also, both the MCS and edit distance plots indicate a reasonable amount of day-to-day dynamic logical network behavior over this physical link.

The three prominent peaks occurring in the MCS and edit distance plots were verified through the visual examination of the corresponding graphs using a graph visualization tool. Comparison of graphs over successive days generally showed a large degree of edge and vertex consistency. This is reasonable because one would expect a fair degree of consistency in communications between business groups in a large corporate data network. While consistency was evident, there were also typically many changes to edges and vertices, indicating a dynamic nature in logical connectivity from one day to the next. This shows up in the general day-to-day "background" change in the three plots.

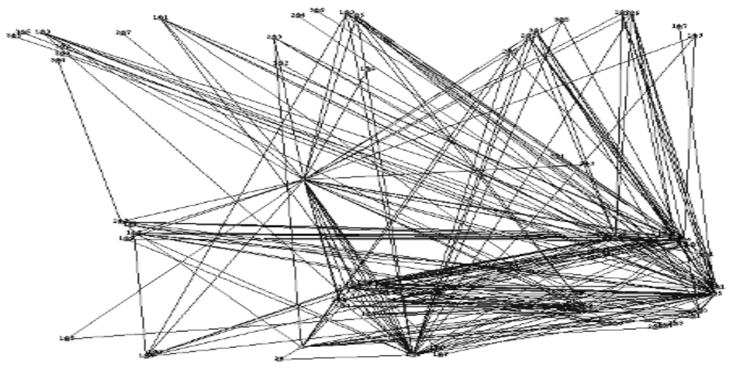


Fig. 4.4. Logical connectivity over physical link before change.

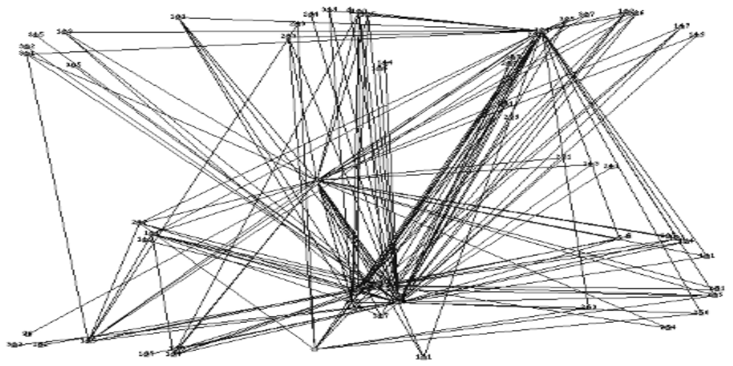


Fig. 4.5. Logical connectivity over physical link after change.

In the transition from day 63 to day 64, a significant amount of change was measured using both MCS and edit distance measures. To provide an example of the degree of change measured, the logical connectivity between business domains on days 63 and 64 is shown in Figures 4.4 and 4.5 respectively. Note that all node positions are maintained the same in both figures. There is clearly a significant change in connectivity, in the lower right region of the graphs, between these two days. Such changes were not evident in the comparison of other adjacent graphs in the time series with lower distance scores.

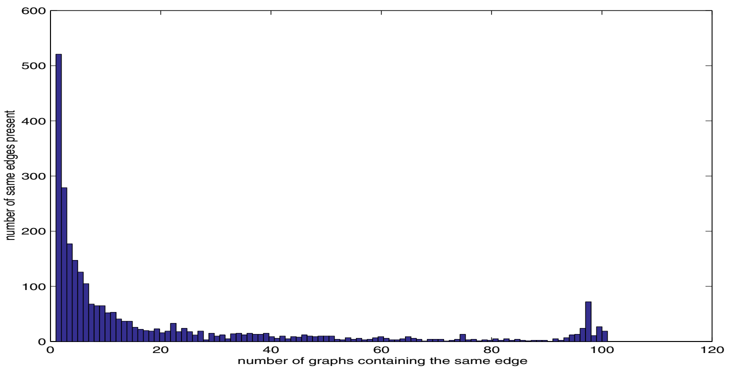


Fig. 4.6. Distribution of edge presence in the series of graphs.

Figure 4.6 shows the distribution of edge occurrences in the time series of graphs over the 102 days. From this figure, a large number of edges appear in a reasonably small ($\leq$ 10) number of graphs; these edges are the cause of the large amount of distance variation. Also, there are quite a few edges that occur consistently in a large number of the graphs; 99 graphs in the time series have approximately 100 edges in common. This further explains the three significant events observed in the previous graph distance plots, in which many of these 100 consistent edges were found missing from the graphs

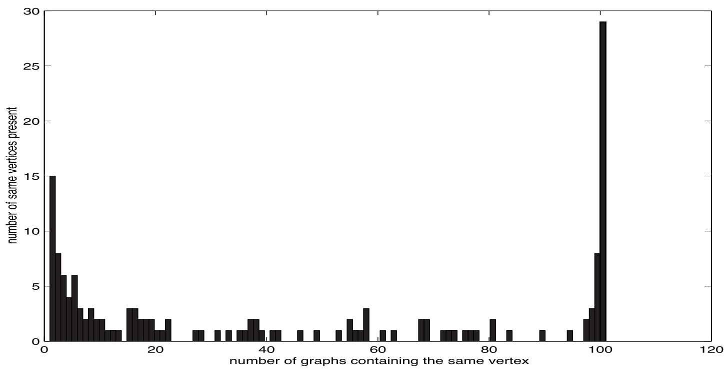


Fig. 4.7. Distribution of vertex presence in the series of graphs.

of those three particular days. The distribution of vertex presence in the series of graphs is shown in Figure 4.7. Again there is a large number of dynamic vertices and also many consistent vertices in the series. Almost 30 vertices occur in all graphs of the time series. This shows a consistent group of business domains communicating daily.

4.4 Traffic-Based Distance Measures

Traffic loading in a network is a prime concern of network performance management, so the ability to automatically detect abnormal change in traffic patterns is very useful. The expression for graph edit distance given in equation (4.2) provides a measure of difference between two graphs in terms of topological connectivity. Communications traffic is represented in the graphs by edge-weight and can be incorporated into a ged algorithm using an edge-weight substitution cost, although it is difficult to design a suitable cost function that is sensitive to both topology changes and traffic variations. One alternative is to quantize traffic into integer values and insert multiple edges into the graph between adjacent vertices, with the number of edges corresponding to the integer value of traffic. This effectively converts the problem into one of topology change. However, this too presents problems in combining connectivity changes with traffic variations because large fluctuations in edge-weight value have different implications for communications than does the insertion or deletion of nodes in a network. In this section we introduce two alternative measures based on difference in edge-weight values and graph spectrum.

4.4.1 Differences in Edge-Weight Values

Measures of graph distance based on differences in edge-weight values for two graphs $g = (V, E, \alpha, \beta)$ and $g' = (V', E', \alpha', \beta')$ have been proposed [139]. Using this distance measure it is possible to assess relative traffic variations by summing the differences in edge-weight value over all edges in the two graphs. The distance measure is defined as

$$d(g, g') = \sum_{u,v \in V} \frac{|\beta(u, v) - \beta'(u, v)|}{\max\{\beta(u, v), \beta'(u, v)\}} \,. \tag{4.3}$$

Other expressions similar to equation (4.3) appear in [139, 174].

Dividing $d(g, g')$ in equation (4.3) by the total number of edges in the double summation (i.e., by $|E \cup E'|$) provides a normalized relative measure of traffic variation over the entire network. All edges in both graphs can be included by substituting an edge-weight value of zero for edges that exist only in one of the two graphs g and g'. As a result, this distance measure includes the effects of topological change by including edges that have been inserted or deleted. It is also possible to include only those edges belonging to the maximum common subgraph of g and g', thus enabling the detection of traffic variations between users maintaining communications over two adjacent time intervals.

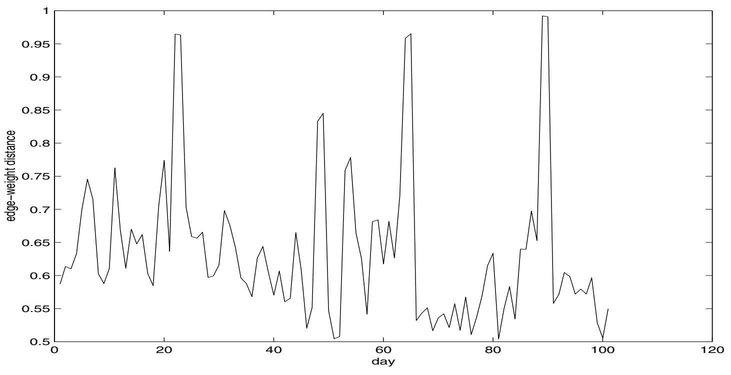


Fig. 4.8. Edge-weight distance.

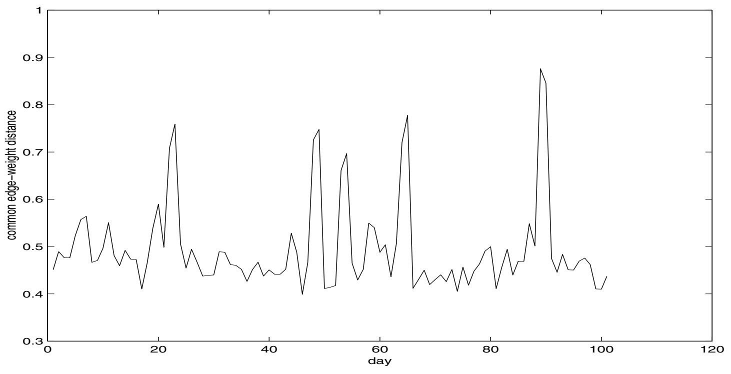


Fig. 4.9. Edge-weight distance (MCS).

Figure 4.8 shows the comparative plot of edge-weight distance derived from the data network observations represented as a time series of graphs. Three of the peaks, indicating increased network change, correlate with previous plots showing changes in connectivity. Clearly these logical connectivity changes involve relatively large amounts of traffic, as illustrated in Figure 4.8. Two peaks at days 48 and 53 indicate large changes in traffic distribution, between business domains (vertices), without greatly affecting overall connectivity. This becomes apparent by comparing Figure 4.8 with say Figure 4.3 for days 48 and 53. Figure 4.9 plots the same edge-weight distance but in this case only with edges belonging to the maximum common subgraph between the graphs being compared. This plot is useful because it shows that there was a reasonable amount of change in traffic on five of the days independent of any changes in logical connectivity. By considering together the plots of connectivity (Figures 4.1–4.3), total traffic (Figure 4.8), and traffic variations in the maximum common subgraph (Figure 4.9), it is possible to make judgements regarding the type of network changes occurring.

4.4.2 Analysis of Graph Spectra

Another approach to the measurement of change in a time series of graphs is through the analysis of graph spectra. Algebraic aspects of spectral graph theory are useful in the analysis of graphs [153, 174]. There are several ways of associating matrix spectra (sets of all eigenvalues) with a given weighted graph g. The most obvious way is to investigate the structure of a graph g by analyzing the spectrum of its adjacency matrix $\mathbf{A}_g$. Note that $\mathbf{A}_g$ is not a symmetric matrix for directed graphs because weight may not be a symmetric function.

For a given ordering of the set of n vertices V in a graph $g = (V, E, \alpha, \beta)$, one can investigate the *spectrum* $\sigma(g) = \{\lambda_1, \lambda_2, \dots, \lambda_n\}$, where λ_i are the eigenvalues of the weighted *adjacency matrix* $\mathbf{A}_g$. Obviously, $\sigma(g)$ does not depend on the ordering of V.

A different approach to graph spectra for undirected graphs [46] investigates the eigenvalues of the *Laplace matrix* $\mathbf{L}_g = \mathbf{D}_g - \mathbf{A}_g$, where the *degree matrix* $\mathbf{D}_g$ of graph g is defined as $\mathbf{D}_g = \operatorname{diag}\{\sum_{v \in V} \beta(u, v) \mid u \in V_g\}$. Note that in the unweighted case, diagonal elements of $\mathbf{D}_g$ are simply the vertex degrees of indexed vertices V. In the case of directed graphs, the Laplacian is defined as $\mathbf{L}_g = \mathbf{D}_g - (\mathbf{A}_g + \mathbf{A}_g^T)$ (in order to ensure that $\sigma(\mathbf{L}_g) \subseteq \{0\} \cup \mathbb{R}^+$). Laplacian spectra of graphs have many applications in graph partitions, isoperimetric problems, semidefinite programming, random walks on graphs, and infinite graphs [46].

The relationship between graph spectra and graph distance measures can be established using an eigenvalue interpretation: given two weighted graphs g and g' with respective spectra $\sigma(\mathbf{A}_g) = \{\lambda_1, \lambda_2, \dots, \lambda_{n_1}\}$ and $\sigma(\mathbf{A}_{g'}) = \{\mu_1, \mu_2, \dots, \mu_{n_2}\}$, the k largest positive eigenvalues are incorporated into the graph distance measure [153] as

$$d(g, g') = \sqrt{\frac{\sum_{i=1}^{k} (\lambda_i - \mu_i)^2}{\min\left\{\sum_{i=1}^{k} \lambda_i^2 , \sum_{j=1}^{k} \mu_j^2\right\}}} , \tag{4.4}$$

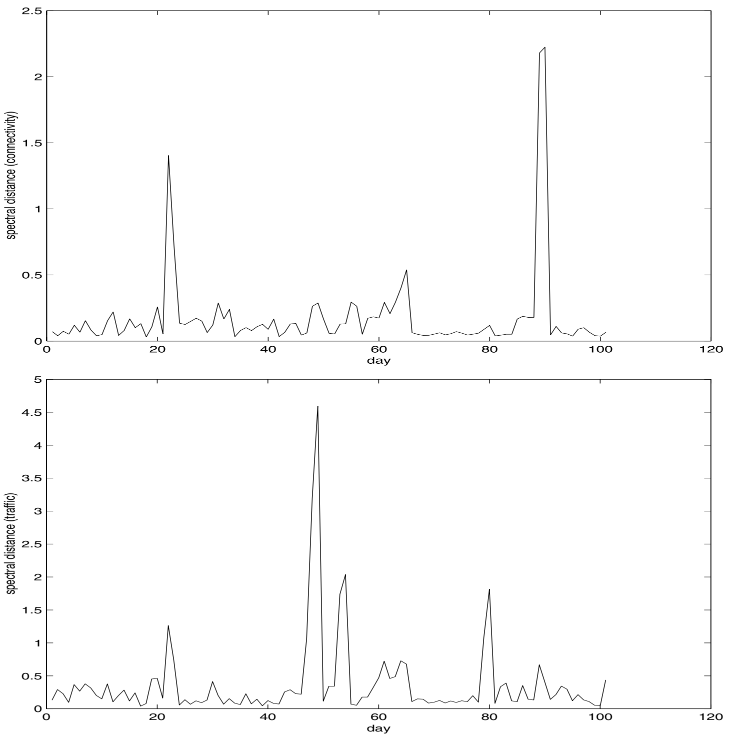


Fig. 4.10. Spectral distance (connectivity and traffic).

where k is an arbitrary summation limit. Empirical studies in pattern recognition and image analysis show that $k \approx 20$ is a good choice [153, 158]. Notice that similar approaches to distance measures can be applied to the case of Laplacian spectra.

Figure 4.10 shows the outcomes of connectivity and traffic (weighted graph) spectral distances respectively, applied to the time series of graphs derived from the network data. In spite of the less-intuitive interpretation of these graph distance measures, there exists reasonable correlation with the peaks of other distance measures and far less sensitivity to daily variations using this approach. However, at this time it is not clear why the traffic peak at day 80 is so prominent in the lower part of Figure 4.10.

4.5 Measures Using Graph Structure

Instead of measuring change by concentrating on network elements, as in the cases for graph edit distance and maximum common subgraph, better indicators of abnormal change could perhaps be obtained by basing comparisons on specific structures or graph properties within the graphs. Such measures are likely to be less sensitive to minor variations of individual elements but highlight network changes of greater significance.

One possible approach is to use various path sets for the definition of new graph measures. For specified vertices (network terminals) $u, v \in V_g$, one can consider the family $P_k^g(u, v)$ of all paths (in g) of length k connecting u and v. This can be generalized to the collection P_k^g of all k-long paths in graph g (i.e., the *distance-k path set* of g). One could also replace the k-length paths with paths containing the *minimal* number of edges, or with *shortest* paths satisfying certain vertex- and/or edge-weight minimality. Let $P^g(u, v) = \bigcup_{k \geq 2} P_k^g(u, v)$, and similarly, by $P^g = \bigcup_{k \geq 2} P_k^g$ denote the sets of *all* paths of length greater than or equal to 2 joining u and v, or joining any two vertices, respectively.

In communication networks, it is common for information to be passed from a source node to a destination node along a path via intermediate nodes. Therefore, if an edge is deleted from a graph, and many paths contain this edge, it will have a greater impact on communications than an edge that affects only one path. It therefore seems reasonable that in the context of communications, graph distance measures based on the number of paths containing a *specific edge(s)* should be more sensitive to network changes of interest.

For a chosen (nonempty) subset of edges $\widehat{E} \subseteq E$, a new graph $g'(\widehat{E}) = (V', E', \alpha', \beta')$ can be generated from $g = (V, E, \alpha, \beta)$ such that only those edges in g are retained that are within paths p in $P^g(\cdot, \cdot)$ (or in P^g) containing edges in $\widehat{E}$. The graph $g(\widehat{E})$ is constructed using the following conditions:

1. $V' = V$.
2. An edge e is in E' if and only if there exists a path $p \in P_k^g$ such that $e \in p$ and p contains at least one edge in $\widehat{E}$.
3. $\alpha' = \alpha$.
4. The edge-weight values in $G_{\widehat{g}}$ denote the number of paths in P^g containing that edge and at least one edge in $\widehat{E}$.

Edge attributes in $g(\widehat{E})$ provide an indication of edge significance because edges contained within many paths have high associated edge-weight value and are likely to affect communications connectivity between more vertices.

Graphs $g_1'(\cdot)$ and $g_2'(\cdot)$ can be constructed from graphs g_1 and g_2 respectively and compared using equation (4.3) as the basis for measuring graph distance. There are some obvious choices of respective sets $\widehat{E}$ as subsets of E_1 and E_2; either $\widehat{E} = E_1$ and $\widehat{E} = E_2$ respectively, or $\widehat{E}$ can consist of edges belonging to $\text{mcs}(g_1, g_2)$. Furthermore, $\widehat{E}$ can comprise edges of the core backbone of the communication network or important logical links. In the case that there exist disjoint edge sets of g_1 and g_2, both graphs are treated as completely different. The resulting distance measure should be more sensitive to change that potentially impacts a greater number of nodes or users within the network. Analysis using the sampled network traffic showed no improvement in clearly identifying further change events over the previous methods. However, this approach is worth being investigated further due to the potential advantages it offers.

4.5.1 Graphs Denoting 2-hop Distance

A variant of the above path-based approach is to consider a graph generated from paths of length two only; this shows the 2-hop connectedness of the given graph. As a result, adjacent vertices in this derived graph denote vertices in the original graph that are connected via a common neighbor or neighbors. Any change to individual edges of these new graphs (describing 2-hop connectedness) indicates significantly greater impact on communications topology than change in individual edges associated with the original graphs.

For a given graph $g = (V, E, \alpha, \beta)$, the graph $g_{2\text{-hop}}$ is generated as $g'(\widehat{E})$ using either $P_2^g(u, v)$ or P_2^g for path sets P, and E for $\widehat{E}$. Furthermore, parallel reductions are performed on all edges of $g_{2\text{-hop}}$, and the new graph is unweighted. If graphs $g_{1,2\text{-hop}}$ and $g_{2,2\text{-hop}}$ are constructed from two given graphs g_1 and g_2 respectively, and compared using equations (4.1) or (4.2), the resulting graph distance will be more sensitive to the loss (or gain) of connectivity through common neighbors. The loss (or gain) of individual edges within the given graphs is likely to have less effect on the resulting graph distance measure. This approach attempts to filter minor variations that would otherwise contribute unwanted variance to graph distance measurements.

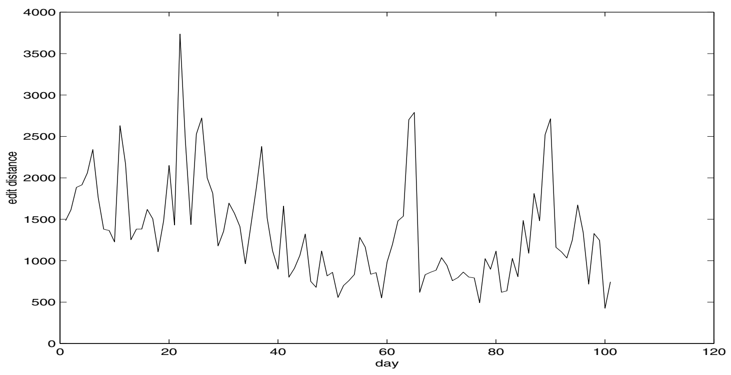


Fig. 4.11. Edit distance using 2-hop connectedness.

The 2-hop edit distance has been applied to the time series of network data, producing a result showing correlation of significant events with previous distance measures (Figure 4.11). In addition, this distance measure highlights a number of secondary level events (in terms of change magnitude) not obvious with earlier results. However, clear evidence demonstrating the significance of these events has not yet been obtained.

4.6 Identifying Regions of Change

The various graph distance measures previously described enable detection of relative change as a network, represented by a series of graphs, evolves over time. While global similarity measures such as these are useful in determining when abnormal events have occurred that might warrant closer examination, these measures do not provide information describing where in the network greatest change has occurred. For this latter problem an indication of change distribution within the network is required.

4.6.1 Symmetric Difference

One approach to locating regions in the network most affected by topology change is to rank vertices in the order they experienced insertions or deletions of incident edges during the transition from one time interval to the next. Consider two graphs g and g' representing network communications over two successive time intervals, and assume that the graph distance between g and g' is deemed to be significant. Of particular interest is the distribution of the change during the transition from g to g'. The following method ranks all vertices $V \cup V'$ within the two graphs in ascending order of the number of network topology change events experienced by individual vertices.

Differences between graphs g and g' can be described by a *change matrix* $\mathbf{C} = [c_{uv}]$ that indicates where edges have been deleted from g or inserted into g'. This matrix $\mathbf{C}$ has a row and column for every vertex contained within the two graphs. An edge deleted or inserted in transition from one graph to the other is represented in the matrix by a corresponding row column entry $c_{uv} = 1$. Indices u and v denote the respective originator and destination vertices of the deleted or inserted directed edge. Any edges (u, v) that remain incident to the same pair of vertices in both g and g' result in the corresponding entry $c_{uv} = 0$, indicating that no change has occurred. All other entries in $\mathbf{C}$ equal 0. This matrix $\mathbf{C}$ essentially describes the symmetric difference between graphs g and g', where the *symmetric difference* $g \triangle g'$ is the graph containing vertices $V \cup V'$ whose edges appear in exactly one of either graph g or g' [157, 188].

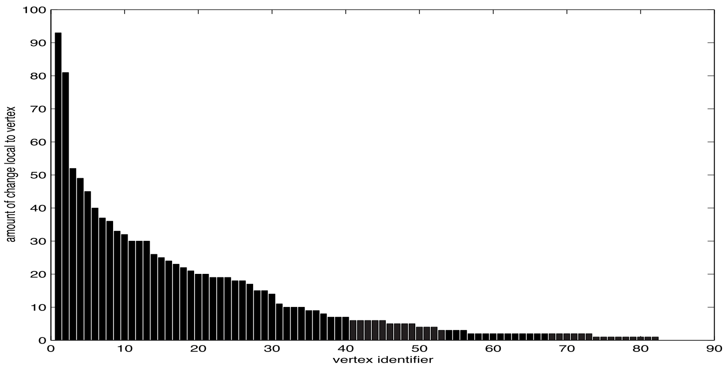


Fig. 4.12. Local change experienced by vertices.

Respective row and column sums of $\mathbf{C}$ indicate the amount of change experienced locally by the corresponding vertex. The amount of change experienced by each vertex can be plotted in descending order to show the distribution of local change occurring in the transition from g to g'. This is illustrated in Figure 4.12 derived from the network data for change occurring between days 63 and 64. Vertices contributing most to the network change can be readily identified from the figure.

This approach could be modified to identify those vertices experiencing greatest variations in traffic along incident edges by substituting the entries c_{uv} in $\mathbf{C}$ with relative differences in edge-weight value between g and g', for example

$$c_{uv} = \frac{|\beta(u,v) - \beta'(u,v)|}{\max\{\beta(u,v), \beta'(u,v)\}} ,$$

where $(u, v) \in E \cup E'$.

4.6.2 Vertex Neighborhoods

An alternative to the graph-symmetric difference approach just described is to measure graph distances between corresponding vertex neighborhood subgraphs. This technique will produce a vector of graph distance measures describing the differences between two graphs g_1 and g_2. Each vector coordinate indicates the distance between g_1 and g_2 from the perspective of an individual vertex and its adjacent vertices, essentially providing a measure of change for a local region. The *neighborhood subgraph* of a vertex u in $g_1 = (V_1, E_1, \alpha_1, \beta_1)$ is the subgraph $g_1'(u) = (V_1'(u), E_1'(u), \alpha_1', \beta_1')$, where $E_1'(u) = E_1 \cap (N_1[u] \times N_1[u])$ is the set of incident edges between adjacent vertices in $N_1[u]$. Vertex- and edge-weight functions in $g_1'(u)$ are α_1 and β_1, restricted to $N_1[u]$ and $E_1(u)$ respectively.

Successive graphs in the time series of networks can be compared using this neighborhood approach with the corresponding neighborhoods' graph distances calculated using equations (4.1), (4.2), (4.3), or (4.4). A vertex that exists only in one of the two graphs g_1 and g_2 has its graph neighborhood compared with the *empty neighborhood*, where $N[u] = E(u) = \emptyset$. The resulting neighborhood graph distances are stored in a *distance vector* $\mathbf{d}$, defined as

$$\mathbf{d} = [d(g_1'(u), g_2'(u))] ,$$

for all $u \in (V_1 \cup V_2)$, and where $d(\cdot, \cdot)$ is a particular graph distance measure. Those coordinates contained within the neighborhood distance vector $\mathbf{d}$ with higher subgraph distance measures correspond to the vertices u that experienced greatest change in their respective local region as the network transitioned from one time interval to the next. The distance vector coordinates can be ordered to rank vertices by the degree to which they experienced local change.

The vertex neighborhoods defined above describe single-hop (1-hop) neighborhoods that include only the adjacent vertices to a given vertex u. It may be useful to consider neighborhoods of 2-hops, whereby vertices adjacent to those in $N[u]$ are also included in the neighborhood subgraph (together with incident edges). Of course this can be extended to the general case for k-hops.

4.7 Conclusions

This chapter has examined several techniques that can be used to measure the degree of change occurring within a data network as it evolves over time. Communication transactions collected by a network management system between logical network nodes occurring over periodic time intervals are represented as a series of weighted graphs. Graph distance measures are used to assess the changes in communications between user groups (nodes), over successive time intervals, to focus a network operator's attention on the time and regions within the logical network where abnormal events occur.

Identifying what is a "significant event" or what is the network's "normal" operational behavior is difficult. However, drawing attention to abnormal events, when compared with previous network observations, appears feasible and potentially very useful. It relieves network operators from the need to continually monitor networks if connectivity and traffic patterns can be shown to be similar to the activity over previous time periods. Alternatively, if traffic volumes are seen to abruptly increase over the physical link, network operators are able to more readily identify the individual user groups contributing to this increase in aggregate traffic.

Both maximum common subgraph and edit distance measures identified significant changes in logical connectivity using relative and absolute measures of change respectively. The more significant events were verified by comparing graph visualizations of days around the events and on other "normal" days. The edge-weight distance measure highlighted large variations in logical traffic distributions on two days in particular that were not affected by large changes in connectivity. These results correlate with distance measures based on analysis of the graph spectra. The use of vertex neighborhood distance vectors and graph-symmetric difference assists with identifying those vertices contributing most to network change.

5

Median Graphs for Abnormal Change Detection

5.1 Introduction

In Chapter 4, abnormal network behavior was detected based on the distance of a pair of graphs. In this chapter we derive more general procedures that use more than just two graphs. Considering a larger number of graphs, it can be expected that abnormal event detection procedures will exhibit more stability and robustness against random fluctuations and noise in the underlying network.

The procedures derived in this chapter will be based on the median of a set, or a sequence of graphs as introduced in Chapter 4. Intuitively speaking, the *median* of a sequence of graphs $S = (g_1, \ldots, g_n)$ is a graph that represents the given g_i's in the best possible manner. Using any of the graph distance measures introduced in Chapter 4, the median of a sequence of graphs S is defined as a graph that minimizes the sum of all edit distances to all members of sequence S. Formally, let U be the family of all graphs that can be constructed using labels from L_V for vertices and real numbers for edges. Then $\overline{g}$ is a *median graph* of the sequence $S = (g_1, \ldots, g_n)$ if

$$\overline{g} = \arg\min_{g \in U} \sum_{i=1}^{n} d(g, g_i) . \tag{5.1}$$

If we constrain $\overline{g}$ to be a member of S then the graph that satisfies equation (5.1) is called the *set median* of S. In a general context where node labels are not unique a set median is usually easier to compute than a median graph. However, in the context of the present chapter, where we deal with uniquely labeled graphs, we will exclusively focus on median graph computation and its application to abnormal change detection. It is to be noted that median graphs need not be unique.

Here we want to point out that the term "median graph" is used because the median x of an ordered sequence of real numbers $(x_1, \ldots, x_n)$, which is defined as

$$\overline{x} = \text{median}(x_1, \ldots, x_n) = \begin{cases} x_{n/2}, & \text{if } n \text{ is even} , \\ x_{\lceil n/2 \rceil}, & \text{if } n \text{ is odd} , \end{cases} \tag{5.2}$$

has a similar property to the median of a sequence of graphs: it minimizes the sum of distances to all elements of the given sequence, i.e., it minimizes the expression $\sum_{i=1}^{n} |\overline{x} - x_i|$.

In Chapter 3, an efficient procedure for median graph computation was introduced. This procedure uses the edit distance of graphs and assumes particular costs of the underlying edit operations. Each of the edit operations node deletion, node insertion, edge deletion, edge insertion, and edge substitution has a cost equal to one. Node label substitution is not admissible and has infinite cost. We will use the notation $d_1(g_1, g_2)$ to refer to this kind of graph edit distance. In Section 5.2 we introduce a second graph edit distance, called $d_2(d_1, g_2)$, and describe a procedure for median graph computation based on $d_2(g_1, g_2)$. The cost function underlying $d_2(g_1, g_2)$ is more general than the cost function underlying $d_1(g_1, g_2)$ in that it takes differences in edge weight into account. In Section 5.3 we derive procedures for abnormal event detection in telecommunication networks using median graphs. Experimental results are reported in Section 5.4 and conclusions are drawn in Section 5.5.

5.2 Median Graph for the Generalized Graph Distance Measure d_2

In this section we introduce a graph edit distance measure, called $d_2(g_1, g_2)$, which also takes edge weight into account. We focus our attention on graphs with unique node labels and drop the node labeling function α. Let $g_1 = (V_1, E_1, \beta_1)$ and $g_2 = (V_2, E_2, \beta_2)$ denote two graphs. Under measure d_2 we seek the minimum cost sequence of edit operations that transform graph g_1 into g_2. Each vertex insertion and deletion has a cost $c > 0$. Vertex label substitutions are not admissible and have infinite cost. The cost of changing weight $\beta_1(e)$ on edge $e \in E_1$ into $\beta_2(e)$ on $e \in E_2$ is defined as $|\beta_1(e) - \beta_2(e)|$. To simplify our notation we let our graphs be completely connected (i.e., there is an edge $e \in E_1$ between any two vertices in g_1 and an edge $e \in E_2$ between any two vertices in g_2) and assign a weight equal to zero to edge $e \in E_1$ ($e \in E_2$) if this edge does not exist in $g_1(g_2)$. Hence substitution of edge weights, edge deletions, and edge insertions can be treated uniformly. Note that the deletion of an edge $e \in E_1$ with weight $\beta_1(e)$ has a cost equal to $\beta_1(e)$. Similarly, the insertion of an edge $e \in E_2$ has the weight of that edge $\beta_2(e)$ assigned as its cost. Consquently, the graph edit distance under this cost function becomes

$$\begin{aligned} d_2(g_1, g_2) = c[|V_1| + |V_2| - 2|V_1 \cap V_2|] + \sum_{e \in E_1 \cap E_2} |\beta_1(e) - \beta_2(e)| \\ + \sum_{e \in E_1 \setminus (E_1 \cap E_2)} \beta_1(e) + \sum_{e \in E_2 \setminus (E_1 \cap E_2)} \beta_2(e) \,. \end{aligned}$$

The constant c is a parameter that allows us to weight the importance of a node deletion or insertion relative to the weight changes on the edges.

We start from $S = (g_1, \ldots, g_n)$, $g = (V, E, \beta)$ with $V = \bigcup_{i=1}^{n} V_i$ and $E = \bigcup_{i=1}^{n} E_i$, and let $\gamma(u)$ be the same as in Chapter 3, i.e., $\gamma(u)$ represents the number of occurrences of node u in sequence S.

Let graph $\widehat{g} = (\widehat{V}, \widehat{E}, \widehat{\beta})$ be defined as follows:

$$\begin{aligned}\widehat{V} &= \{u \mid u \in V \text{ and } \gamma(u) > n/2\}\,,\\ \widehat{E} &= \{(u, v) \mid u, v \in \widehat{V}\}\,,\\ \widehat{\beta}(u, v) &= \text{median}\{\beta_i(u, v) \mid i = 1, \ldots, n\}\,.\end{aligned}$$

Theorem 5.1. *Graph $\widehat{g} = (\widehat{V}, \widehat{E}, \widehat{\beta})$ is a median of set S under graph distance measure d_2.*

Proof. Similarly to the proof of Lemma 3.7 we observe that the set of nodes potentially included in $\widehat{V}$ is bounded from above and below by $\bigcup_{i=1}^{n} V_i$ and $\emptyset$, respectively. The quantity to be minimized is

$$\Delta = c\left[n|\widehat{V}| + \sum_{i=1}^{n}|V_i| - 2\sum_{i=1}^{n}|\widehat{V} \cap V_i|\right] + \sum_{(u,v)\in E}\sum\nolimits_{i=1}^{n}|\widehat{\beta}(u, v) - \beta_i(u, v)|\,,$$

which is equivalent to minimizing $\delta_{\text{nodes}} + \delta_{\text{edges}}$, where

$$\delta_{\text{nodes}} = c\left[n|\widehat{V}| - 2\sum_{i=1}^{n}|\widehat{V} \cap V_i|\right]$$

and

$$\delta_{\text{edges}} = \sum_{(u,v)\in E}\sum\nolimits_{i=1}^{n}|\widehat{\beta}(u, v) - \beta_i(u, v)|\,.$$

Clearly, δ_{nodes} is minimized by the same procedure used in Lemma 3.7, i.e., we include a node $u \in V$ in $\widehat{V}$ if and only if $\gamma(u) > n/2$. For the minimization of δ_{edges} we use the property of the median of an ordered sequence of real numbers cited in Section 5.1. That is, δ_{edges} is minimized by assigning to each edge $(u, v) \in \widehat{E}$ the median of the weights of the edge (u, v) in $E_1, \ldots, E_n$. Formally, $\widehat{\beta}(u, v) = \text{median}(\beta_{i_1}(u, v), \ldots, \beta_{i_n}(u, v))$, where $\beta_{i_1}(u, v), \ldots, \beta_{i_n}(u, v)$ is the ordered sequence of $\beta_1(u, v), \ldots, \beta_n(u, v)$.

Similarly to Lemma 3.7, the values of δ_{nodes} and δ_{edges} are not independent of each other, because the exclusion of a node u in $\widehat{V}$ implies the exclusion of all edges (u, v) and (v, u) in $\widehat{E}$, which means $\widehat{\beta}(u, v) = 0$ or $\widehat{\beta}(v, u) = 0$ for those edges. We observe that $\text{median}(\beta_i(u, v) \mid i = 1, \ldots, n) = 0$ and $\text{median}(\beta_i(v, u) \mid i = 1, \ldots, n) = 0$ if $\gamma(u) \leqslant n/2$ or $\gamma(v) \leqslant n/2$. Hence none of the edges incident to u will ever be a candidate for inclusion in $\widehat{E}$ during the minimization of δ_{edges}. This concludes the proof.

Comparing the median graph construction procedure described in Lemma 3.7 with the one introduced in this chapter, we notice that the former is a special case of the latter, constraining edge weights to assume only binary values. Edge weight zero (or one) indicates the absence (or presence) of an edge. Including an edge (u, v) in the

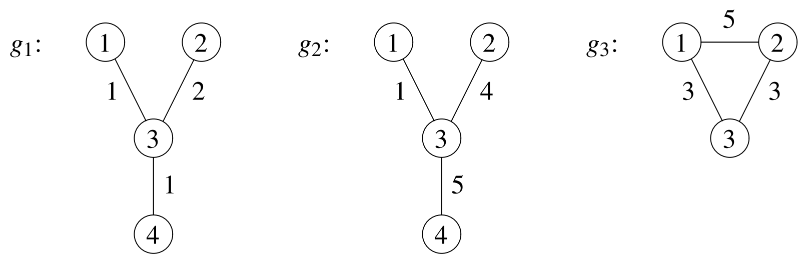


Fig. 5.1. Three graphs (g_1, g_2, g_3).

median graph $\overline{g}$ because it occurs in more than $n/2$ of the given graphs is equivalent to labeling that edge in $\widehat{g}$ with the median of the weights assigned to it in the given graphs.

The median of a set of numbers according to equation (5.2) is unique. Hence in constructing a median graph under graph distance measure d_2, there will be no ambiguity in edge weight. But inclusion of a node in the median graph is in general not unique.

We conclude this section with an example of median graph under graph distance d_2. Three different graphs g_1, g_2, and g_3 are shown in Figure 5.1. Their median is unique and is displayed in Figure 5.2. Moreover, we observe that $\Delta = c + 14$.

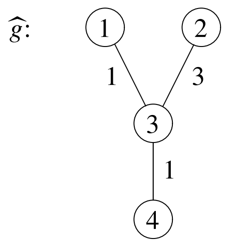


Fig. 5.2. The median of (g_1, g_2, g_3) in Figure 5.1 using d_2.

5.3 Median Graphs and Abnormal Change Detection in Data Networks

In Chapter 4 several graph distance measures are applied to detect abnormal change in telecommunication networks. However, these measures are applied only to consecutive graphs in a time series of graphs $g_1, \ldots, g_n$. That is, values of $d(g_{i-1}, g_i)$ are computed for $i = 2, \ldots, n$, and the change from time $i - 1$ to i is classified as abnormal if $d(g_{i-1}, g_i)$ is larger than a certain threshold. This approach has shown good results. Nevertheless, it can be argued that measuring network change only between consecutive points of time is potentially vulnerable to noise, i.e., the random appearance or disappearance of some nodes together with some random fluctuation of the edge weights may lead to a graph distance larger than the chosen threshold, though these changes are not really significant.

One expects that a more robust change detection procedure would be obtained through the use of median graphs. In statistical signal processing the median filter is widely used for removing impulsive noise. A median filter is computed by sliding a window of length L over data values in a time series. At each step the output to this process is the median of values within the window. This process is also termed the *running median* [3]. In the following we discuss four different approaches to abnormal change detection that utilize median filters. All these approaches assume that a time series of graphs $(g_1, \ldots, g_n, g_{n+1}, \ldots)$ is given. The median graph of a subsequence of these graphs can be computed using either graph distance measure d_1 or d_2.

5.3.1 Median vs. Single Graph, Adjacent in Time (`msa`)

Given the time series of graphs, we compute the median graph in a window of length L, where L is a parameter that is to be specified by the user, depending on the underlying application. Let $\widetilde{g}_n$ be the median of the sequence $(g_{n-L+1}, \ldots, g_n)$. Then $d(\widetilde{g}_n, g_{n+1})$ can be used to measure the abnormal network change. We classify the change between g_n and g_{n+1} as *abnormal* if $d(\widetilde{g}_n, g_{n+1})$ is larger than some threshold.

Increased robustness can be expected if we take the average deviation φ of graphs $(g_{n-L+1}, \ldots, g_n)$ into account. We compute

$$\varphi = \frac{1}{L} \sum_{i=n-L+1}^{n} d(\widetilde{g}_n, g_i)\,, \tag{5.3}$$

and classify the change between g_n and g_{n+1} as abnormal if

$$d(\widetilde{g}_n, g_{n+1}) \geqslant \alpha\varphi\,, \tag{5.4}$$

where α is a parameter that needs to be determined from examples of normal and abnormal network change. Note that the median $\widetilde{g}_n$ is by definition a graph that minimizes φ in equation (5.3).

Earlier in this chapter it was pointed out that $\widetilde{g}_n$ is not necessarily unique. If several instances $\widetilde{g}_{n_1}, \ldots, \widetilde{g}_{n_t}$ of $\widetilde{g}_n$ exist, one can apply equations (5.3) and (5.4) to all of them. This will result in a series of values $\varphi_1, \ldots, \varphi_t$ and a series of values $d(\widetilde{g}_{n_1}, g_{n+1}), \ldots, d(\widetilde{g}_{n_t}, g_{n+1})$. Under a conservative scheme, an abnormal change will be reported if

$$d(\widetilde{g}_{n_1}, g_{n+1}) \geqslant \alpha\varphi_1 \wedge \cdots \wedge d(\widetilde{g}_{n_t}, g_{n+1}) \geqslant \alpha\varphi_t\,.$$

By contrast, a more sensitive change detector is obtained if a change is reported as soon as there exists at least one i for which

$$d(\widetilde{g}_{n_i}, g_{n+1}) \geqslant \alpha\varphi_i,\ \ 1 \leqslant i \leqslant t\,.$$

5.3.2 Median vs. Median Graph, Adjacent in Time (`mma`)

Here we compute two median graphs $\widetilde{g}_1$ and $\widetilde{g}_2$ in windows of lengths L_1 and L_2, respectively, i.e., $\widetilde{g}_1$ is the median of the sequence $(g_{n-L_1+1}, \ldots, g_n)$ and $\widetilde{g}_2$ is the median of the sequence $(g_{n+1}, \ldots, g_{n+L_2})$. We measure now the abnormal change between time n and $n+1$ by means of $d(\widetilde{g}_1, \widetilde{g}_2)$. That is, we compute φ_1 and φ_2 for each of the two windows using equation (5.3) and classify the change from g_n to g_{n+1} as abnormal if

$$d(\widetilde{g}_1, \widetilde{g}_2) \geqslant \alpha \left[\frac{L_1\varphi_1 + L_2\varphi_2}{L_1 + L_2} \right].$$

Measure `mma` can be expected to be even more robust against noise and outliers than measure `msa`. If the considered median graphs are not unique, similar techniques (discussed for measure `msa`) can be applied.

5.3.3 Median vs. Single Graph, Distant in Time (`msd`)

If network changes are evolving rather slowly over time, it may be better not to compare two consecutive graphs g_n and g_{n+1} with each other, but g_n and g_{n+l}, where $l > 1$. Instead of `msa`, as proposed above, we use $d(\widetilde{g}_n, g_{n+l})$ as a measure of change between g_n and g_{n+l}, where l is a parameter defined by the user and depends on the underlying application.

5.3.4 Median vs. Median Graph, Distant in Time (`mmd`)

This measure is a combination of the measures `mma` and `msd`. We use $\widetilde{g}_1$ as defined for `mma`, and let $\widetilde{g}_2 = \text{median}(g_{n+l+1}, \ldots, g_{n+l+L_2})$. Then $d(\widetilde{g}_1, \widetilde{g}_2)$ can serve as a measure of network change between time n and $n+l+1$. Obviously, equations (5.3) and (5.4) can be adapted to `msd` and `mmd` similarly to the way they are adapted to `mma`.

5.4 Experimental Results

The network change detection approaches discussed in Section 5.3 have been applied to the same data as in Chapter 4. A sliding window was applied to the time series of graphs to implement a median filter. Measures of graph distance were then computed at each point in the time series, according to the approaches outlined in Section 5.3. This results in a new time series of numbers that represents the extent of network change occurring at any point in the time series. Experimental results were obtained for each technique described in Section 5.3 using both d_1 and d_2. For comparative purposes, we also show results obtained using the application of d_1 and d_2 to consecutive graphs in the time series. The selection of suitable values for L and l were derived through experimentation. The identification of points in the new time series of numbers where *significant* network change occurred was computed using equations (5.3) and (5.4). These points of change are highlighted in the figures by an asterisk ($*$).

Normalization of edge weight was applied to the computation of distance measure d_2 to prevent edge weight dominating the distance measure. Normalization consisted in dividing the absolute edge weight differences by the maximum of the two weights from each graph. This approach required less tuning than the application of a single scalar multiplier to the vertex sum.

5.4.1 Edit Distance and Single Graph vs. Single Graph Adjacent in Time (ssa)

Figure 5.3[1] shows results for edit distance applied to consecutive graphs of the time series for the topology-only measure d_1. This measure produces three significant peaks (on days 25, 65, and 90). The figure also shows several secondary peaks that may also indicate events of potential interest. Also, there is significant minor fluctuation throughout the whole data set appearing as background noise.

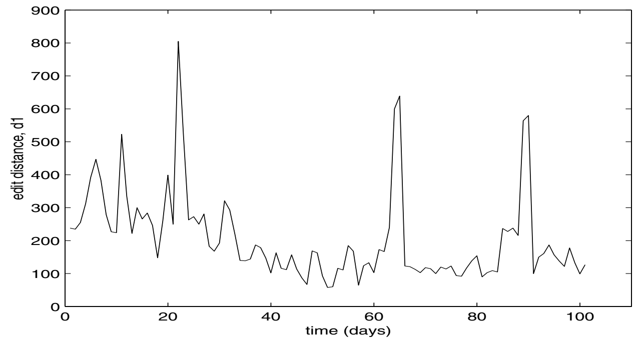


Fig. 5.3. Consecutive days using measure d_1.

Figure 5.4 shows the results for distance measure d_2 being applied to consecutive graphs in the time series. While this measure considers both topology and traffic, it does not appear to provide any additional indicators of change additional to those found using the topology only measure. The main difference with this measure is that it has increased the amplitude of the first two peaks so that they now appear as major peaks. This suggests that these peaks were a result of network change consisting of large change in edge weights.

The findings for this technique indicate that the application of measures to consecutive graphs in a time series is most suited for detecting daily fluctuations in network behavior. This is especially useful for identifying outliers in a time series of graphs.

[1]Figure 5.3 is identical to Figure 4.3 but is shown here again for easier comparison.

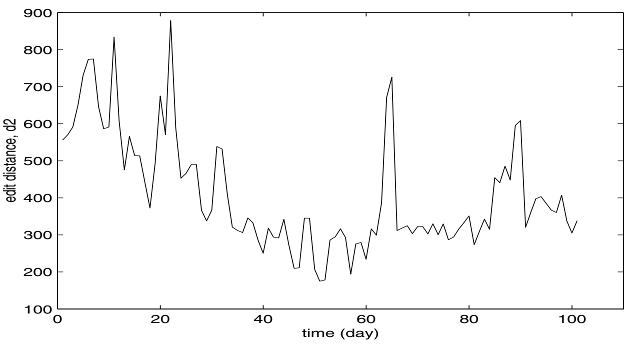


Fig. 5.4. Consecutive days using measure d_2.

5.4.2 Edit Distance and Median Graph vs. Single Graph Adjacent in Time (`msa`)

The results computed for `msa` used a median graph constructed from $L = 5$ consecutive graphs with $\alpha = 2$ used to compute the threshold of significant network change. Figures 5.5 and 5.6 show `msa` results for topology and topology plus traffic, respectively.

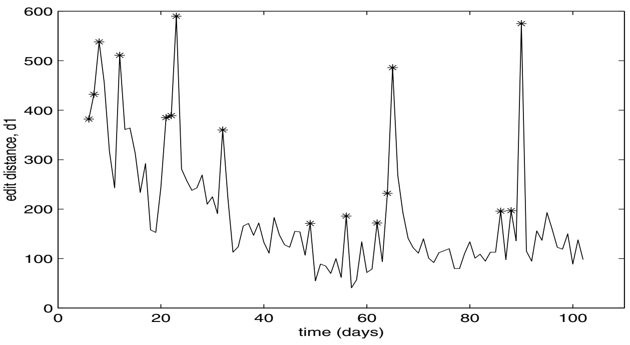


Fig. 5.5. `msa` using measure d_1.

In Figure 5.5 there are now three additional major peaks identified on days 8, 12, and 32, compared to those observed in `ssa`. The peaks occurring on days 8 and 12 have been accentuated using `msa`. This method also shows that there was significant change on days 6, 7, 8, 12, 21, 22, 23, 32, 49, 56, 62, 64, 65, 86, 88, and 90 based on the threshold using $\alpha = 2$. Raising the threshold (i.e., $\alpha = 3$) results in a reduction in the number of days on which significant change was detected to three. Here significant change occurred on days 56, 65, and 90. Interestingly enough, day 56 corresponded to a minor peak, yet has been deemed significant based on the deviations observed within the preceding median graph window. Finally, from Figure 5.5 it is evident that the network is undergoing a gradual transition from considerable daily network change to less daily change between days one through 60.

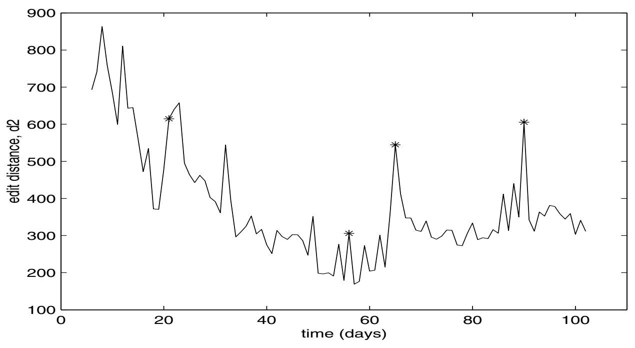


Fig. 5.6. `msa` using measure d_2.

The results presented in Figure 5.6 show the effect of introducing edge weight. While the peaks here occur at the same time as the unweighted experiment, their amplitudes do differ somewhat. The amplitude of peaks on days 22, 65, and 90 have been reduced, relative to the two peaks at the start of the time series. This suggests that these changes are more influenced by changes in topology. The number of significant events detected is noticeably smaller for the same value α. Significant change was detected on four occasions: days 21, 56, 65, and 90. The inclusion of edge weight has resulted in an increase in average deviation within the median graph window, which is the result of a large traffic variation on edges. This had the overall effect of increasing the threshold for detection of significant change.

This method is particularly useful for detection of significant events with improved robustness to noise. Increasing the length of the median graph window results in a greater smoothing effect.

5.4.3 Edit Distance and Median Graph vs. Median Graph Adjacent in Time (`mma`)

The `mma` procedure computes an edit distance between two adjacent median graphs in the time series. In this experiment a median graph window length of $L_1 = L_2 = 5$ was used. There is no reason why the window lengths must be equal for both median graphs. A value of $\alpha = 2$ was used.

Results obtained using d_1 are shown in Figure 5.7. In this figure there are only two major peaks, one occurring at the start of the time series and around day 65. Significant change was detected around day 7 and on days 12 and 46. Surprisingly, significant change was not detected around day 65. This is most likely due to the increased smoothing effect, which would have reduced the effect of the large outlier

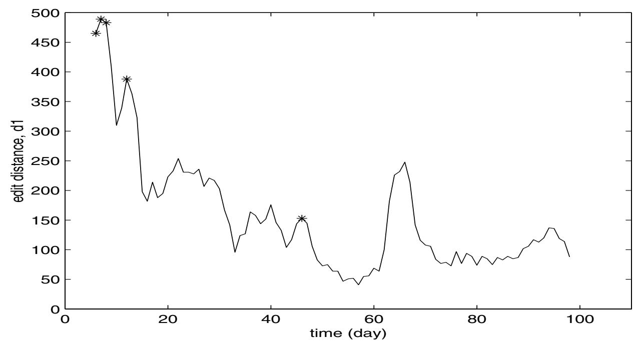


Fig. 5.7. mma using measure d_1.

occurring on day 65. Significant change is, however, detected on day 65 with a value of α that is below 1.8. The mma procedure improves the observation of network change decreasing slowly between days 1 through 60.

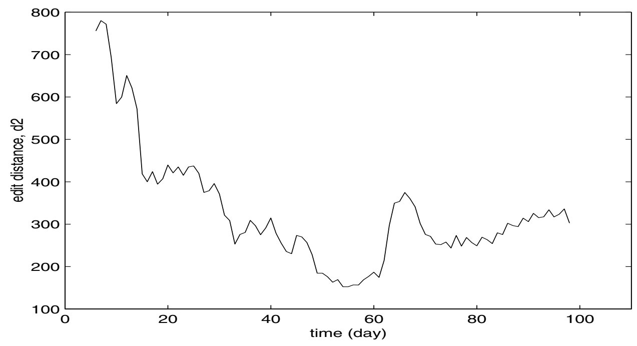


Fig. 5.8. mma using measure d_2.

When distance measure d_2 is applied to the mma procedure the results shown in Figure 5.8 are obtained. Here the results are very similar to those attained using measure d_1. The main point to note is that there are no points identifed as exhibiting significant change with a value of $\alpha = 2$. This again indicates that inclusion of edge weight has increased the average deviation within the median graph window. The value of α must be reduced below 1.5 to get the same results as those of measure d_1.

The use of the mma procedure seems particularly useful for providing an additional smoothing effect that eliminates the effects caused by large outliers. This makes it more robust to noise than that of msa.

5.4.4 Edit Distance and <u>M</u>edian Graph vs. <u>S</u>ingle Graph <u>D</u>istant in Time (msd)

The experiment involving the msd procedure used a median graph window of length $L = 5$ with an offset of $l = 10$ graphs between the median graph and the second graph used in the distance computation. Values of $\alpha = 3$ for d_1 and $\alpha = 2$ for d_2 were used in the detection of significant change. The larger value of α for d_1 was required to reduce an unusually large number of significant events being detected with $\alpha = 2$. Figures 5.9 and 5.10 only have outputs at points relating to the graph that is distant in time. Thus the starting point of a trace is dependent on both the size L of the median graph window and the offset period l between graphs.

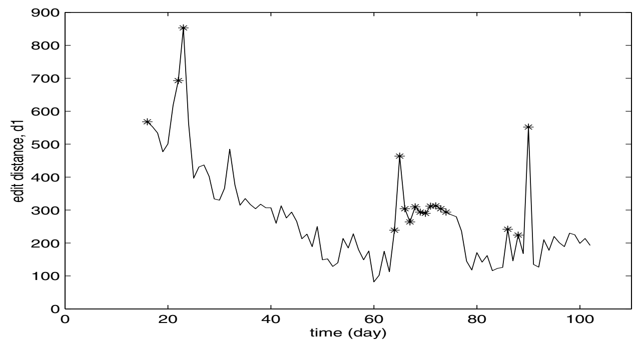


Fig. 5.9. msd using measure d_1.

Figure 5.9 shows the results achieved for measure d_1. This figure shows three major peaks on days 23, 65, and 90. The main point of interest in this result is the cluster of indicators of significant change between days 64 and 74. This suggests that there was a step change in network behavior that took place on day 64. This characteristic could not clearly be observed using ssa, msa, or mma. It is also important to note that the indicators of significant change were based on a threshold of three times the average deviation occurring within the median graph window, indicating a large change in network behavior.

Figure 5.10 shows the results achieved for measure d_2. Again there was quite a degree of similarity with the results achieved using measure d_1. The main difference was the relative increase in the peak occurring on day 90.

While the msd procedure would be better suited to a data set exhibiting gradual change, it was crucial in highlighting a step change in network behavior.

5.4.5 Edit Distance and <u>M</u>edian Graph vs. <u>M</u>edian Graph <u>D</u>istant in Time (mmd)

Median graph windows of length $L_1 = L_2 = 5$ were used in the mmd procedure with an offset of $l = 10$ graphs between the two median graphs used in the distance computation. A value of $\alpha = 3$ was used for d_1 and $\alpha = 2$ for d_2.

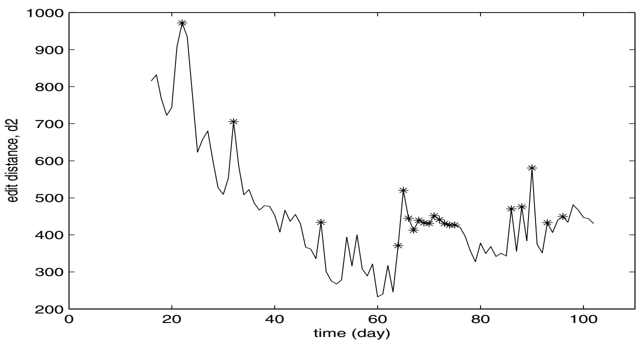


Fig. 5.10. msd using measure d_2.

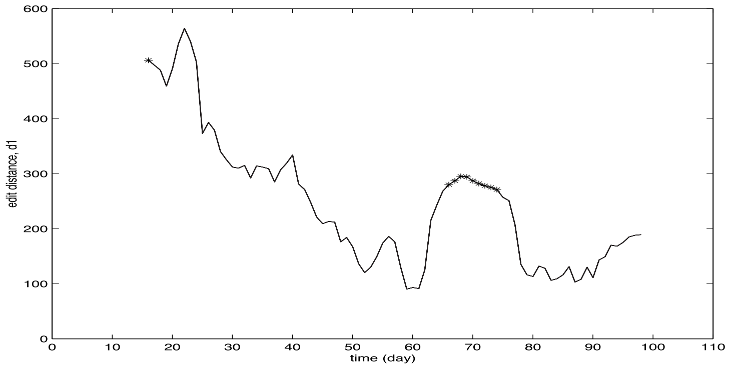


Fig. 5.11. mmd using measure d_1.

The results for mmd can be seen in Figures 5.11 and 5.12 using measures d_1 and d_2, respectively. The two figures are very similar to one another with peaks and significant change detected in the same periods. Compared with the msd results the mmd procedure provides additional smoothing, giving greater robustness to noise and large outliers. This can be seen by the relative fall in amplitude of the peak on day 65 and the near elimination of the peak on day 90. Like the msd procedure, mmd is also capable of showing the step change in the network behavior commencing on day 64.

5.5 Conclusions

In this chapter the use of the median graph for detecting abnormal change in data networks is proposed. The median of a sequence of graphs S is a graph that minimizes the average edit distance from all members in S. Thus the median can be regarded as the best single representative of a sequence of graphs. Abnormal change in a network can be detected by computing the median of a time series of graphs over a window of a given

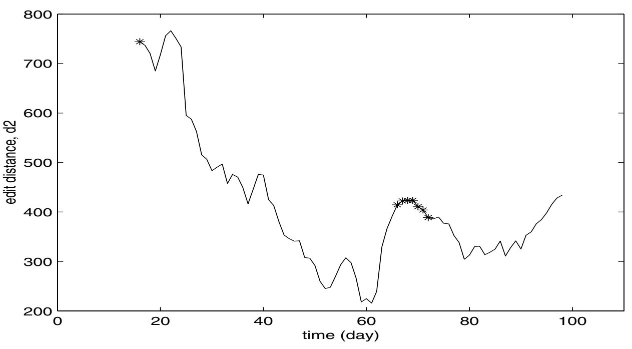


Fig. 5.12. mmd using measure d_2.

span, and comparing it to a graph, or the median of a sequence of graphs, following that window. Whenever the comparison results in a large graph distance value, it can be concluded that an abnormal change has occurred.

The computation of the median of a set of graphs is a computationally expensive process that is intractable by optimal procedures [28]. However, the graphs considered here have the distinctive property that their node labels are unique. This results in optimal procedures for the median graph computation. In fact, it can be expected that the proposed procedures can be applied to networks consisting of millions of nodes and links.

The median graph was applied in four techniques for abnormal change detection in data networks. The data set under observation was collected from a single physical link in the core of a large-enterprise data network. It consisted of logical communications observed on the link. Distance measures were computed between combinations of median graphs and single graphs in the time series. Network change was deemed to be significant when a measure of graph distance exceeded a predefined threshold. The threshold value is determined by multiplying a constant by the average deviation occurring within the median graph window(s). It is important to note that when significant change is detected it does not automatically imply that a performance anomaly exists. A network manager would use this information as an early warning indicator that a problem may be developing.

The four median graph-based techniques (msa, mma, msd, and mmd) for detecting change in data networks provide considerable improvement over the technique that applies distance measures d_1 and d_2 to consecutive graphs in the time series of graphs (i.e., ssa). In particular, these approaches provide increased robustness to noise and large outliers. An appreciable smoothing effect was achieved by the use of mma and mmd. Both msd and mmd were able to detect a change in the state of behavior of the network. These methods are particularly suited to the detection of slow changes in network behavior. All four approaches were able to identify periods during which network change was steadily increasing or decreasing. This is more evident with the mma and mmd techniques. Finally, all four techniques were able to detect points at which significant network change had occurred. From a network management perspective it is possible that these points of significant change could be correlated to various types of abnormal change in the behavior of the network.

6

Graph Clustering for Abnormal Change Detection

6.1 Introduction

Graph similarity measures, including graph edit distance, and their application to the detection of abnormal change in telecommunication networks were introduced in Chapter 4. In Section 3.3, the median of a set of graphs was studied. Also this concept has proven useful to measure change in communication networks and to detect unusual behavior. Recently, graph clustering based on graph edit distance has been proposed [78]. In this chapter, some known clustering algorithms will be reviewed first. Then the potential of graph clustering for analyzing time series of graphs and telecommunication networks will be studied.

Clustering is the process of dividing a set of objects into groups, or clusters, such that objects that are similar are assigned to the same cluster, while dissimilar objects are put into different clusters. Clustering is a fundamental technique in the whole discipline of computer science. However, almost all clustering algorithms published until today are based on object representations in terms of feature vectors. Only very few papers address the clustering of symbolic data structures, particularly the clustering of graphs. For a general introduction to clustering see [66, 96]. Graph clustering was addressed in [155]. Recently an extension of self-organizing maps [106] for graph clustering was proposed in [78]. This method will be reviewed in greater detail in Section 6.2.2.

The meaning of the term "graph clustering" in the literature is not unique. In [149] the identification of groups of nodes *within the same graph* is called "graph clustering," while in [78, 155] the term is used to denote procedures that cluster graphs rather than feature vectors. Throughout this chapter, "graph clustering" is to be interpreted in the latter sense.

The rest of this chapter is organized as follows. In the next section the most important clustering algorithms for the domain of feature vectors will be reviewed. In Section 6.3 techniques will be introduced that allow us to extend the algorithms presented in Section 6.2 into the graph domain. The application of the resulting graph clustering algorithms to time series of graphs and the detection of abnormal change in telecommunication networks will be discussed in Section 6.4. Finally, conclusions will be drawn in Section 6.5.

6.2 Clustering Algorithms

It is not attempted to exhaustively cover all existing clustering methods in this section. Only some of the more popular algorithms will be taken into regard.

The existing clustering algorithms can be classified into hierarchical and nonhierarchical approaches. In hierarchical clustering, the set of given objects is iteratively split into smaller subsets (top-down), or small subsets are iteratively merged into larger ones (bottom-up). Under such a procedure a sequence of hierarchically nested subsets, or clusters, is produced rather than a single clustering. Nonhierarchical approaches are "single shot" procedures that generate just one particular grouping of the given objects. Hierarchical and nonhierarchical clustering algorithms will be introduced in Sections 6.2.3 and 6.2.4, respectively.

In many nonhierarchical clustering algorithms the desired number of clusters has to be given as a parameter. But in many applications this number is not known. Cluster validation indices can be used to find the optimum number of clusters in a clustering problem automatically. Some of these validation indices will be reviewed in Section 6.2.3.

Traditional clustering algorithms make "hard" decisions when assigning objects to clusters. That is, object $\mathbf{x}_i$ either belongs to cluster c_j or does not. If $\mathbf{x}_i$ belongs to c_j then $\mathbf{x}_i$ can't belong to any other cluster c_k, $k \neq j$. In fuzzy information processing, class membership is no longer defined in a binary fashion. Hence, an object $\mathbf{x}_i$ can belong to a number of clusters at the same time, with a different membership degree for each. Fuzzy clustering deals with algorithms that allow us to make such "soft," multiple cluster assignments. Fuzzy clustering will be discussed in Section 6.2.4.

6.2.1 Hierarchical Clustering

In the following we consider a set of objects

$$\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_M\}.$$

A clustering of $\mathbf{X}$ is a set of subsets

$$C = \{c_1, \dots, c_k\}$$

such that $c_i \subseteq \mathbf{X}$, $c_i \neq \emptyset$; $c_i \cap c_j = \emptyset$ for $i \neq j$; $\bigcup_{i=1}^{k} c_i = \mathbf{X}$ for $i, j = 1, \dots, k$.

The basic version of the hierarchical clustering algorithm is given in Figure 6.1. In the algorithm, $d(c_i, c_j)$ is a function that computes the distance between clusters c_i and c_j; for details see below. Obviously, there are M clusters before the algorithm enters the repeat loops. In each run through the repeat loop the number of clusters is reduced by one. Therefore, the algorithm terminates after the repeat loop has been executed $M - 1$ times. We observe that $R_0 = \{\{\mathbf{x}_1\}, \dots, \{\mathbf{x}_M\}\}$ and $R_{M-1} = \{\{\mathbf{x}_1, \dots, \mathbf{x}_M\}\}$.

The clusters resulting from the hierarchical clustering algorithm can be represented as a tree with M levels, where R_{M-1} corresponds to the root and R_0 to the level of the leaves. The distance on the horizontal axis corresponds to the distance between clusters. Such a tree representation is also called a *dendrogram*.

```
input:  $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_M\}$
output: a hierarchical partitioning of set $\mathbf{X}$, i.e., a sequence $R_0$,
        $R_1, \ldots, R_{M-1}$, where each $R_i$ is a partition of set $\mathbf{X}$, and
        $R_i$ is finer than $R_{i+1}$
begin
$R_0 = \{c_1 = \{\mathbf{x}_1\}, c_2 = \{\mathbf{x}_2\}, \ldots, c_M = \{\mathbf{x}_M\}\}$;
$t = 0$;
repeat
        $t = t + 1$;
        put each cluster $c_i$ from $R_{t-1}$ into $R_t$;
        find among all pairs of clusters in $R_t$ the one with
        minimum distance, i.e., find $(c_i, c_j)$ such that
        $d(c_i, c_j) = \min\{(c_r, c_s) \mid c_r \neq c_s; c_r, c_s \in R_t\}$
        /* ties are broken arbitrarily */
        generate a new cluster $c_{\text{new}} = c_i \cup c_j$;
        remove $c_i$ and $c_j$ from $R_t$;
        add $c_{\text{new}}$ to $R_t$
until   all $\mathbf{x}_i$ belong to the same cluster
end
```

Fig. 6.1. Hierarchical clustering.

Example 6.1. Assume that set $\mathbf{X}$ consists of the seven points $A, B, \ldots, G$ in the x-y plane shown in Figure 6.2. The dendrogram resulting from the hierarchical clustering algorithm is depicted in Figure 6.3. First, objects B and C are merged into one cluster. Next D and E, and then F and G are merged. In the next step the cluster consisting of B and C is merged with object A, and so on. Finally, one cluster is obtained that includes all seven objects.

One important detail needed for the implementation of the hierarchical clustering algorithm is the definition of the distance of clusters, $d(c_i, c_j)$. The three most popular distance functions are the following:

- **single-linkage distance:**

$$d(c_i, c_j) = \min\{d(\mathbf{x}, \mathbf{y}) \mid \mathbf{x} \in c_i, \mathbf{y} \in c_j\}.$$

  The distance of two clusters is equal to the distance of the two closest representatives, one from c_i and the other from c_j.
- **complete-linkage distance:**

$$d(c_i, c_j) = \max\{d(\mathbf{x}, \mathbf{y}) \mid \mathbf{x} \in c_i, \mathbf{y} \in c_j\}.$$

  Here the representatives that have the largest distance define the distance of two clusters.

- **average distance:**

$$d(c_i, c_j) = \frac{1}{|c_i||c_j|} \sum_{\mathbf{x} \in c_i} \sum_{\mathbf{y} \in c_j} d(\mathbf{x}, \mathbf{y}).$$

  This is the average distance of two elements that belong to different clusters.

It has been reported by several authors that the complete-linkage distance generally leads to compact clusters, while the single-linkage distance has a tendency to produce rather diffuse clusters. The behavior of the average distance is somewhat between complete- and single-linkage. Another possibility is to represent each cluster by its mean or its median and to use the distance between those as distance measure for clusters.

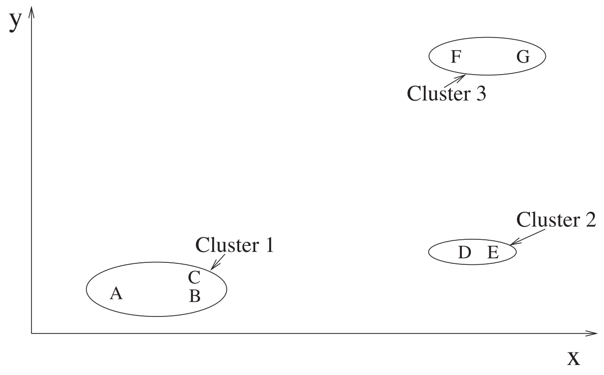


Fig. 6.2. A clustering example. (The clusters correspond to the horizontal dashed line in Figure 6.3).

A dendrogram, such as the one shown in Figure 6.3, is an excellent tool to visualize the structure of the given objects. The dendrogram can also be used to partition set **X** into a given number of clusters. For this purpose one needs only to split the dendrogram at a certain height. For an example see the horizontal dashed line in Figure 6.3 that splits the data into three clusters, namely, $\{A, B, C\}$, $\{D, E\}$, and $\{F, G\}$.

Usually one needs to reorder the elements of set **X** in a dendrogram in order to achieve a “nice” graphical representation in which branches of the tree don’t cross each other. Obviously, such a reordering is always possible, regardless of the number of elements in set **X**.

The algorithm in Figure 6.1 is a bottom-up hierarchical procedure. It starts with singleton clusters and successively builds larger sets. It is also possible to start with the full set **X** and split it recursively into subsets. Such an algorithm is called top-down. A closer examination reveals that the splitting criteria needed in top-down clustering are often more difficult to define than the merging criteria required in bottom-up clustering. Therefore, bottom-up approaches are much more popular than hierarchical top-down clustering.

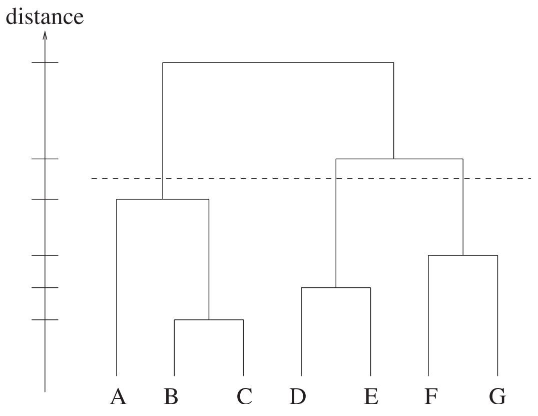


Fig. 6.3. Dendogram of Figure 6.2.

6.2.2 Nonhierarchical Clustering

The task in nonhierarchical clustering is to produce one partition of the given set of objects, $\mathbf{X}$, into subsets $\{c_1, \dots, c_k\}$ such that $c_i \subseteq \mathbf{X}$; $c_i \neq \emptyset$; $c_i \cap c_j \neq \emptyset$ for $i \neq j$; $\bigcup_{i=1}^{k} c_i = \mathbf{X}$ for $i, j = 1, \dots, k$. In this section we will review the k-means clustering algorithm and self-organizing maps (SOMs), which are two of the most popular approaches to nonhierarchical clustering.

A pseudocode description of the k-means algorithm is given in Figure 6.4. Notice that k, the number of clusters to be produced, is needed as a parameter for the algorithm.[1] Potential methods for the selection of the initial cluster centers include:

- randomly select k elements of $\mathbf{X}$ as initial cluster centers;
- randomly generate k objects as initial cluster centers; if the objects are represented by feature vectors, i.e., if each object is a point in n-dimensional real space, then one usually selects k random points inside the hyperbox that is defined by the extremal points of set $\mathbf{X}$.

Popular termination criteria are:

- termination after a predefined number of iterations;
- the cluster centers change only by a small amount from one iteration to the next;
- only a few objects from set $\mathbf{X}$ are assigned to a different cluster from one iteration to the next;
- the error is below a predefined threshold (see below).

Example 6.2. Let $\mathbf{x}_1 = 1$; $\mathbf{x}_2 = 2$; $\mathbf{x}_3 = 4$; $\mathbf{x}_4 = 5$; $k = 2$.
Assume we choose $\mathbf{m}_1 = 1$; $\mathbf{m}_2 = 2$.

[1]Techniques to find optimal values of this parameter automatically will be discussed in Section 6.2.3.

- 1st iteration:
  $\mathbf{x}_1 \mapsto \mathbf{m}_1; \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4 \mapsto \mathbf{m}_2;$
  $\mathbf{m}_1 = 1; \mathbf{m}_2 = 3.66;$
- 2nd iteration:
  $\mathbf{x}_1, \mathbf{x}_2 \mapsto \mathbf{m}_1; \mathbf{x}_3, \mathbf{x}_4 \mapsto \mathbf{m}_2;$
  $\mathbf{m}_1 = 1.5; \mathbf{m}_2 = 4.5;$
- 3rd and following iterations:
  no more change, i.e., $c_1 = \{\mathbf{x}_1, \mathbf{x}_2\}, c_2 = \{\mathbf{x}_3, \mathbf{x}_4\}$.

Example 6.3. For this example we use again Figure 6.2. Assume $k = 3$ and the initial cluster centers are A, D, and F. Then the result of the k-means algorithm is the one shown in Figure 6.2. However, if we select A, B, and C as initial cluster centers, we will obtain the clustering shown in Figure 6.5.

From this example, it becomes clear that the result of the k-means clustering algorithm critically depends on the choice of the initial cluster centers. Therefore, often a number of runs of the complete algorithm are executed, each with a different set of initial cluster centers. Finally, the best result is chosen.

To measure the quality of a clustering produced by the k-means algorithm, the sum E of all quadratic distances from the cluster centers can be used. Let $\mathbf{m}_j$ be the center of cluster c_j. Then we define

$$e_j = \sum_{x \in c_j} d^2(\mathbf{m}_j, \mathbf{x})$$

and choose the clustering that minimizes

$$E = \sum_{j=1}^{k} e_j .$$

It has been proposed to add some postprocessing steps to the results generated by the k-means algorithm. Possible postprocessing operations include the merging of a pair of (small) clusters that have a small distance, or the splitting of clusters that have many elements and/or large variance.

Self organizing maps (SOMs) is a popular method in information processing that is inspired by biological systems [106]. It can be used for various purposes. In this chapter, we consider SOMs exclusively for the purpose of clustering.

A pseudocode description of the classical SOM algorithm is given in Figure 6.6. Given a set of patterns $\mathbf{X}$, the algorithm returns a prototype $\mathbf{y}_i$ for each cluster c_i. The prototypes are sometimes called *neurons*. The number of clusters, k, is a parameter that must be provided a priori. In the algorithm, first each prototype $\mathbf{y}_i$ is randomly initialized (line 4). In the main loop (lines 5–10) one randomly selects an element $\mathbf{x} \in \mathbf{X}$ and determines the neuron $\mathbf{y}^*$ that is nearest to $\mathbf{x}$. In the inner loop (lines 7, 8) one considers all neurons $\mathbf{y}$ that are within a neighborhood $N(\mathbf{y}^*)$ of $\mathbf{y}^*$, including $\mathbf{y}^*$, and updates them according to the formula in line 8. The effect of neuron updating is to move neuron $\mathbf{y}$ closer to pattern $\mathbf{x}$. The degree by which $\mathbf{y}$ is moved toward $\mathbf{x}$

```
input:  $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_M\}$; number of clusters, $k$
output: clusters $c_1, \ldots, c_k$
begin
choose $k$ initial cluster centers $\mathbf{m}_1, \ldots, \mathbf{m}_k$;
repeat
        assign each $\mathbf{x}_i$ to its nearest cluster center $\mathbf{m}_j$;
        recompute the cluster centers $\mathbf{m}_1, \ldots, \mathbf{m}_k$;
until   termination criterion fulfilled
end
```

Fig. 6.4. *k*-means clustering.

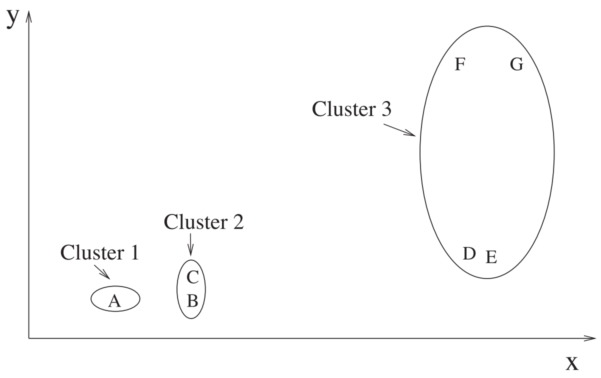


Fig. 6.5. An example.

is controlled by the parameter α, which is called the *learning rate*. It has to be noted that α is dependent on the distance between $\mathbf{y}$ and $\mathbf{y}^*$, i.e., the smaller this distance is, the larger is the change on neuron $\mathbf{y}$. After each iteration through the repeat-loop, the learning rate α is reduced by a small amount, thus facilitating convergence of the algorithm. It can be expected that after a sufficient number of iterations the $\mathbf{y}_i$'s have moved into areas where many $\mathbf{x}_j$'s are concentrated. Hence each $\mathbf{y}_i$ can be regarded as a cluster center. The cluster around center $\mathbf{y}_i$ consists of exactly those patterns that have $\mathbf{y}_i$ as closest neuron.

SOM clustering is in fact similar to the k-means clustering algorithm. The main difference between both algorithms is that SOM is incremental in its nature, while k-means is batch-oriented. That is, under k-means cluster centers are updated only after a complete cycle through all $\mathbf{x} \in \mathbf{X}$ has been conducted, while in SOM cluster centers are updated immediately after representation of each individual object $\mathbf{x} \in \mathbf{X}$. Similarly to k-means, SOM clustering critically depends on a good initialization strategy.

```
(1)  input:  a set of patterns, $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_M\}$
(2)  output: a set of prototypes, $\mathbf{Y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_k\}$
(3)  begin
(4)  initialize $\mathbf{Y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_k\}$ randomly
(5)  repeat select $\mathbf{x} \in X$ randomly
(6)           find $\mathbf{y}^*$ such that $d(\mathbf{x}, \mathbf{y}^*) = \min\{d(\mathbf{x}, \mathbf{y}) \mid \mathbf{y} \in \mathbf{Y}\}$
(7)           for all $\mathbf{y} \in N(\mathbf{y}^*)$ do
(8)           $\mathbf{y} = \mathbf{y} + \alpha(\mathbf{x} - \mathbf{y})$
(9)           reduce learning rate $\alpha$
(10) until termination criterion is true
(11) end
```

Fig. 6.6. The SOM algorithm.

6.2.3 Cluster Validation

Many clustering methods, for example, k-means and SOM, require the number of clusters being known beforehand. But often this kind of knowledge is not available. There are two potential approaches to finding the optimal number of clusters automatically. The first is clustering algorithms that dynamically change the number of clusters during their execution. Typically, large clusters are split, or small clusters that are adjacent in feature space are merged with each other. However, the proper parameterization of these algorithms, for example, the thresholds that define when a cluster is "small" or "large," remains an open problem.

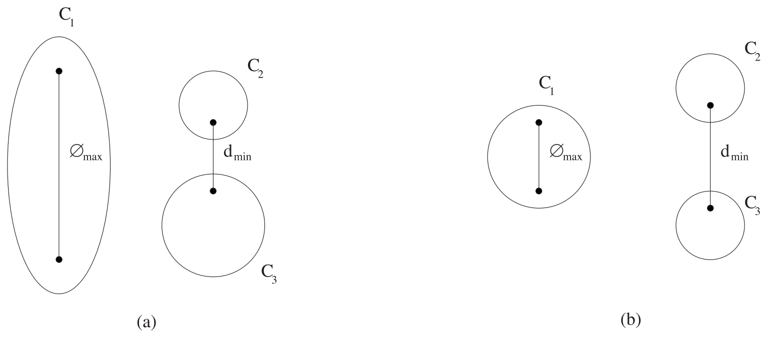


Fig. 6.7. Example of Dunn index: (a) Clustering with a small value of D; (b) Clustering with a large value of D; $d_{\min} = \min\{d(c_i, c_j) \mid i, j = 1, 2, 3\}$.

In this section we will focus on another, simpler approach to automatically finding the optimal number of clusters. It is based on so-called *cluster validation indices*. A cluster validation index is a function that measures the quality of a given clustering. Hence, given a cluster validation index and a clustering algorithm, such as k-means or SOM, one can execute the clustering algorithm a number of times, specifying a different number of clusters to be produced in each run, and finally select the clustering that yields the optimal value of the cluster validation index.

At first glance, the sum E of all quadratic distances defined in Section 6.2.2 may look like a suitable cluster validation index. It turns out, however, that this measure assumes its minimum value $E = 0$ for $k = M$, i.e., for the case in which each cluster consists of just a single element. A number of more appropriate indices have been proposed in the literature [55, 66, 70, 76, 96]. Some of them are reviewed below.

Dunn Index

Let $d(c_i, c_j)$ denote the distance of clusters c_i and c_j. For this function any of the measures discussed in Section 6.2.1 can be used, i.e., single-linkage, complete-linkage, average distance, as well as the distance between the mean or the median of c_i and c_j. Furthermore, let $\Delta(c_i)$ denote the maximum distance within cluster c_i, i.e.,

$$\Delta(c_i) = \max\{d(\mathbf{x}, \mathbf{y}) \mid \mathbf{x}, \mathbf{y} \in c_i\},$$

and let $\Delta_{\max}$ be the maximum within-cluster distance taken over all clusters, i.e.,

$$\Delta_{\max} = \max\{\Delta(c_i) \mid i = 1, \ldots, k\}.$$

Then the Dunn index D is defined as

$$D = \frac{1}{\Delta_{\max}} \min\{d(c_i, c_j) \mid i, j = 1, \ldots, k\}.$$

The Dunn index D considers the distance of the two nearest clusters in relation to the largest distance within a single cluster. Clearly, the larger D is, the better is the considered clustering. For a graphical illustration see Figure 6.7. It is easy to verify that

$$D \in [0, \infty].$$

This index is easy to compute. Its main shortcoming, however, is the fact that it is vulnerable to outliers, i.e., if only a single outlier is added to the data, the value of the index may drastically change.

Davis–Bouldin Index

Let $\mathbf{m}_i$ be the center of cluster c_i; $i = 1, \ldots, k$. The average distance of element $\mathbf{x}_l \in \mathbf{c}_i$ to $\mathbf{m}_i$ is given by

$$d_i = \frac{1}{|c_i|} \sum_{x_l \in c_i} d(\mathbf{x}_l, \mathbf{m}_i).$$

First we define

$$R_{ij} = R_{ji} = \frac{d_i + d_j}{d(\mathbf{m}_i, \mathbf{m}_j)}.$$

This quantity can be interpreted as the compactness of clusters c_i and c_j in relation to their distance. Clearly, for a good clustering, R_{ij} will be small. For a graphical representation see Figure 6.8.

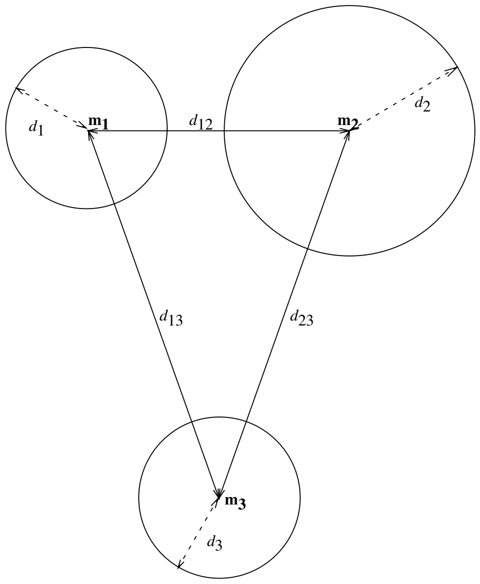


Fig. 6.8. Example of Davis–Bouldin index: The smaller the d_i's and the further the $\mathbf{m}_j$'s are away from each other, the smaller is DB, i.e., the better is the clustering. In the figure, the notation $d_{ij} = d(\mathbf{m}_i, \mathbf{m}_j)$ is used.

Next we are interested, for cluster c_i, in the worst case, i.e., in the cluster c_j that yields the maximum value of R_{ij}. Hence we define

$$R_i = \max\{R_{ij} \mid j = 1, \ldots, k;\ i \neq j\}.$$

Finally the Davis–Bouldin index DB is defined as the average of the R_i's taken over all clusters, i.e.,

$$DB = \frac{1}{k} \sum_{i=1}^{k} R_i.$$

Clearly, the smaller the value of DB is, the better the clustering. It is easy to see that

$$D \in [0, \infty].$$

Goodman–Kruskal Index
We define

$$\rho(\mathbf{x}_i, \mathbf{x}_j) = \begin{cases} 1 \text{ if } \mathbf{x}_i \text{ and } \mathbf{x}_j \text{ belong to different clusters,} \\ 0 \text{ if } \mathbf{x}_i \text{ and } \mathbf{x}_j \text{ belong to same cluster.} \end{cases}$$

To compute the index, one considers all quadruples $(\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_r, \mathbf{x}_s)$ where $\mathbf{x}_i \neq \mathbf{x}_j$, $\mathbf{x}_r \neq \mathbf{x}_s$, $(\mathbf{x}_i, \mathbf{x}_j) \neq (\mathbf{x}_r, \mathbf{x}_s)$. For each such quadruple, the quantities $d(\mathbf{x}_i, \mathbf{x}_j)$, $\rho(\mathbf{x}_i, \mathbf{x}_j)$, $d(\mathbf{x}_r, \mathbf{x}_s)$, and $\rho(\mathbf{x}_r, \mathbf{x}_s)$ are computed. A quadruple is *concordant* if either

$$d(\mathbf{x}_i, \mathbf{x}_j) < d(\mathbf{x}_r, \mathbf{x}_s) \wedge \rho(\mathbf{x}_i, \mathbf{x}_j) < \rho(\mathbf{x}_r, \mathbf{x}_s)$$

or

$$d(\mathbf{x}_i, \mathbf{x}_j) > d(\mathbf{x}_r, \mathbf{x}_s) \wedge \rho(\mathbf{x}_i, \mathbf{x}_j) > \rho(\mathbf{x}_r, \mathbf{x}_s).$$

By contrast, a quadruple is called *discordant* if either

$$d(\mathbf{x}_i, \mathbf{x}_j) < d(\mathbf{x}_r, \mathbf{x}_s) \wedge \rho(\mathbf{x}_i, \mathbf{x}_j) > \rho(\mathbf{x}_r, \mathbf{x}_s)$$

or

$$d(\mathbf{x}_i, \mathbf{x}_j) > d(\mathbf{x}_r, \mathbf{x}_s) \wedge \rho(\mathbf{x}_i, \mathbf{x}_j) < \rho(\mathbf{x}_r, \mathbf{x}_s).$$

Notice that there are usually quadruples that are neither concordant nor discordant, for example if $\rho(\mathbf{x}_i, \mathbf{x}_j) = \rho(\mathbf{x}_r, \mathbf{x}_s)$.

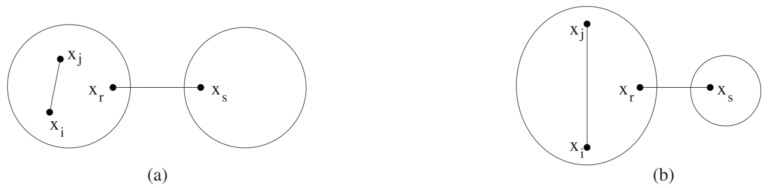


Fig. 6.9. (a) Example of a concordant quadruple (x_i, x_j, x_r, x_s); (b) Example of a discordant quadruple (x_i, x_j, x_r, x_s).

Intuitively, concordant quadruples are hints to a good clustering, while discordant quadruples represent situations that are typical of a poor clustering. For a graphical illustration see Figure 6.9. Let S_+ and S_- denote the number of concordant and discordant quadruples, respectively. The Goodman–Kruskal index GK is defined as

$$GK = \frac{S_+ - S_-}{S_+ + S_-}.$$

Clearly, a good clustering is characterized by a high value of GK. The denominator has a normalizing effect such that

$$GK \in [-1, 1].$$

This measure can be expected to be robust against outliers, but it has a high computational complexity, which amounts to $\mathcal{O}(M^4)$.

6.2.4 Fuzzy Clustering

All clustering techniques considered so far are "hard" procedures in the sense that an element $\mathbf{x}_i$ belongs to exactly one cluster c_j. If $\mathbf{x}_i$ belongs to c_j it can't belong to any other cluster $c_l \neq c_j$. Fuzzy, or soft, clustering is a generalization that allows an element $\mathbf{x}_i$ to be simultaneously assigned to a number of different clusters, with some degree of membership for each cluster. In this section we consider the fuzzy version of the k-means clustering algorithm as a representative from among a number of different fuzzy clustering algorithms proposed in the literature [73].

A fuzzy set [195] is a set F of elements $x \in F$ together with a membership function $\mu(x) \to [0, 1]$ that assigns a membership degree out of the interval $[0, 1]$ to each element $x \in F$. The two extremal cases are $\mu(x) = 0$, which means that x doesn't belong to F at all, and $\mu(x) = 1$, which means that x is a full member of F. Any other value $0 < \mu(x) < 1$ indicates that x is a member of F to a certain degree. Classical sets are obtained as a special case of fuzzy sets if the membership function μ can take on only the values $\mu(x) = 0$ and $\mu = (1)$ for any $x \in F$.

The input elements to be clustered by the fuzzy k-means algorithms are the same as under the conventional k-means algorithms, i.e., they are a (nonfuzzy, or crisp) set $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_M\}$. However, the clusters $c_1, \dots, c_k$ are fuzzy sets. That is, for each element $\mathbf{x}_i$ and each cluster c_j there exists a membership value $\mu_j(\mathbf{x}_i) \in [0, 1]$ indicating the degree to which $\mathbf{x}_i$ belongs to cluster c_j. Usually it is required that

$$\sum_{j=1}^{k} \mu_j(\mathbf{x}_i) = 1 \quad \text{for} \quad i = 1, \dots, M \tag{6.1}$$

and

$$0 < \sum_{i=1}^{M} \mu_j(\mathbf{x}_i) < M \quad \text{for} \quad j = 1, \dots, k. \tag{6.2}$$

The first condition means that for each $\mathbf{x}_i$ the membership degrees, taken over all clusters c_j, must sum up to unity. By means of the second condition, clusterings are excluded in which all elements belong to just a single cluster with membership degree one.

Let $\mathbf{m}_j$ denote the center of cluster c_j. We compute the degree of membership of an element $\mathbf{x}_i$ in cluster c_j as follows:

$$\mu_j(\mathbf{x}_i) = \begin{cases} 1, \quad \text{if} \quad \mathbf{x}_i = \mathbf{m}_j, \\ \left(\sum_{l=1}^{k} \left(\frac{d(\mathbf{x}_i, \mathbf{m}_j)}{d(\mathbf{x}_i, \mathbf{m}_l)} \right)^{\frac{2}{\beta - 1}} \right)^{-1}, \quad \text{otherwise.} \end{cases} \tag{6.3}$$

It can be verified that $\mu_j(\mathbf{x}_i)$ will have a value close to 1 if $\mathbf{x}_i$ is near $\mathbf{m}_j$, and a value close to 0 if $\mathbf{x}_i$ is far away from $\mathbf{m}_j$. In the equation, $\beta > 1$ is a parameter that controls how quickly the value of $\mu_j(\mathbf{x}_i)$ drops from 1 to 0 if $\mathbf{x}_i$ is moved away from $\mathbf{m}_j$. A value of β close to 1 means a quick drop of the membership function, while large values of β imply a slower decrease of $\mu_j(\mathbf{x}_i)$.

In the classical version of the k-means algorithm, there is an iterative updating of the cluster centers. A similar updating operation on the cluster centers takes place in fuzzy k-means clustering:

$$\mathbf{m}_j = \frac{\sum_{i=1}^{M} \mu_j(\mathbf{x}_i)\mathbf{x}_i}{\sum_{i=1}^{M} \mu_j(\mathbf{x}_i)}. \tag{6.4}$$

This operation can be interpreted as computing the weighted average over all input elements, $\mathbf{x}_i$, where each $\mathbf{x}_i$ is weighted by its degree of membership in cluster c_j.

Given equations (6.3) and (6.4), the fuzzy k-means clustering algorithm can be formulated as shown in Figure 6.10. Notice that, similarly to its classical counterpart given in Figure 6.4, the number of clusters needs to be given as a parameter. The same techniques for cluster center initialization and termination as discussed in Section 6.2.2 can be applied, but the error measure E needs to be appropriately redefined, for example by letting

$$FE = \sum_{i=1}^{M} \sum_{j=1}^{k} \mu_j(\mathbf{x}_i) d(\mathbf{x}_i, \mathbf{m}_j).$$

It can be shown that the validity of equations (6.1) and (6.2) is maintained during the execution of the fuzzy k-means algorithm. The result of the fuzzy k-means algorithm is a set of cluster centers $\mathbf{m}_1, \ldots, \mathbf{m}_k$ and a sequence of membership values, $(\mu_1(\mathbf{x}_i), \ldots, \mu_k(\mathbf{x}_i))$, for each input element $\mathbf{x}_i$. It is possible to "harden" this result by means of a *defuzzyfication* procedure. One possibility of defuzzyfication is the *winner-take-all* strategy. Under this strategy, $\mathbf{x}_i$ is assigned to the cluster c_j with maximum membership value $\mu_j(\mathbf{x}_i)$, i.e., we assign $\mathbf{x}_i$ to c_j if and only if

$$j = \arg\max\{\mu_j(\mathbf{x}_i) \mid j = 1, \ldots, k\}.$$

This decision rule is equivalent to assigning $\mathbf{x}_i$ to that cluster c_j the center $\mathbf{m}_j$ of which is closest to $\mathbf{x}_i$.

A multitude of clustering algorithms have been published in the literature, but there is no general principle that can be used to predict which method performs best. The actual performance of a method crucially depends on the given task and the underlying data, and has to be experimentally determined.

6.3 Clustering in the Graph Domain

All methods discussed in Section 6.2 are exclusively devoted to the clustering of objects that are described in terms of feature vectors. Only a few works have been published

```
input:  $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_M\}$; number of clusters, $k$
output: cluster centers $\mathbf{m}_1, \ldots, \mathbf{m}_k$; membership degrees $\mu_j(\mathbf{x}_i)$
        for $i = 1, \ldots, M$ and $j = 1, \ldots, k$
begin
choose $k$ initial cluster centers $\mathbf{m}_1, \ldots, \mathbf{m}_k$;
repeat
        compute $\mu_j(\mathbf{x}_i)$ for $i = 1, \ldots, M$ and $j = 1, \ldots, k$
        using equation (6.3);
        recompute cluster centers $\mathbf{m}_1, \ldots, \mathbf{m}_k$ using equation (6.4);
until   termination criterion fulfilled
end
```

Fig. 6.10. Fuzzy k-means clustering.

on graph clustering, i.e., on the clustering of objects that are represented by means of graphs.

In [155] the objects to be clustered are random graphs. In such a graph, each node and each edge has a probability assigned to it, which reflects its likelihood of existence. An information-theoretic measure is used to assess the quality of a clustering. This measure also controls the assignment of individual random graphs to clusters.

In [78] an extension of the SOM clustering procedure from feature vectors to the domain of graphs is described. The basic steps of the algorithm are identical to those given in Figure 6.6. However, two important extensions were introduced to make the SOM clustering applicable to graphs. The first is the replacement of the Euclidian distance by graph edit distance (see Chapter 3), and the second a generalization of the SOM updating rule (see line 8 in Figure 6.6) from n-dimensional real space to the graph domain.

The method introduced in [78] was augmented by cluster validation indices for the graph domain in [79]. As a result, a graph clustering procedure is obtained that can find the optimal number of clusters automatically.

Next we discuss, from a general standpoint, what is needed to extend the clustering algorithms introduced in Section 6.2 from n-dimensional real space to the domain of graphs. Obviously, one of the concepts essential to each of the clustering algorithms is a distance function. Very often, Euclidian distance is used. In order to apply clustering algorithms in the graph domain, we need a function $d(g_1, g_2)$ that computes the distance of any two given graphs, g_1 and g_2. Fortunately, there are a number of such graph distance functions; see Chapter 4. Also, graph distance measures d_1 and d_2 introduced in Section 5.2 can be used as a graph distance function in the context of graph clustering.

A close look reveals that the availability of a function that computes graph distance is already sufficient to implement the hierarchical clustering algorithm discussed in Section 6.2.1 (see Figure 6.1). For the implementation of the k-means algorithm (Figure 6.4) we need, in addition to a distance function on graphs, a method to compute the

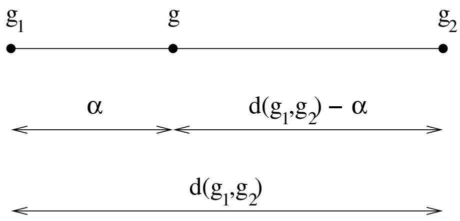


Fig. 6.11. Illustration of weighted mean.

center of a cluster, i.e., the center of a finite set of graphs. For this task, any procedure for median (or set median) graph computation can be used; see Chapters 3 and 5.

Median graph computation is not needed for the implementation of the SOM algorithm in the graph domain. Here a different operation is required. We need to synthesize a new graph g on the basis of two given graphs g_1 and g_2 such that

$$d(g_1, g) = \alpha \quad \text{and} \quad d(g, g_2) = d(g_1, g_2) - \alpha, \tag{6.5}$$

where α is a given constant, called the *learning rate*. It controls the degree by which a cluster center is moved closer to an input element (see Figure 6.6, line 8). Intuitively, g is a graph on the connecting line between g_1 and g_2 at distance α to g_1 in the graph domain; see Figure 6.11 for an illustration. Such a graph has been called a *weighted mean* in [24].

Fortunately, it turns out that the computation of a weighted mean g, given g_1, g_2, and α, is a straightforward task. In fact, this computation can be accomplished as a postprocessing step of edit distance computation. The computation of $d(g_1, g_2)$ yields a sequence of edit operations $(e_1, \dots, e_l)$ that transform g_1 into g_2 with minimum cost. Let $c(e_i)$ denote the cost of edit operation e_i in this sequence. Now g can be synthesized from g_1 by selecting a subsequence $(e_{i_1}, \dots, e_{i_r})$ of sequence $(e_1, \dots, e_l)$, $r \leq l$, such that $\sum_{j=1}^{r} c(e_{i_j})$ approximates α as closely as possible, and applying all edit operations of this subsequence to g_1. It can be proven that the graph g that results from this procedure is in fact a weighted mean, satisfying equation (6.5).

The procedure for weighted mean graph computation is also applicable to distance measures d_1 and d_2 introduced in Chapter 5, as we will show in the following two examples.

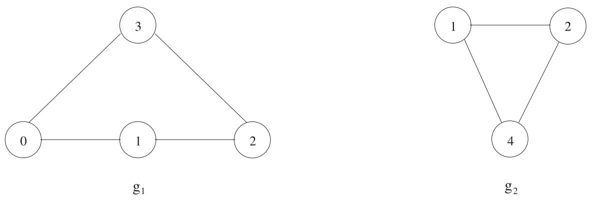


Fig. 6.12. An example of weighted mean graph using d_1 (see also Figure 6.13 and Figure 6.14).

Example 6.4. In Figure 6.12 two graphs, g_1 and g_2, are shown. It is easy to verify that $d_1(g_1, g_2) = 8$. An optimal, i.e., minimum cost, sequence of edit operations transforming g_1 into g_2 is as follows:

$$\begin{aligned}
e_1 &: \text{delete edge } (0, 1) \\
e_2 &: \text{delete edge } (0, 3) \\
e_3 &: \text{delete node } 0 \\
e_4 &: \text{delete edge } (2, 3) \\
e_5 &: \text{delete node } 3 \\
e_6 &: \text{insert node } 4 \\
e_7 &: \text{insert edge } (1, 4) \\
e_8 &: \text{insert edge } (2, 4)
\end{aligned}$$

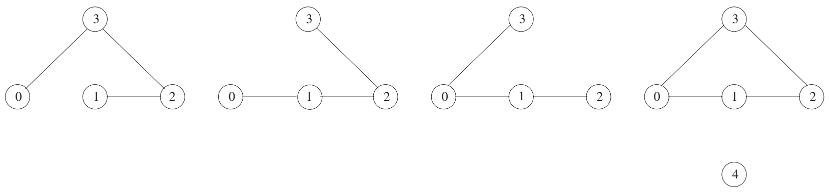


Fig. 6.13. Four weighted mean graphs of g_1 and g_2 in Figure 6.12 for $\alpha = 1$.

For $\alpha = 1$ we may select any of the edge deletions e_1, e_2, e_4, or the node insertion e_6 and apply it to g_1. The resulting graphs are shown in Figure 6.13. We observe that $d(g_1, g) = 1$ and $d(g_2, g) = 7$ for any of the weighted mean graphs g shown in Figure 6.13. Note, however, that it is not legitimate to select any of the other edit operations, i.e., e_3, e_5, e_7, or e_8, because these edit operations can't be carried out independently. This means they need additional edit operations to produce a valid graph. For example, if we delete node 0 through edit operation e_3, we also must delete edges (0, 1) and (0, 3) through edit operations e_1 and e_2, respectively. Similarly, insertion of edge $(1, 4)$ through edit operation e_7 requires the insertion of node 4 by means of e_6.

For $\alpha = 2$ we may select any pair of edit operations among (e_1, e_2), (e_1, e_4), (e_1, e_6), (e_2, e_4), (e_2, e_6), (e_4, e_6), (e_6, e_7), (e_6, e_8), resulting in one of the graphs shown in Figure 6.14. Here we observe that $d(g_1, g) = 2$ and $d(g, g_2) = 6$, for any graph g depicted in Figure 6.14. Any pair of edit operations other than the ones listed above are not legitimate for the same reasons as given for the case $\alpha = 1$.

The cases $\alpha = 3, \ldots, 7$ are similar. Note that for $\alpha = 0$ and $\alpha = 8$, the weighted means will be isomorphic to g_1 and g_2, respectively.

Example 6.5. Two graphs with edge labels are shown in Figure 6.15. Here we observe that $d_2(g_1, g_2) = 6$ if we let $c = 1$. An optimal sequence of edit operations that transform g_1 into g_2 is:

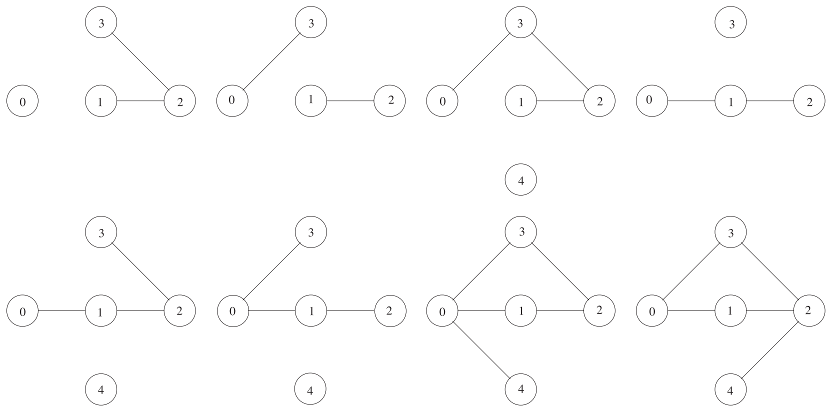


Fig. 6.14. Eight weighted mean graphs of g_1 and g_2 in Figure 6.12 for $\alpha = 2$.

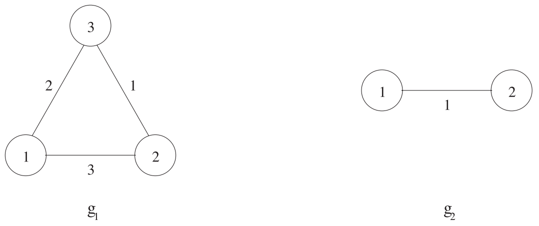


Fig. 6.15. An example of weighted mean graph using d_2 (see also Figures 6.16 and 6.17).

e_1 : change the label on edge (1, 2) from 3 to 1 (cost 2)
e_2 : delete edge (1, 3) (cost $2)^2$
e_3 : delete edge (2, 3) (cost $1)^2$
e_4 : delete node 3 (cost 1)

To generate a weighted mean for $\alpha = 1$, we may choose any of the following operations:

- delete edge (2, 3)
- change the label on edge (1, 2) from 3 to 2
- change the label on edge (1, 3) from 2 to 1

The resulting three graphs are shown in Figure 6.16. Obviously, $d_2(g_1, g) = 1$ and $d_2(g, g_2) = 5$ for any of the weighted means.

For $\alpha = 2$ the following possibilities exist:

- delete edge (2, 3) and change the label of edge (1, 2) from 3 to 2
- delete edge (2, 3) and change the label of edge (1, 3) from 2 to 1
- delete edge (1, 3)
- change the label on edge (1, 2) from 3 to 1

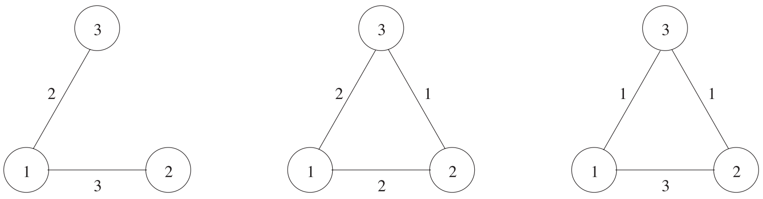


Fig. 6.16. Three weighted mean graphs of g_1 and g_2 in Figure 6.15 for $\alpha = 1$.

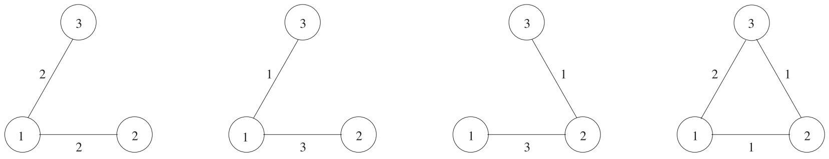


Fig. 6.17. Three weighted mean graphs of g_1 and g_2 in Figure 6.15 for $\alpha = 2$.

The resulting graphs are shown in Figure 6.17. Now we have $d_2(g_1, g) = 2$ and $d_2(g, g_2) = 4$.

The graphs for $\alpha = 3, \ldots, 6$ can be constructed similarly.

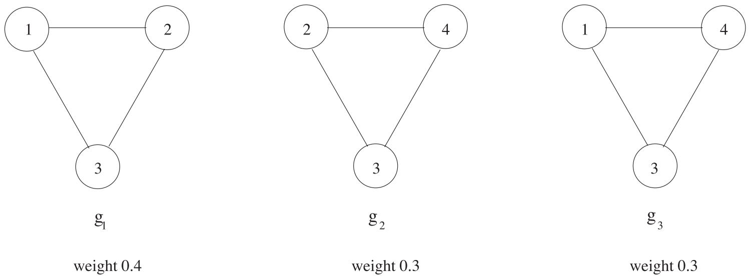


Fig. 6.18. An example of weighted median graph computation using d_1 (see also Figure 6.19).

Having a procedure for weighted mean graph computation at our disposal, we are able to implement the SOM clustering algorithm in the graph domain. Next we consider the fuzzy k-means clustering algorithm. It is easy to see that for the implementation of equation (6.3) the availability of a function for graph distance computation is sufficient. However, equation (6.4) needs to be elaborated on. Here the aim is to compute a cluster center $\mathbf{m}_j$. As a generalization over the median graph computation procedure discussed in the context of nonfuzzy k-means clustering, each of the elements $\mathbf{x}_i$ now has an individual weight, namely $\mu_j(\mathbf{x}_i)$. Yet the computation of the median of a weighted set of graphs can be considered as a straightforward extension of normal median graph

[2] Note that the deletion of an edge is equivalent to substituting the label by weight 0.

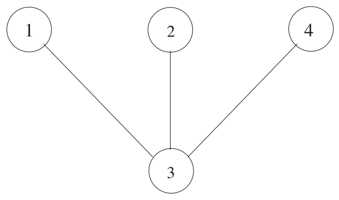


Fig. 6.19. Weighted mean of the graphs in Figure 6.18.

computation. All that needs to be done is to include multiple copies of each graph in the underlying set, with the number of copies of a graph being proportional to its weight.

Example 6.6. Three graphs, g_1, g_2, and g_3, together with their weights are shown in Figure 6.18. Let us consider the computation of a weighted median under graph distance d_1. From g_1, g_2, and g_3 we produce a set of ten graphs. This set includes four copies of g_1, three copies of g_2, and three copies of g_3. Application of the median graph computation procedure as described in Chapter 3 to this set yields the graph shown in Figure 6.19.

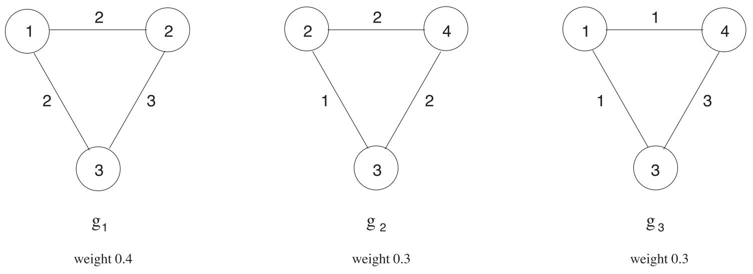


Fig. 6.20. An example of weighted median graph computation using d_2 (see also Figure 6.21).

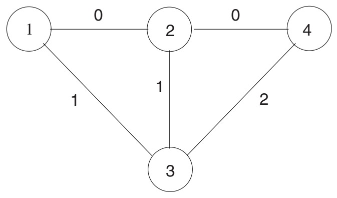


Fig. 6.21. Weighted mean of the graphs in Figure 6.20.

Example 6.7. The graphs in Figure 6.18, augmented by edge labels, are shown in Figure 6.20. The weighted median computed under graph distance d_2 (see Section 5.2) is depicted in Figure 6.21.

Further inspection of the pseudocode in Figure 6.10 reveals that no additional tools are needed to implement fuzzy k-means clustering in the graph domain.

It is easy to see that all cluster validation indices introduced in Section 6.2.3 are based only on distances. Therefore, only graph distance is needed for their implementation in the graph domain.

6.4 Clustering Time Series of Graphs with Applications to the analysis of Telecommunication Networks

Applications of median graphs and graph similarity measures to the analysis of time series of graphs and telecommunication networks have been studied in Chapters 4 and 5. In this section we will discuss how graph clustering can potentially be used in this area.

For the purpose of clustering, a given time series of graphs $S = (g_1, \dots, g_m)$ is interpreted as a set $G = \{g_1, \dots, g_m\}$, in which multiple instances of the same graph may occur. Application of any of the clustering algorithms described in Section 6.2 will result in a partition of set G into subsets $c_1, \dots, c_k$ such that $c_i \subseteq G$; $c_i \neq \emptyset$; $c_i \cap c_j = \emptyset$ for $i \neq j$; $\bigcup_{i=1}^{k} c_i = G$; $i, j = 1, \dots, k$. We expect that graphs in the same cluster are similar to each other, while graphs that are assigned to different clusters are dissimilar. It is easy to verify that, under any of the considered algorithms, multiple occurrences of the same graph will always be assigned to the same cluster.

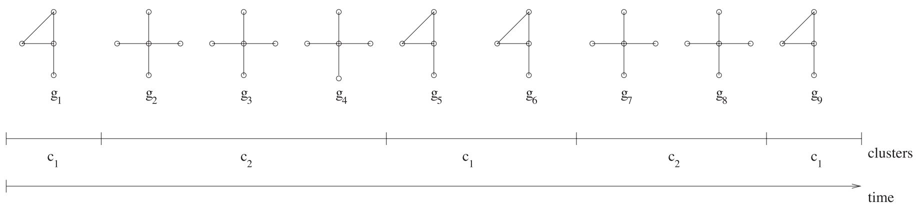


Fig. 6.22. Example of noncontiguous clustering.

We cannot expect, however, that the clusters to be produced are *contiguous*. A cluster c_j is contiguous if all its graphs can be ordered $(g_{i_1}, \dots, g_{i_l})$ such that $i_1 + 1 = i_2; i_2 + 1 = i_3; \dots; i_{l-1} + 1 = i_l$. In general the graphs of a cluster will be noncontiguous, i.e., they are scattered over the whole time series with different clusters interleaving each other. But this effect may be desirable. It allows us, for example, to identify sets of similar graphs, i.e., similar states of the network, without requiring that these states occur at consecutive points in time. A graphical illustration is shown in Figure 6.22. Here the time series of graphs consists of two noncontiguous clusters, c_1

and c_2. A partitioning of a time series of graphs, such as the one shown in Figure 6.22, may be useful to characterize a whole series of graphs in more general and global terms than just referring to individual points in time where abnormal change occurs. For the sequence shown in Figure 6.22 one could infer, for example, that the sequence consists of two different alternating types of graphs. If the behavior of the network follows some (deterministic or nondeterministic) rules, it may be possible to automatically detect such rules by means of grammatical inference [29] or tools from the discipline of machine learning (see Chapter 11). Once appropriate rules have been derived from an existing sequence, they may be used to predict the future behavior of this sequence, or to patch up points in time where no measurements are available. In such a system, clustering would be needed to provide the basic similarity classes of network states.[2]

Any noncontiguous clustering can be easily converted into a contiguous one by splitting the clusters into contiguous subsequences. Figure 6.23 shows the contiguous clusters that are obtained from Figure 6.22. Another procedure for the generation of contiguous clusters can be derived from the hierarchical clustering algorithm introduced in Section 6.2.1. Here we start with the individual graphs as the initial clusters. Then, whenever two clusters (or individual graphs) are to be merged, only candidates that are adjacent in time will be considered. In other words, only pairs of (contiguous) clusters that will result in a new contiguous cluster are eligible for merging.

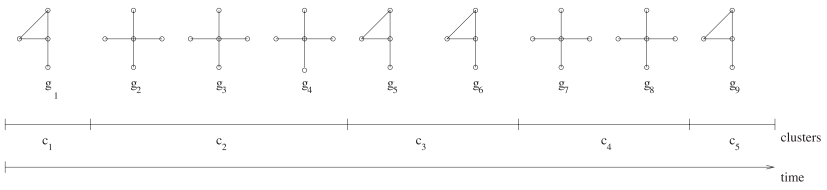


Fig. 6.23. Contiguous clustering resulting from splitting the clusters in Figure 6.22.

Graph clustering can be used to develop additional tools for abnormal change detection in time series of graphs. The basic idea is to first produce a contiguous clustering of a given time series of graphs. Then pairs of clusters that are adjacent in time are compared to each other. If their dissimilarity is above some threshold, it can be concluded that some abnormal change has occurred at the point in time at which the two clusters meet. Potential measures for cluster similarity, or dissimilarity, are single linkage, complete linkage, average distance, and distance of median. It can be expected that such measures for abnormal change detection have noise resistance properties that are similar to the median graph. But they are more general because they can adapt to the actual data in a more flexible way than median graphs, which are always computed over a time window of fixed size.

[2]For a more detailed treatment of predicting future network behavior and recovering missing informaton see Chapter 11.

6.5 Conclusion

Graph representation and similarity measures on graphs have proven useful in many applications, including the analysis of telecommunication networks. In this chapter a number of clustering algorithms are reviewed and their extension from the n-dimensional real space to the graph domain is studied. Furthermore, the application of these algorithms to the monitoring of telecommunication networks is discussed.

It can be expected that graph clustering will provide additional tools to detect abnormal change in time series of graphs and telecommunication networks. Moreover, it is potentially useful to identify patterns of similar behavior in long network observation sequences. This may lead to enhanced tools that are able not only to detect network events in short time windows, but also to analyze network behavior and deliver human-understandable high-level interpretation over long periods of time.

7

Graph Distance Measures based on Intragraph Clustering and Cluster Distance

7.1 Introduction

Various graph distance measures were considered in previous chapters. All of these measures have in common that the distance of two given graphs g_1 and g_2 is equal to zero if and only if g_1 and g_2 are isomorphic to each other. Sometimes, however, it may be desirable to have a more flexible distance measure for which $d(g_1, g_2) = 0$ if g_1 and g_2 are similar, but not necessarily isomorphic. Such a measure is potentially useful to make our graph-distance-based computer network monitoring procedures more robust against noise and small random perturbations.

A family of new graph distance measures with this property is introduced in this chapter. Given two graphs g_1 and g_2, their distance $d(g_1, g_2)$ is computed in two steps. First an intragraph clustering procedure is applied that partitions the set of nodes of each graph into a set of clusters, based on the weights on the edges. In the second step, the distance $d(C_1, C_2)$ of the two clusterings C_1 and C_2 derived in the first step will be computed, and this quantity will serve as our graph distance measure $d(g_1, g_2)$. Thus the problem of computing the distance of two graphs is transformed into the problem of measuring the similarity of two given clusterings. For the latter problem, some algorithms are known from the literature [147, 164]. In addition to these algorithms we propose a novel approach to measuring cluster similarity, based on bipartite graph matching. The intragraph clustering procedure can be combined with any of the methods that measure clustering similarity. Hence a whole family of new graph distance measures, rather than just a single procedure, is obtained.

In the next section our basic terminology and an algorithm for intragraph clustering will be introduced. Then in Section 7.3, distance measures for clusterings will be discussed. Combining the concepts from Sections 7.2 and 7.3, our novel graph-matching methods will be described in Section 7.4. In Section 7.5, we continue with a discussion of applications of the new graph distance measures in the field of computer network monitoring. Finally, in Section 7.6 conclusions and suggestions for further research are provided.

7.2 Basic Terminology and Intragraph Clustering

We consider graphs with unique node labels as introduced in Chapter 3. To simplify our notation, the terminology $g = (V, E, \beta)$ will be used to denote a graph, where V is the finite set of nodes, $E \subseteq V \times V$ is the set of edges, and $\beta : E \rightarrow L_E$ is the edge labeling function. Because of the unique node labeling property, we drop node labeling function α from our graph representation and identify, without loss of generality, nodes and node labels.

In this chapter new graph distance measures are introduced. To compute these distance measures we first apply an intragraph clustering procedure. The term *intragraph* refers to the fact that the purpose of this clustering procedure is to build groups, or clusters, *within* the same graph. By contrast, intergraph clustering procedures, such as those discussed in Chapter 6, group whole sets or subsets of graphs into clusters.

In general, clustering refer to the process of dividing a set of objects into groups, or clusters, such that objects that are similar to each other are assigned to the same cluster, while dissimilar objects are put into different clusters. Clustering is a fundamental technique in many disciplines of science and engineering. A brief introduction to clustering is provided in Chapter 6.

In this chapter we propose to use the graph clustering algorithm described in [196] to partition the set of nodes V of a given graph $g = (V, E, \beta)$ into subsets, or clusters. The original version of this algorithm starts with a set of general objects $O = \{o_1, o_2, \ldots, o_n\}$ rather than a set of nodes. Objects from O are typically members of n-dimensional Euclidean space. First the distance $d(o_i, o_j)$ is computed for each pair of objects o_i, o_j. Then the objects together with their distances are represented by a distance graph G. In this distance graph each node corresponds to an object o_i, and there is an edge between each pair of objects o_i and o_j, the weight of which represents the distance $d(o_i, o_j)$. Next the minimum spanning tree (mst) of distance graph G is computed. The mst of any graph g with weights on its edges is a tree that includes all nodes of g and has, among all such trees, the smallest sum of edge weights. An algorithm for computing the minimum spanning tree T of an arbitrary graph $g = (V, E, \beta)$ is given below:

```
initialize $T$ as an empty tree;
choose the edge $e = (x, y)$ with minimum weight $\beta(e)$ from $E$
and include it in $T$;
while $T$ has fewer nodes than $|V|$ do
        find the edge $e$ with minimum weight connecting $T$
        with $g - T$ and add it to $T$
```

For the purpose of clarity we would like to mention here that any edge (x, y) that is added to T during the execution of the while-loop has one of its incident nodes, either x or y (but not both), already in T. Say the node already included in T is x. Then adding edge (x, y) means adding that edge plus node y. An example of this algorithm is shown in Figure 7.1. The algorithm starts with edge (H, G) and adds edges (H, G),

(G, F), (F, C), (C, I), (C, D), (H, A), (A, B), (D, E) to T, in this order. The edges belonging to T are printed in bold. It can be proven that the mst of a given graph is uniquely defined if all edge weights are different. Otherwise several msts may exist for the same given graph. In the remainder of this monograph we shall assume, for the purpose of simplification, that each graph under consideration has a unique mst.

As mentioned above, the objective in this chapter is not to cluster a set of objects, as is done in the algorithm described in [196], but to cluster the nodes in a graph $g = (V, E, \beta)$. However, we can directly apply the algorithm given in [196] in case the underlying graph is connected, i.e., any node $y \in V$ can be reached from any other node $x \in V$ via a sequence of edges (x, x_1), (x_1, x_2), (x_2, x_3), …, (x_i, y) that all belong to E. Otherwise, if g is not connected, we add edges to graph g such that it becomes a complete graph (i.e., such that there is an edge (x, y) between any pair of nodes $x, y \in V$). Each of the additional edges e gets assigned a weight $\beta(e) = \infty$. Given graph g we compute an mst, using the algorithm described above. Given the mst, the k edges with the largest weights are determined and deleted from the mst. This results in $k + 1$ subsets, or clusters, of the set of nodes V. Each cluster is defined by a connected component of the mst.

Example 7.1. For this example we use the graph and the mst from Figure 7.1. If we delete just the single edge with maximum weight from the mst, i.e., edge (D, E) with weight 9, then the nodes of graph g are divided into two clusters, the one just including node E, and the other consisting of all other nodes. Alternatively, if we delete the two edges with maximum weight from the mst, we get three clusters, $\{E\}$, $\{A, B\}$, and $\{C, D, F, G, H, I\}$. Deletion of three edges from the mst yields four clusters, $\{E\}$, $\{D\}$, $\{A, B\}$, $\{C, F, G, H, I\}$, and so on.

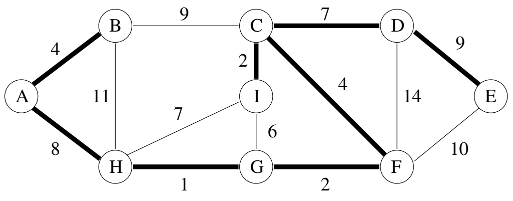


Fig. 7.1. An example of the mst computation.

The number of edges to be deleted from the mst is a parameter that controls the number of clusters to be produced. Two basic strategies can be applied to select appropriate values of this parameter. First, we can put a threshold t on the edge weights and delete all edges from the mst with a weight larger than t. Under this strategy, the number of clusters is not fixed a priori, but depends on the actual weights in the given graph. Secondly, the number of edges to be deleted can be fixed. This will always result in the same number of clusters being produced, independent of the actual weights on the edges. Which of the two strategies is more suitable usually depends on the underlying application.

The algorithm described above is based on the idea that edge weights represent distances and that clusters should comprise objects that have a small distance to each other. Therefore, the minimum spanning tree is computed, and those k edges are deleted from the mst that have the largest weights. There is a dual version of this problem in which edge weights represent affinity between nodes, and we want to cluster those nodes together that have a high degree of affinity. In this dual version we compute the maximum spanning tree of the complete graph analogously to the minimum spanning tree and delete those k edges from the maximum spanning tree that have the smallest weights. If we consider the second version of the problem, a small modification of the graph completion procedure will be required in case g is not connected: To each edge e added to the original graph g with the purpose of turning g into a complete graph, a weight $\beta(e) = 0$, rather than $\beta(e) = \infty$, should be assigned. Intuitively, this means that a missing edge corresponds to the case of zero affinity between two nodes.

We like to conclude this section by pointing out that for the new graph distance measures to be introduced below, any other procedure that partitions the set of nodes of a graph into disjoint subsets, or clusters, can be used instead of the mst-based clustering algorithm.

7.3 Distance of Clusterings

Given a set of objects $O = \{o_1, \ldots, o_n\}$, a clustering of O is a set of subsets $C = \{c_1, \ldots, c_k\}$ such that $c_i \subseteq O, c_i \cap c_j = \emptyset$ if $i \neq j, \bigcup_{i=1}^{k} c_i = O$ for $i, j \in \{1, \ldots, k\}$. Each c_i is called a cluster. The problem considered in this section is to measure how different two clusterings are, i.e., to measure their distance. Throughout the section we will consider two clusterings $C_1 = \{c_{11}, \ldots, c_{1k}\}$ and $C_2 = \{c_{21}, \ldots, c_{2l}\}$ of the same set O. To illustrate the concepts introduced in this section, we will use the set of objects and the clusterings given in the example below.

Example 7.2. $O = \{o_1, o_2, \ldots, o_6\}$;
$C_1 = \{c_{11}, c_{12}, c_{13}\}$ with $c_{11} = \{o_1, o_2\}, c_{12} = \{o_3, o_4\}, c_{13} = \{o_5, o_6\}$;
$C_2 = \{c_{21}, c_{22}, c_{23}\}$ with $c_{21} = \{o_3, o_4\}, c_{22} = \{o_5, o_6\}, c_{23} = \{o_1, o_2\}$;
$C_3 = \{c_{31}, c_{32}, c_{33}\}$ with $c_{31} = \{o_1, o_3\}, c_{32} = \{o_2, o_5\}, c_{33} = \{o_4, o_6\}$;
$C_4 = \{c_{41}, c_{42}\}$ with $c_{41} = \{o_1, o_2, o_3\}, c_{42} = \{o_4, o_5, o_6\}$;
$C_5 = \{c_{51}, c_{52}\}$ with $c_{51} = \{o_1, o_3, o_5\}, c_{52} = \{o_2, o_4, o_6\}$.

7.3.1 Rand Index

The Rand index was introduced in [147]. Consider two clusterings C_1 and C_2 of a set of objects O. The purpose of the Rand index is to measure how different C_1 and C_2 are. In order to compute the Rand index $R(C_1, C_2)$, we consider all pairs of objects (o_i, o_j) with $i \neq j$ from $O \times O$. A pair (o_i, o_j) is called

- **consistent** if either
  - o_i and o_j are in the same cluster in C_1 and in the same cluster in C_2, or
  - o_i and o_j are in different clusters in C_1 and in different clusters in C_2;

- **inconsistent** if either
  - o_i and o_j are in the same cluster in C_1, but in different clusters in C_2, or
  - o_i and o_j are in different clusters in C_1, but in the same cluster in C_2.

Let R^+ be the number of consistent and R^- the number of inconsistent pairs in $O \times O$. Then the Rand index is defined as

$$R(C_1, C_2) = 1 - \frac{R^+}{R^+ + R^-} . \tag{7.1}$$

It is easy to see that $R(C_1, C_2) \in [0, 1]$ for any two given clusterings C_1 and C_2 over a set of objects O. We have $R(C_1, C_2) = 0$ if and only if $k = l$ and there exists a bijective mapping f between C_1 and C_2 such that $f(c_{1i}) = c_{2j}$ for $i, j \in \{1, \ldots, l\}$, which means that the two clusterings are the same except for possibly assigning different names to the individual clusters, or listing the clusters in different order. The case $R(C_1, C_2) = 1$ corresponds to the maximum degree of cluster dissimilarity. It indicates that there exists no consistent pair $(o_i, o_j) \in O \times O$. In order to compute the Rand index, $O(n^2)$ pairs of objects from O have to be considered since $R^+ + R^- = n(n-1)/2$.

Example 7.3. The following values of the Rand index are obtained for the clusterings given in Example 7.2:

$$\begin{array}{lll}
d(C_1, C_2) = 0 , & R^+ = 15 , & R^- = 0 ; \\
d(C_1, C_3) = 6/15 , & R^+ = 9 , & R^- = 6 ; \\
d(C_1, C_4) = 5/15 , & R^+ = 10 , & R^- = 5 ; \\
d(C_1, C_5) = 9/15 , & R^+ = 6 , & R^- = 9 ; \\
d(C_2, C_i) = d(C_1, C_i) & \text{for } i = 3, 4, 5 ; & \\
d(C_3, C_4) = 5/15 , & R^+ = 10 , & R^- = 5 ; \\
d(C_3, C_5) = 5/15 , & R^+ = 10 , & R^- = 5 ; \\
d(C_4, C_5) = 8/15 , & R^+ = 7 , & R^- = 8 .
\end{array}$$

Hence if we order these distance values, we may conclude, for example, that the clustering that is most similar to C_1 is C_2, followed by C_4, C_3, and C_5.

7.3.2 Mutual Information

Mutual information (MI) is a well-known concept in information theory [53]. It measures how much information about random variable Y is obtained from observing random variable X. Let X and Y be two random variables with joint probability distribution $p(x, y)$ and marginal probability functions $p(x)$ and $p(y)$. Then the mutual information of X and Y, $MI(X, Y)$, is defined as

$$MI(X, Y) = \sum_{(x,y)} p(x, y) \log \frac{p(x, y)}{p(x)p(y)} . \tag{7.2}$$

Some properties of MI are summarized below. For a more detailed treatment the reader is referred to [53].

1. $MI(X, Y) = MI(Y, X)$;
2. $MI(X, Y) \geq 0$;
3. $MI(X, Y) = 0$ if and only if X and Y are independent;
4. $MI(X, Y) \leq \max(H(X), H(Y))$, where $H(X) = -\sum_x p(x) \log p(x)$ is the entropy of random variable X ;
5. $MI(X, Y) = H(X) + H(Y) - H(X, Y)$, where $H(X, Y) = -\sum_{(x,y)} p(x, y) \log p(x, y)$ is the joint entropy of X and Y.

In the context of measuring the distance of two given clusterings C_1 and C_2 over a set of objects O, the discrete values of random variable X are the different clusters c_{1i} of C_1 to which an element of O can be assigned. Similarly, the discrete values of Y are the different clusters c_{2j} of C_2 to which an object of O can be assigned. Hence equation (7.2) becomes

$$MI(C_1, C_2) = \sum_{(c_{1i}, c_{2j})} p(c_{1i}, c_{2j}) \log \frac{p(c_{1i}, c_{2j})}{p(c_{1i})p(c_{2j})} \,. \tag{7.3}$$

Given set O and clusterings C_1 and C_2, we can construct a table such as the one shown in Figure 7.2 to compute $MI(C_1, C_2)$. An entry at position (c_{1i}, c_{2j}) in this table indicates how often an object has been assigned to cluster c_{1i} in clustering C_1, and to cluster c_{2j} in clustering C_2. To compute $p(c_{1i}, c_{2j})$ in equation (7.3) we just have to divide the element at position (c_{1i}, c_{2j}) by the sum of all elements in the table. For $p(c_{1i})$ we sum all elements in row c_{1i} and divide again by the sum of all elements in the table. To get $p(c_{2j})$ we proceed similarly, summing over all values in column c_{2j}, and dividing by the total sum.

$C_1 \setminus C_2$	c_{21}	$\ldots$	c_{2j}	$\ldots$	c_{2l}
c_{11}					
$\vdots$					
c_{1i}					
$\vdots$					
c_{1k}					

Fig. 7.2. Illustration of the computation of $MI(C_1, C_2)$.

In contrast with the Rand index, which is normalized to the interval [0, 1], no normalization is provided in equation (7.3). As $MI(C_1, C_2) \leq \max(H(C_1), H(C_2))$ and $H(C) \leq \log(k)$, with k being the number of clusters present in clustering C, the upper bound of $MI(C_1, C_2)$ depends on the number of clusters in C_1 and C_2. To get a normalized value, it was proposed to divide $MI(X, Y)$ in equation (7.3) by $\log(k \cdot l)$, where k and l are the number of discrete values of X and Y, respectively [164]. This leads to the normalized mutual information

$$NMI(C_1, C_2) = 1 - \frac{1}{\log(k \cdot l)} \sum_{(c_{1i}, c_{2j})} p(c_{1i}, c_{2j}) \log \frac{p(c_{1i}, c_{2j})}{p(c_{1i})p(c_{2j})} \ . \tag{7.4}$$

We note that smaller values of $NMI(C_1, C_2)$ indicate a higher degree of similarity, i.e., a smaller distance, of the two clusterings under consideration.

Example 7.4. Some of the values of $MI(C_i, C_j)$ for the clusterings given in Example 7.2 are

$$\begin{aligned}
MI(C_1, C_2) &= 3\left(\frac{1}{3}\log\frac{1/3}{1/3 \cdot 1/3}\right) = \log 3 \approx 0.48,\\
MI(C_1, C_3) &= 6\left(\frac{1}{6}\log\frac{1/6}{1/3 \cdot 1/3}\right) = \log\frac{3}{2} \approx 0.18,\\
MI(C_1, C_4) &= 2\left(\frac{1}{3}\log\frac{1/3}{1/3 \cdot 1/2}\right) + 2\left(\frac{1}{6}\log\frac{1/6}{1/3 \cdot 1/2}\right) = \frac{2}{3}\log 2 \approx 0.2,\\
MI(C_1, C_5) &= 6\left(\frac{1}{6}\log\frac{1/6}{1/3 \cdot 1/2}\right) = 0.
\end{aligned}$$

The corresponding tables are shown in Figure 7.3.

$MI(C_1, C_2)$

	c_{21}	c_{22}	c_{23}	$\sum$
c_{11}	0	0	2	2
c_{12}	2	0	0	2
c_{13}	0	2	0	2
$\sum$	2	2	2	6

$MI(C_1, C_3)$

	c_{31}	c_{32}	c_{33}	$\sum$
c_{11}	1	1	0	2
c_{12}	1	0	1	2
c_{13}	0	1	1	2
$\sum$	2	2	2	6

$MI(C_1, C_4)$

	c_{41}	c_{42}	$\sum$
c_{11}	2	0	2
c_{12}	1	1	2
c_{13}	0	2	2
$\sum$	3	3	6

$MI(C_1, C_5)$

	c_{51}	c_{52}	$\sum$
c_{11}	1	1	2
c_{12}	1	1	2
c_{13}	1	1	2
$\sum$	3	3	6

Fig. 7.3. Illustration of MI-computation.

Furthermore one has

$$\begin{aligned}
&MI(C_2, C_i) = MI(C_1, C_i) \text{ for } i = 3, 4, 5\ ,\\
&MI(C_3, C_4) = MI(C_3, C_5) = MI(C_1, C_4),\\
&MI(C_4, C_5) = 2\left(\frac{1}{3}\log\frac{2/6}{1/2 \cdot 1/2}\right) + 2\left(\frac{1}{6}\log\frac{1/6}{1/2 \cdot 1/2}\right) = \frac{2}{3}\log\frac{4}{3} + \frac{1}{3}\log\frac{2}{3}\ .
\end{aligned}$$

7.3.3 Bipartite Graph Matching

In this subsection a novel procedure for measuring the distance of two clusterings is introduced. It is based on bipartite graph matching.[1] We represent the two given clusterings C_1 and C_2 as one common set of nodes $\{c_{11}, \dots, c_{1k}\} \cup \{c_{21}, \dots, c_{2l}\}$ of a graph, i.e., each cluster from either C_1 or C_2 is considerd to be a node. Then between each pair of nodes (c_{1i}, c_{2j}) an edge is inserted. The weight of this edge is equal to $|c_{1i} \cap c_{2j}|$, i.e., it is equal to the number of elements that occur in both c_{1i} and c_{2j}.

Given this graph, we determine a maximum-weight bipartite graph matching. Such a matching is defined by a subset $\{(c_{1i_1}, c_{2j_1}), \dots, (c_{1i_r}, c_{2j_r})\}$ such that each node c_{1i} and c_{2j} has at most one incident edge, and the total sum of weights is maximized over all possible subsets of edges. Intuitively, the maximum-weight bipartite graph matching can be understood as a correspondence between the clusters of C_1 and the clusters of C_2 such that no two clusters of $C_1$are mapped to the same cluster in C_2, and vice versa. Moreover, the correspondence optimizes the total number of objects that belong to corresponding clusters. Algorithms for computing maximum-weight bipartite graph matching can be found in [104], for example.

The sum of weights w of a maximum-weight bipartite graph matching is bounded from above by the number of objects n in set O. Therefore, a suitable normalized measure for the distance of C_1 and C_2 is

$$BGM(C_1, C_2) = 1 - \frac{w}{n} . \tag{7.5}$$

Clearly, this measure is equal to 0 if and only if $k = l$ and there is a bijective mapping f between the clusters of C_1 and C_2 such that $f(c_{1i}) = c_{2j}$ for $i, j \in \{1, \dots, k\}$. Values close to one indicate that no good mapping between the clusters of C_1 and C_2 exists such that corresponding clusters have many elements in common.

Example 7.5. Based on the clusterings $C_1, \dots, C_5$ introduced in Example 7.2, the graphs used to find the maximum-weight bipartite graph matching between (C_1, C_2), $\dots, (C_4, C_5)$ are shown in Figures 7.4a-g. In these graphs the numbers in the left columns represent the clusters in C_1, and the numbers in the right columns the clusters in C_2. Edges are drawn between any two clusters that have a nonempty intersection. The maximum-weight bipartite matching is indicated in bold. For example, in Figure 7.4c, we compare clusters c_{11}, c_{12}, and c_{13} (represented by numbers $1, 2, 3$ on the left) with clusters c_{41} and c_{42} (represented by numbers $1, 2$ on the right). Since $c_{11} \cap c_{41} = \{o_1, o_2\}$, an edge with weight 2 is drawn between c_{11} and c_{41}. Similarly, an edge with weight 2 is drawn between c_{13} and c_{42}. Cluster c_{12} has one element in common with each c_{41} and c_{42}, which is reflected in the two edges with weight 1. Since all other pairs of clusters have an empty intersection, there are no more edges included in Figure 7.4c. The maximum-weight bipartite matching is the one corresponding to the bold edges. Note that in Figures 7.4a,c,e,f,g the maximum-weight bipartite graph

[1] In bipartite graph matching we aim at matching two subsets of nodes of the same graph with each other. This is different from the meaning of *graph matching* as used in the rest of this book, where our aim is to match the nodes and edges of two different graphs with each other.

matching is uniquely defined, while in Figures 7.4b,d several solutions exist. However, all these solutions lead to the same distance value. The distances are as follows:

$$\begin{array}{ll} BGM(C_1, C_2) = 0, & \\ BGM(C_1, C_3) = 3/6, & BGM(C_1, C_4) = 2/6, \\ BGM(C_1, C_5) = 4/6, & BGM(C_3, C_4) = 2/6, \\ BGM(C_3, C_5) = 2/6, & BGM(C_4, C_5) = 2/6. \end{array}$$

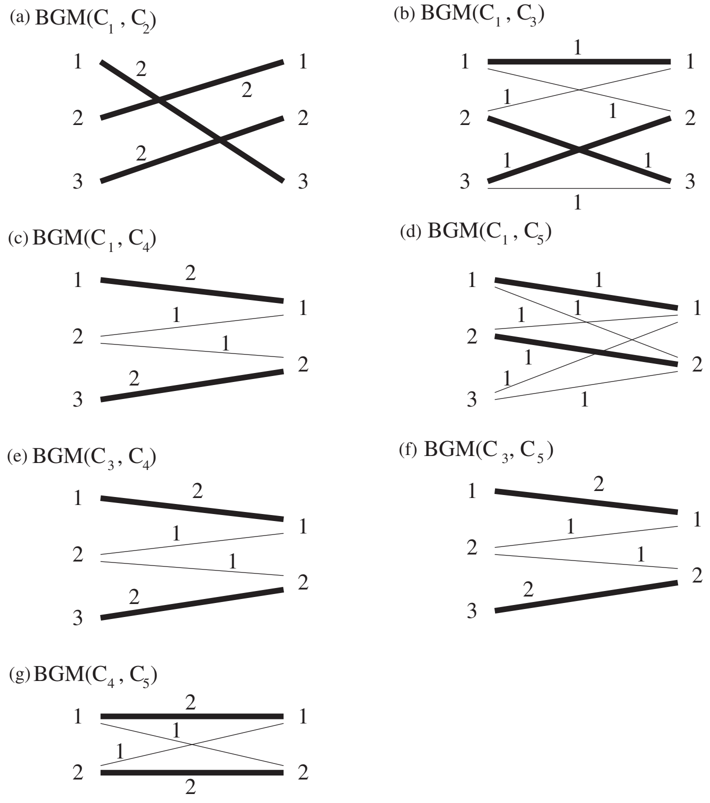


Fig. 7.4. Illustration of bipartite graph matching.

7.4 Novel Graph Distance Measures

Having the intragraph clustering algorithm of Section 7.2 and the cluster distance measures of Section 7.3 at our disposal, we can now combine both tools to obtain new distance measures for a pair of graphs g_1 and g_2. The basic idea of the measures proposed in the following is first to apply the algorithm of Section 7.2 to get a clustering C_1

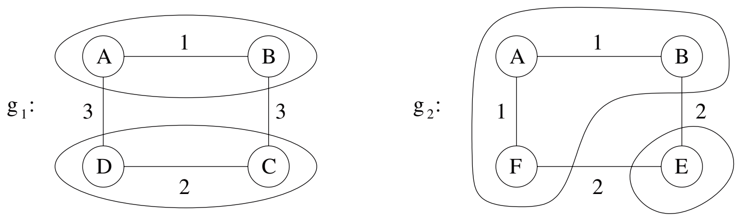


Fig. 7.5. An example (see text).

of the nodes of g_1, and a clustering C_2 of the nodes of g_2. Then any of the three cluster distance measures discussed in Section 7.3 can be used to get the distance $d(g_1, g_2)$ between g_1 and g_2. This yields three different new graph distance measures. Note that even more graph distance measures can be obtained if the mst-clustering procedure applied in the first step is replaced by another intragraph clustering procedure.

We assume that there is a common pool $V = V_1 \cup V_2$ from which the nodes of graphs $g_1 = (V_1, E_1, \beta_1)$ and $g_2 = (V_2, E_2, \beta_2)$ are chosen. Note, however, that we do not require $V_1 = V_2$. This means that in general, there will be nodes in g_1 that do not occur in g_2, and there will be nodes in g_2 that are not present in g_1. From this point of view, the situation is more general than the one considered in Section 7.3 because it was assumed there that both clusterings C_1 and C_2 are derived from the same set of objects. In the following we will discuss three extensions to the distance measures introduced in Section 7.3 that allow us to deal with this problem.

Assume that we are given two clusterings, $C_1 = \{c_{11}, c_{12}, \ldots, c_{1k}\}$ and $C_2 = \{c_{21}, c_{22}, \ldots, c_{2l}\}$ produced by our intragraph clustering algorithm, and we want to apply the Rand index $R(C_1, C_2)$ to measure the distance of C_1 and C_2. First we add a dummy cluster c_{10} to C_1. This cluster includes exactly the nodes that are present in V_2, but not in V_1. Similarly, we add a dummy cluster c_{20} to C_2, consisting of exactly the nodes that are present in V_1, but not in V_2. We can think of the nodes in cluster c_{20} as nodes that are deleted from g_1, while the nodes in cluster c_{10} can be understood as nodes that are inserted in g_2. Now consider the Rand index as defined in equation (7.1). In order to cope with the situation that V_1 may be different from V_2, we need to generalize the notion of a consistent pair. Any pair (x, y) with $x \in c_{10}$ or $y \in c_{10}$ (or both $x, y \in c_{10}$) will be considered inconsistent in our new setting. Moreover, any pair (x, y) with $x \in c_{20}$ or $y \in c_{20}$ (or both $x, y \in c_{20}$) will be considered inconsistent as well. This means that a consistent pair (x, y) needs to fulfill, in addition to the properties stated in Section 7.3.1, the condition that none of x and y is in c_{10} or c_{20}.

An example is shown in Figure 7.5. We assume that we run the intragraph clustering algorithm in such a way that it produces two clusters. Then the clusters represented by the ovals will be obtained. We have $c_{10} = \{E, F\}$, $c_{11} = \{A, B\}$, $c_{12} = \{C, D\}$, $c_{20} = \{C, D\}$, $c_{21} = \{A, B, F\}$, $c_{22} = \{E\}$. There are altogether 15 different pairs to be considered, out of which only one is consistent, namely pair (A, B). Hence $d(g_1, g_2) = 1 - 1/15 = 14/15$. So we get a relatively large distance value, which makes sense from the intuitive point of view since the two clusterings are in fact quite different.

For the mutual information-based clustering similarity measure defined by equations (7.3) and (7.4) we need to extend the table shown in Figure 7.2 by adding one row for dummy cluster c_{10}, and one column for dummy cluster c_{20}. Given the extended table, we propose to compute the distance measure in the same fashion as described in Section 7.3.2, but to carry out the summation in equations (7.3) and (7.4) only over the rows corresponding to $c_{11}, \ldots, c_{1k}$ and the columns corresponding to $c_{21}, \ldots, c_{2l}$. In this way elements from c_{10} and c_{20} will not contribute to the value of $NMI(C_1, C_2)$.

For the example shown in Figure 7.5 we get the table shown in Figure 7.6, where the entries correspond to absolute frequencies. From these numbers we derive $MI(C_1, C_2) = \frac{2}{6} \log \frac{2/6}{1/2 \cdot 1/3} = \frac{1}{3} \cdot \log 2$.

	CD	ABF	E	$\sum$
EF	0	1	1	2
AB	0	2	0	2
CD	2	0	0	2
$\sum$	2	3	1	6

Fig. 7.6. Table for mutual information computation.

For the bipartite graph-matching-based measure we do not need to add clusters c_{10} and c_{20} to the result of the intragraph clustering procedure, but we adjust the normalization factor n in equation (7.5). In Section 7.3.3 this factor was equal to the number of objects under consideration, i.e., $|V_1| = |V_2| = n$. In the generalized setting discussed in this section, we let $n = |V_1 \cup V_2|$, which means that all nodes occurring in graphs g_1 and g_2 are taken into consideration.

Using the example in Figure 7.5, we derive the bipartite weighted graph shown in Figure 7.7, from which we get $d(g_1, g_2) = 1 - 2/6 = 4/6$.

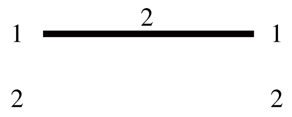


Fig. 7.7. Example of bipartite graph matching.

Next we provide a more extensive example using the bipartite graph matching scheme for measuring the distance of two clusterings. Four graphs, g_1, g_2, g_3, and g_4, are shown in Figures 7.8a,b,c,d. The bold edges represent the msts of these graphs. The clusters that result when all edges with a weight greater than or equal to 3 are deleted from the mst are graphically represented by ellipses.[2] For these four graphs, the bipartite graph matching measure yields

[2]In this example, the edge weights are not unique and several msts exist. Nevertheless, the resulting clusterings of all four graphs are uniquely defined.

$$\begin{aligned}
&d(g_1, g_2) = 0,\\
&d(g_1, g_3) = d(g_2, g_3) = 2/9,\\
&d(g_1, g_4) = d(g_2, g_4) = 1/2,\\
&d(g_3, g_4) = 4/9.
\end{aligned}$$

The bipartite matchings for $d(g_1, g_2)$, $d(g_1, g_3)$, $d(g_1, g_4)$, and $d(g_3, g_4)$ are shown in Figures 7.9a,b,c,d. We note that the bipartite matchings for $d(g_2, g_3)$ and $d(g_2, g_4)$ are identical to those for $d(g_1, g_3)$ and $d(g_1, g_4)$, respectively. In the computation of $d(g_1, g_2)$ and $d(g_1, g_3)$, a total of nine nodes are involved. Hence, for example, $d(g_1, g_3) = 1 - 7/9 = 2/9$, with 7 being the sum of the weights on the edges of the maximum weight bipartite graph shown in Figure 7.9b. For $d(g_1, g_4)$ we need to consider ten nodes, and for $d(g_3, g_4)$ nine nodes.

In the remainder of this section we formally characterize some properties of the new graph distance measures. For the following considerations we assume that the intragraph clustering procedure described in Section 7.2 is applied in such a way that it always returns the same number of clusters. That is, we always cut a fixed number of edges from the mst, rather than defining the edges to be cut by means of a threshold on the weight of the edges. Let $g_1 = (V_1, E_1, \beta_1)$ and $g_2 = (V_2, E_2, \beta_2)$ be graphs. Furthermore, let $E_1 = \{e_1, \ldots, e_k\}$ and $\beta_1(e_1) < \beta_1(e_2) < \cdots < \beta_1(e_k)$. We call graph g_2 a *scaled version* of g_1, symbolically $g_2 = \sigma(g_1)$, if $V_1 = V_2$, $E_1 = E_2$, and $\beta_2(e_i) = c \cdot \beta_1(e_i)$ for $i = 1, \ldots, k$, where $c > 0$ is a constant. Basically, g_1 and g_2 are the same up to a constant factor that is applied on each edge weight in g_1 to get the corresponding edge weight in g_2.

Lemma 7.6. *Let g be a graph and $\sigma(g)$ a scaled version of g. Then $d(g, \sigma(g)) = 0$.*

Proof. Let $g = g_1$ and $\sigma(g) = g_2$. Because each $\beta_2(e_i)$ is identical to $\beta_1(e_i)$ up to a constant scaling factor, the mst of g_1, which is uniquely defined because all edge weights are different by assumption, is also an mst of g_2. Hence cutting a fixed number of edges with the maximum weight from those msts will lead to identical clusterings in g_1 and g_2. Consequently, $d(g_1, g_2) = 0$.

Lemma 7.7. *Let g_1 and g_2 be graphs, and $\sigma(g_2)$ a scaled version of g_2. Then $d(g_1, g_2) = d(g_1, \sigma(g_2))$.*

Proof. Similarly to the proof of Lemma 7.6, we notice that the clustering obtained for g_2 will be the same as the clustering obtained for $\sigma(g_2)$. Hence $d(g_1, g_2) = d(g_1, \sigma(g_2))$.

Finally, we consider a transformation on the edge weights of a graph that is more general than scaling. Let $g_1 = (V_1, E_1, \beta_1)$ and $g_2 = (V_2, E_2, \beta_2)$ be graphs. Furthermore, let $E_1 = \{e_1, \ldots, e_k\}$ and $\beta_1(e_1) < \beta_1(e_2) < \cdots < \beta_1(e_k)$. We call g_2 an *order-preserved transformed version* of g_1, symbolically $g_2 = \rho(g_1)$, if $V_1 = V_2$, $E_1 = E_2$, and $\beta_2(e_1) < \beta_2(e_2) < \cdots < \beta_2(e_k)$.

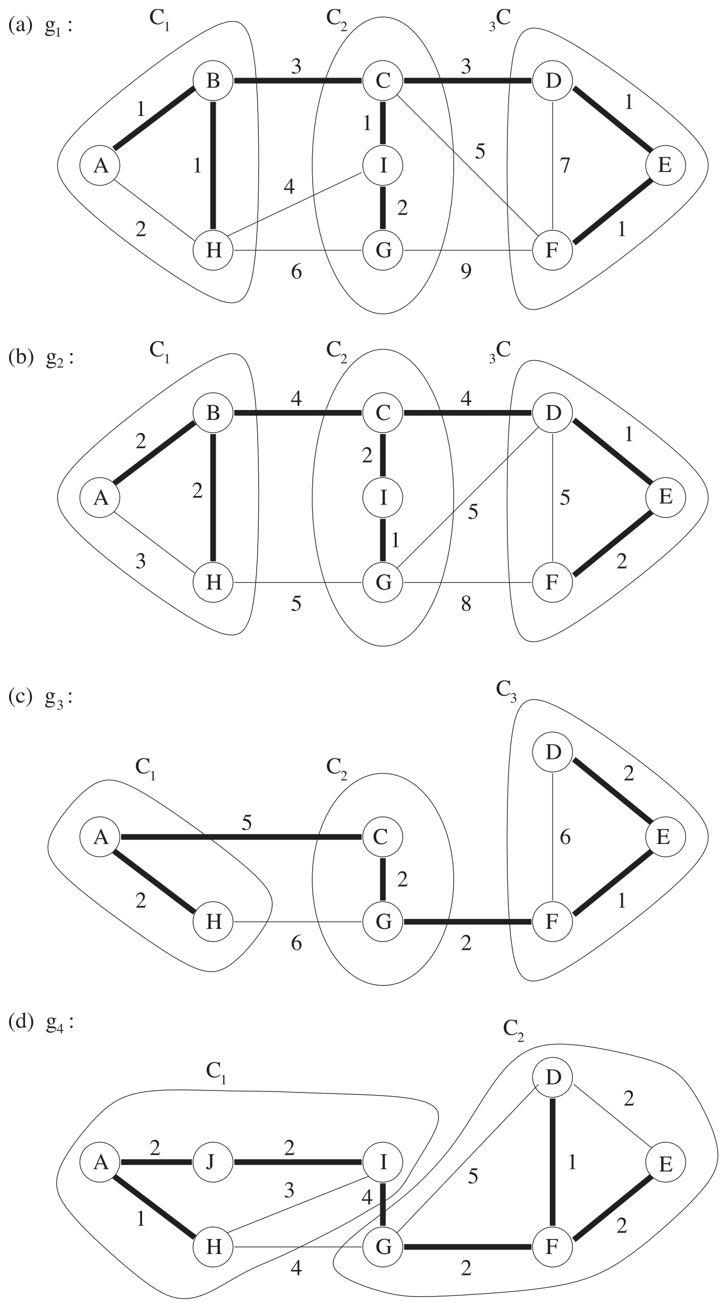


Fig. 7.8. Four graphs and their clusters.

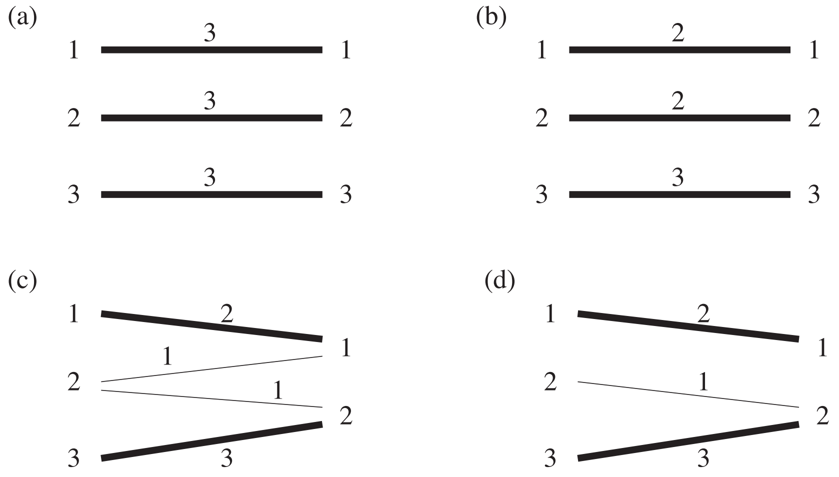


Fig. 7.9. Bipartite graph matching corresponding to (a) $d(g_1, g_2)$, (b) $d(g_1, g_3)$, (c) $d(g_1, g_4)$, (d) $d(g_3, g_4)$.

Theorem 7.8. *Let g_1 and g_2 be graphs and $\rho(g_2)$ an order preserved transformed version of g_2. Then*

(i) $d(g_1, \rho(g_1)) = 0$,
(ii) $d(g_1, g_2) = d(g_1, \rho(g_2))$.

The proof is based on arguments similar to the ones used for Lemmas 7.6 and 7.7. Clearly, Theorem 7.8 is a generalization of Lemmas 7.6 and 7.7 because any scaled version of a graph g is necessarily an order-preserved transformed version of g.

For an example of Theorem 7.8 consider Figure 7.10. Here we apply the intragraph clustering algorithm in such a way that it always produces two clusters. Clearly, g_2 is a scaled and therefore an order-preserved transformed version of g_1, and $d(g_1, g_2) = 0$. The distance of any other graph g to g_1 will always be the same as the distance of g to g_2, because g_1 and g_2 are clustered in the same way; hence $d(g, g_1) = d(g, g_2)$. For example, $d(g_1, g_3) = d(g_2, g_3) = 0.5$, using the bipartite graph-matching-based cluster similarity measure.

7.5 Applications to Computer Network Monitoring

Various graph distance measures and their application to computer network monitoring have been discussed in previous chapters of this book. It is expected that the measure proposed in this chapter will be a valuable addition to the repository of tools for measuring graph distance. A more detailed discussion of properties of the distance measures introduced in this chapter and their potential usefulness in the domain of computer network monitoring will be provided below.

In Chapter 5 two particular graph distance measures, d_1 and d_2, were proposed. Measure d_1 reflects only changes in the topology of a graph, but does not capture changes in edge weight. Hence, the insertion and deletion of both nodes and edges will be reported by this measure, but changes in edge weight, even large ones, remain undetected. As a generalization of d_1, measure d_2 captures not only topological changes, but also differences in edge weight. However, a potential shortcoming of this measure is its sensitivity to small edge weight changes. Any change of an edge weight, even a very small one, will have an impact on the graph distance value. The measure introduced in this chapter captures both topological and edge weight changes, but it has a certain degree of insensitivity with respect to small weight changes. Those changes are very common and may arise from regular "noise" present in the network under observation. The insensitivity with respect to small weight change comes from the fact that such changes are unlikely to change the clustering of the nodes. Clearly, if the clustering does not change, then a distance equal to zero will be reported by the new graph distance measure even if the two underlying graphs to be compared are not isomorphic.

The following example is provided to further demonstrate the differences between the graph distance measure $d(g_1, g_2)$ introduced in this chapter and $d_2(g_1, g_2)$ introduced in Chapter 5. Three graphs g_1, g_2, g_3 are shown in Figure 7.10a,b,c. It is easy to verify that $d_2(g_1, g_2) = 10$ and $d_2(g_1, g_3) = 8$. Thus g_3 is more similar to g_1 than g_2 to g_1 under measure d_2. Now assume that we run the intragraph clustering procedure described in Section 7.2 in such a way that it produces two clusters (i.e., only one edge, the one with maximum weight, is cut from the mst). Then we get the clusterings shown by the ellipses in Figure 7.10 and consequently $d(g_1, g_2) = 0$ and $d(g_1, g_3) = 0.5$. We notice that under the new graph distance measure, g_2 is more similar to g_1 than g_3 to g_1. By definition, measure d_2 merely sums up all edge weight differences, but the new measure also takes structural information about the underlying graphs into account. We observe that the change that leads from g_1 to g_2 is just a pure scaling change. That is, the smaller/larger relation between any pair of edges in the graph is maintained when we go from g_1 to g_2, since g_2 is identical to g_1 up to the fact that all edge weights have been multiplied by the same constant. As discussed above, the new measure d doesn't respond at all to such a scale change. However, it does indicate a change between g_1 and g_3. This change may be interpreted as a structural one, because in g_3 other node pairs more closely linked than in g_1.

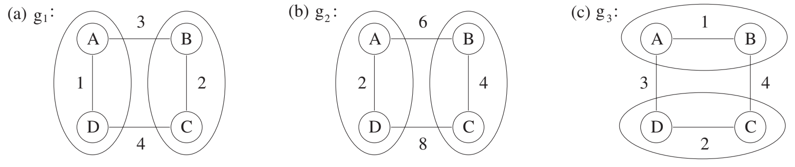


Fig. 7.10. An example.

Another very important feature of the graph distance measure introduced in this chapter is the fact that it may lend itself to useful visualization tools that allow a human operator to visually track the changes in a computer network. First of all, the clusters resulting from the first step of the new method can be readily displayed (as single nodes) on a screen, which allows us to view the network from a bird's-eye perspective. The level of detail, i.e., the number of clusters or, equivalently, their size, can be chosen by the user. It corresponds to the number of edges to be deleted from the mst, as described in Section 7.2. Secondly, if we use the bipartite graph matching procedure in the second step of the algorithm for computing graph distance, we can exactly see what happens to the individual clusters when we go from time t to $t + 1$. That is, we can see how the individual clusters of nodes develop over time. This allows a human operator not only to quantitatively measure network change, but also to qualitatively interpret these changes. Using the bipartite graph matching procedure introduced in Section 7.3.3 one can also highlight, for example, the k clusters that have undergone the largest change between time t and $t + 1$, where k is a user-defined parameter. Similarly, one can define a threshold T and display all clusters on the screen with a change larger than T. It is perhaps more difficult to implement similar visualization tools based on graph distance measures d_1, d_2, or others described in Chapter 4.

7.6 Conclusion

In this chapter we have proposed a set of new distance measures for graphs with unique node labels and weighted edges. These measures are based on a clustering procedure that partitions the set of nodes of a graph into clusters based on the edge weights, and algorithms to determine the distance of two given clusterings. We have also discussed potential applications of the new measures in the field of computer network monitoring and have pointed out that they may be a valuable addition to the repository of existing tools with some potential advantages over other graph distance measures.

In the second step of the new algorithm the distance of two given clusterings is evaluated. For this task, three different methods have been proposed. Also for the first step, i.e., intragraph clustering, several alternatives exist to the algorithm described in Section 7.2. Hence, what is described in the current chapter may be regarded as a novel general algorithmic framework rather than a single concrete procedure for graph distance computation.

8 Matching Sequences of Graphs

8.1 Introduction

In the current chapter we are going to address a new problem in the domain of graph matching. All algorithms for graph distance computation discussed previously in this book consider just a pair of graphs g and G, and derive their distance $d(g, G)$. Our proposed extension consists in computing the distance $d(s, S)$ of a pair of graph sequences $s = g_1, \ldots, g_n$ and $S = G_1, \ldots, G_m$. Both sequences can be of arbitrary length. In particular, the length of s can be different from the length of S. The normal graph distance is obtained as a special case of the proposed graph sequence distance if each s and S consists of only one graph. Similarly to any of the classical graph distance measures $d(g, G)$, the proposed graph sequence distance $d(s, S)$ will be equal to zero if s and S are the same, i.e., $n = m$ and $g_i = G_i$ for $i = 1, \ldots, n$. On the other hand, $d(s, S)$ will be large for two highly dissimilar sequences.

In the next section, we review basic concepts and procedures in sequence matching that will be used later in this chapter. In Section 8.3 we will bring the two concepts, classical sequence matching and graph matching, together and develop novel procedures for graph sequence matching. Applications of graph sequence matching, particularly in the field of network behavior analysis, will be discussed in Section 8.4. Finally, in Section 8.5 conclusions will be drawn.

8.2 Matching Sequences of Symbols

8.2.1 Preliminaries

We will consider sequences of symbols from some finite or infinite alphabet A. Such a sequence is given by

$$x = x_1 x_2 \ldots x_n ,$$

where $x_i \in A$ for $i = 1, 2, \ldots, n$. The integer n represents the *length* of sequence x, also denoted by $|x|$. A special case is the empty sequence ϵ that has length zero.

Sequences of symbols can be concatenated. Let $x = x_1 \ldots x_n$ and $y = y_1 \ldots y_m$ be sequences. Then the concatenation of x and y yields the sequence

$$z = xy = x_1 \ldots x_n y_1 \ldots y_m \, .$$

Let $x = x_1 \ldots x_n$ be a sequence. A *continuous subsequence* of x, $x_{k+1} \ldots x_{n-l}$, is obtained by deleting the first k symbols $x_1 \ldots x_k$ and the last l symbols $x_{n-l+1} \ldots x_n$ from x, where $l, k \geq 0$ and $l + k \leq n$. A *subsequence* of x is obtained by deleting up to n symbols from x at arbitrary positions. Hence any continuous subsequence is a subsequence but not vice versa. We call the continuous subsequence that consists of the first k symbols $x_1 \ldots x_k$ a *prefix*, and the sequence that consists of the last l symbols $x_{n-l+1} \ldots x_n$ a *suffix* of x, for $0 \leq k, l \leq n$.

Example 8.1. Let $A = \{a, b, c\}$, $x = aabca$, $y = cca$. We observe that $|x| = 5$ and $|y| = 3$. Concatenation yields, for example, $xy = aabcacca$, $yx = ccaaabca$, $yy = ccacca$, $xxy = aabcaaabcacca$. Symbol a is a prefix of x and a suffix of y. Sequences aa and aab are both a prefix of x. All of the following sequences are continuous subsequences of x : $a, ab, abc, aa, aabc$. The set of all continuous subsequences of y is $\{c, cc, cca, ca, a\}$. All of the following sequences are subsequences of x : ac, ba, aba, aca, aaa, but none is a continuous subsequence.

Given two sequences $x = x_1 \ldots x_n$ and $y = y_1 \ldots y_m$, it is trivial to check their identity: we just need to find out whether $n = m$ and $x_i = y_i$ for $i = 1, \ldots, n$.

There is also a simple algorithm to find out whether x is a continuous subsequence of y; see Figure 8.1. We assume that $n \leq m$ (otherwise, x can't be a continuous subsequence of y). The algorithm considers each symbol y_i in sequence y as a potential starting point of sequence x, for $i = 1, \ldots, m - n + 1$. It compares symbols y_{i+j} and x_j for $j = 1, \ldots, n$. If both symbols are the same then the algorithm continues and compares the next pair of symbols, y_{i+j+1} and x_{j+1}. Otherwise, if a position j is encountered for which $y_{i+j} \neq x_j$, the algorithm continues with the next symbol of y, y_{i+1}, as a potential starting point of sequence x. If there exists a symbol y_i such that $y_{i+j} = x_j$ for $j = 1, \ldots, n$ then we can conclude that x is in fact a continuous subsequence of y. As an example, the reader may execute the algorithm shown in Figure 8.1 with $x = ab$ and $y = aaab$.

Obviously, the algorithm shown in Figure 8.1 runs in $O(nm)$ time and $O(m)$ space. There exist more elaborate procedures that require less time [163]. A procedure similar to the one given above can be designed to find out whether x is a subsequence of y. Another algorithm that solves this task will be introduced in Section 8.2.

8.2.2 Edit Distance of Sequences of Symbols

Edit distance is a concept that is useful to measure the distance $d(x, y)$ of two given sequences x and y. The basic idea in sequence edit distance computation is to introduce a set of edit operations that operate on the symbols of x and y. Typically the set of edit operations consists of the deletion, insertion, and substitution of a symbol. More formally, an edit operation is one of the following:

(a) $a \to \epsilon$ denotes the deletion of symbol a
(b) $\epsilon \to a$ denotes the insertion of symbol a
(c) $a \to b$ denotes the substitution of symbol a by b

Often a cost is assigned to each individual edit operation. Let $c(a \to \epsilon)$, $c(\epsilon \to a)$, and $c(a \to b)$ denote the cost assigned to edit operations $a \to \epsilon$, $\epsilon \to a$, and $a \to b$, respectively. The costs are usually application-dependent and are chosen in such a way that frequent operations have a low cost, while a high cost is assigned to operations that occur seldomly. Regardless of the application, each cost is a nonnegative number, and the cost of an identical substitution $c(a \to a)$ is always equal to zero for any symbol a.

Given two sequences x and y, the aim of sequence matching is to apply edit operations so as to make the two sequences identical to each other. In general, there exist many possibilities, i.e., many possible sequences of edit operations, to achieve this goal. For example, one can first delete all symbols from x and then insert all symbols from y. Among all possible sequences of edit operations that transform x into y we are particularly interested in the one that has minimum cost. Let σ be such a sequence with minimum cost (note that there may exist several such minimum cost sequences), and let $c(\sigma)$ denote the cost of σ. Then the edit distance $d(x, y)$ of x and y is equal to $c(\sigma)$. Formally, we define

$$d(x, y) = \min\{c(\sigma) | \sigma \text{ is a sequence of edit operations that transform } x \text{ into } y\} \ . \tag{8.1}$$

In this definition we assume that the cost of a sequence of edit operations is equal to the sum of the individual costs. That is, let $\sigma = e_1, \ldots, e_l$ be a sequence of edit operations, where edit operation e_i has cost $c(e_i)$ for $i = 1, \ldots, l$. Then $c(\sigma) = \sum_{i=1}^{l} c(e_i)$.

We note that equation (8.1) is just a formal definition of the edit distance $d(x, y)$ of two sequences x and y. It doesn't tell us how to actually compute $d(x, y)$ for a given pair of sequences x and y. The standard algorithm for computing $d(x, y)$ is based on dynamic programming. A pseudocode description of the algorithm is given in Figure 8.2 [176].

The algorithm uses a matrix $D(i, j)$, which is also called *cost matrix*, because its elements represent the cost of certain sequences of edit operations. If $|x| = n$ and $|y| = m$ then $D(i, j)$ is of dimension $(n+1) \times (m+1)$. There is one row for each symbol in x and one column for each symbol in y. Additionally there is one initial row and one initial column, both with index 0. Matrix element $D(0, 0)$ is equal to zero, by definition. Furthermore, $D(0, j)$ holds the edit distance $d(\epsilon, y_1 \ldots y_j)$ and matrix element $D(i, 0)$ the edit distance $D(x_1 \ldots x_i, \epsilon)$ for $i = 1, \ldots, n$ and $j = 1, \ldots, m$. For $i > 0$ and $j > 0$, $D(i, j)$ holds the edit distance $d(x_1 \ldots x_i, y_1 \ldots y_j)$. Consequently, element $D(n, m)$, which is obtained immediately before the algorithm terminates, holds the desired edit distance between x and y, $d(x, y) = d(x_1 \ldots x_n, y_1 \ldots y_m)$. An illustration of the cost matrix is shown in Figure 8.3.

During execution of the algorithm, the initial row and initial column are computed first. Then the interior matrix elements are computed in a left-to-right and top-to-bottom fashion. In the computation of matrix element $D(i, j)$ for $i > 0$ and $j > 0$ its three immediate predecessor elements, $D(i - 1, j - 1)$, $D(i - 1, j)$, and $D(i, j - 1)$, are

```
input:  $x = x_1 \dots x_n$, $y = y_1 \dots y_m$
output: success, if $x$ is a continuous subsequence of $y$;
        failure, otherwise.

begin
$i = 1$
$j = 1$
while $i \leq m$ and $j \leq n$ do
    if $x_j = y_i$ then
            $i = i + 1$
            $j = j + 1$
    else
            $i = i - j + 2$
            $j = 1$
if $j > n$ then
        output(success: $x_1 \dots x_n = y_{i-n} \dots y_{i-1}$)
        else
        output(failure)
end
```

Fig. 8.1. Algorithm for finding a continuous subsequence x in sequence y.

considered. The value of $D(i, j)$ is defined to be equal to the minimum of the following three quantities: (a) $D(i-1, j-1)$ plus the cost of substituting x_i by y_j, (b) $D(i-1, j)$ plus the cost of deleting x_i, (c) $D(i, j-1)$ plus the cost of inserting y_j. Hence the value of $D(i, j)$, which represents $d(x_1 \dots x_i, y_1 \dots y_j)$, is obtained by selecting the minimum among of the following three terms:

(a) $d(x_1 \dots x_{i-1}, y_1 \dots y_{j-1}) + c(x_i \rightarrow y_j)$
(b) $d(x_1 \dots x_{i-1}, y_1 \dots y_j) + c(x_i \rightarrow \epsilon)$
(c) $d(x_1 \dots x_i, y_1 \dots y_{j-1}) + c(\epsilon \rightarrow y_j)$

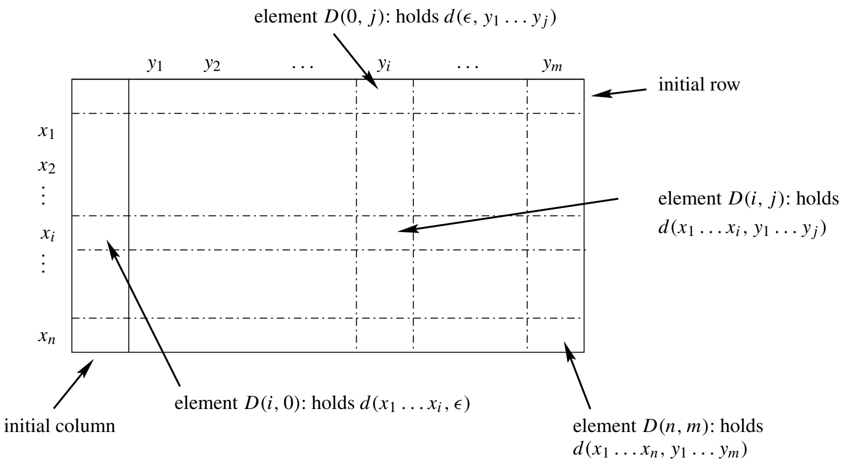
Matrix element $D(n, m)$ will become available last, and it holds the desired value $d(x, y)$. For a proof of the correctness of this algorithm see [176], for example.

From the pointers set in steps 10 to 12 the sequence of edit operations that actually transform x into y can be reconstructed. For this purpose one starts at $D(n, m)$ and traces back these pointers to position $D(0, 0)$. Such a sequence of pointers is also called an *optimal path* in the cost matrix. It has to be noted that, due to the way the algorithm is presented in Figure 8.2, only one pointer is installed for each matrix element. In general, however, the predecessor element from which $D(i, j)$ is computed may not be uniquely defined. In other words, the minimum among (a), (b), and (c) as introduced above may not be unique. If one is interested in recovering all possible sequences of edit operations

```
input:  $x = x_1 \dots x_n$, $y = y_1 \dots y_m$, set of edit operations and their costs
output: cost matrix $D(i, j)$ and pointers; $d(x, y) = D(n, m)$.

begin
1.  $D(0, 0) := 0$;
2.  for $i = 1$ to $n$ do $D(i, 0) := D(i-1, 0) + c(x_i \to \epsilon)$;
3.  for $j = 1$ to $m$ do $D(0, j) := D(0, j-1) + c(\epsilon \to y_j)$;
4.  for $i = 1$ to $n$ do
5.      for $j = 1$ to $m$ do
6.          $m_1 := D(i-1, j-1) + c(x_i \to y_j)$;
7.          $m_2 := D(i-1, j) + c(x_i \to \epsilon)$;
8.          $m_3 := D(i, j-1) + c(\epsilon \to y_j)$;
9.          $D(i, j) = \min(m_1, m_2, m_3)$;
10.         if $m_1 = D(i, j)$ then pointer$(i, j)$:=$(i-1, j-1)$
11.                         else if $m_2 = D(i, j)$ then pointer$(i, j)$:=$(i-1, j)$
12.                                            else pointer$(i, j)$:=$(i, j-1)$;
end
```

Fig. 8.2. Algorithm for edit distance $d(x, y)$ of sequences x and y.



element $D(0, j)$: holds $d(\epsilon, y_1 \dots y_j)$
y_1 y_2 $\dots$ y_i $\dots$ y_m
initial row
x_1
x_2
x_i
x_n
element $D(i, j)$: holds $d(x_1 \dots x_i, y_1 \dots y_j)$
element $D(i, 0)$: holds $d(x_1 \dots x_i, \epsilon)$
initial column
element $D(n, m)$: holds $d(x_1 \dots x_n, y_1 \dots y_m)$

Fig. 8.3. Graphical illustration of cost matrix $D(i, j)$.

that transform x into y with minimum cost, one has to install and track all these pointers.

Example 8.2. Let $x = ababbb$, $y = babaaa$ and let the set of edit operations be given by $a \to b$, $b \to a$, $a \to \epsilon$, $b \to \epsilon$, $\epsilon \to a$, $\epsilon \to b$ with costs $c(a \to \epsilon) = c(b \to \epsilon) = c(\epsilon \to a) = c(\epsilon \to b) = 1$ and $c(a \to b) = c(b \to a) = 2$. Then the execution of the algorithm yields the matrix shown in Figure 8.4. From the element in the lower right corner we conclude that $d(x, y) = 6$.

$$\begin{array}{c|ccccccc|}
 & & \mathrm{b} & \mathrm{a} & \mathrm{b} & \mathrm{a} & \mathrm{a} & \mathrm{a} \\
\hline
 & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
\mathrm{a} & 1 & 2 & 1 & 2 & 3 & 4 & 5 \\
\mathrm{b} & 2 & 1 & 2 & 1 & 2 & 3 & 4 \\
\mathrm{a} & 3 & 2 & 1 & 2 & 1 & 2 & 3 \\
\mathrm{b} & 4 & 3 & 2 & 1 & 2 & 3 & 4 \\
\mathrm{b} & 5 & 4 & 3 & 2 & 3 & 4 & 5 \\
\mathrm{b} & 6 & 5 & 4 & 3 & 4 & 5 & 6 \\
\hline
\end{array}$$

Fig. 8.4. Cost matrix corresponding to Example 8.2.

For the sake of simplicity, pointers have been left out in Figure 8.4. However, the path that can be actually reconstructed from the pointers is shown in Figure 8.5. This path corresponds to the following edit operations:

$$(\epsilon \to b), (a \to a), (b \to b), (a \to a), (b \to \epsilon), (b \to a), (b \to a).$$

Application of this sequence of edit operations to x yields in fact sequence y, and the cost of this sequence of edit operations is equal to 6.

$$\begin{array}{c|ccccccc|}
 & & \mathrm{b} & \mathrm{a} & \mathrm{b} & \mathrm{a} & \mathrm{a} & \mathrm{a} \\
\hline
 & \boxed{0} & \boxed{1} & 2 & 3 & 4 & 5 & 6 \\
\mathrm{a} & 1 & 2 & \boxed{1} & 2 & 3 & 4 & 5 \\
\mathrm{b} & 2 & 1 & 2 & \boxed{1} & 2 & 3 & 4 \\
\mathrm{a} & 3 & 2 & 1 & 2 & \boxed{1} & 2 & 3 \\
\mathrm{b} & 4 & 3 & 2 & 1 & \boxed{2} & 3 & 4 \\
\mathrm{b} & 5 & 4 & 3 & 2 & 3 & \boxed{4} & 5 \\
\mathrm{b} & 6 & 5 & 4 & 3 & 4 & 5 & \boxed{6} \\
\hline
\end{array}$$

Fig. 8.5. Illustration of pointers.

An interesting problem that is closely related to the edit distance of sequences as introduced in this section is longest common subsequence computation. Let x and y be two sequences and z a subsequence of both x and y. We call z a *longest common subsequence* (*lcs*) of x and y if there exists no other subsequence of x and y that is longer than z. Notice that there may exist more than one lcs of two given sequences x and y. It has been shown that the algorithm for string edit distance computation given in Figure 8.2 can be used for lcs computation if the following costs are chosen: $c(a \to b) = 2$ for any substitution $a \to b$, where $a \neq b$, $c(\epsilon \to a) = c(a \to \epsilon) = 1$ for any insertion and deletion [176]. In this case the length l of the lcs of two sequences x and y is related to the edit distance of x and y via the equation

$$d(x, y) = |x| + |y| - 2l \ , \tag{8.2}$$

and an actual lcs corresponds to the identical substitutions on an optimal path in the cost matrix.

Example 8.3. Consider again Example 8.2. Here the costs comply with the condition stated above. Hence we conclude that $l = [|x|+|y|-d(x, y)]/2 = [6+6-6]/2 = 3$, i.e., the lcs of x and y is of length 3. Moreover, from the optimal path shown in Figure 8.5 we extract three identical substitutions, namely $(a \to a)$, $(b \to b)$, $(a \to a)$, which correspond to the lcs aba. Note that in this example the lcs is not unique.

Obviously, the algorithm for lcs computation can be used to find out whether a given sequence s is a subsequence of another given sequence S. We just run the algorithm and check whether $|s| = l$, where l is the length of the lcs of s and S according to equation (8.2).

It is easy to see that the algorithm given in Figure 8.2 has a time and space complexity equal to $O(nm)$. We want to remark that the present section provides only a basic introduction to the field of sequence matching. For further details, including algorithms with a lower time or space complexity, the reader is referred to [80, 152, 163].

8.3 Graph Sequence Matching

In Section 8.2 we have discussed how two sequences of symbols can be tested for identity, and the relations of subsequence and continuous subsequence. Also, a procedure for the computation of the edit distance of a pair of sequences was introduced. In the current section we show how these procedures can be extended from sequences of symbols to sequences of graphs. Throughout the section we consider two sequences of graphs, $s = g_1 \dots g_n$ and $S = G_1 \dots G_m$, that are to be matched to each other.

A closer look reveals that all algorithms considered in Section 8.2 can be extended to deal with sequences of graphs rather than sequences of symbols in a straightforward manner. First of all, we notice that checking two sequences of graphs for identity can be accomplished by testing whether $n = m$ and g_i and G_i are isomorphic to each other, for $i = 1, \dots, m$. Furthermore, the algorithm given in Figure 8.1, which tests two sequences of symbols for the continuous subsequence relation, can be trivially extended to test whether one sequence of graphs is a continuous subsequence of another if we replace the operation that tests two symbols for identity by an operation that tests two graphs for isomorphism.

Moreover, we can generalize the algorithm for edit distance computation in Figure 8.2 from sequences of symbols to sequences of graphs in a straightforward way. The only extension that is needed is to define the edit operations, which are applied on symbols in Section 8.2, to work on whole graphs. Hence our proposed graph sequence matching algorithm will be based on the following edit operations:

(a) $g \to G$, which denotes the substitution of graph g by graph G
(b) $g \to \lambda$, which denotes the deletion of graph g (or, equivalently, the substitution of graph g by the empty graph λ)
(c) $\lambda \to g$, which denotes the insertion of graph g (or, equivalently, the substitution of the empty graph λ by graph g)

Given these edit operations and associated costs, we can adapt the algorithm of Figure 8.2 for edit distance computation of graph sequences. A pseudocode description of the resulting algorithm is given in Figure 8.6.

The costs of the edit operations used in the algorithm in Figure 8.6 depend, similarly to the case of matching sequences of symbols, on the considered application. One obvious choice is, for example, to assign a constant cost to each type of edit operation, regardless of the affected graphs. As a concrete example, one may define $c(g \to G) = 2$ for any substitution and $c(g \to \lambda) = c(\lambda \to G) = 1$ for any deletion and insertion. Under such a cost function, the algorithm shown in Figure 8.6 can be used not only for graph sequence computation but also to compute the length of the longest common subsequence of two sequences of graphs (see Section 8.2 and equation (8.2)).

Another meaningful definition of the costs of the edit operations is the edit distance $d(g, G)$ of the affected graphs g and G. Under this scheme, we define

(a) $c(g \to G) = d(g, G)$,
(b) $c(g \to \lambda) = d(g, \lambda)$,
(c) $c(\lambda \to G) = d(\lambda, G)$,

where any of the graph distance measures introduced in previous chapters can be used to implement $d(g, G)$.

It is easy to see that the proposed algorithm for computing the edit distance of two sequences of graphs has a time and space complexity of $O(nmk)$, where k is the complexity of determining the cost of an individual edit operation. For example, when a constant cost is assigned to each edit operation, as discussed above, then factor k reflects the complexity of testing two graphs for isomorphism. Similarly, when the costs of the edit operations are equal to the distance of the corresponding graphs, then factor k corresponds to the complexity of graph distance computation. In the applications discussed in Section 8.4 this factor will be no larger than p^2 thanks to the fact that the class of graphs under consideration is restricted, where p is the number of nodes of the larger of the two graphs involved. Hence a polynomial complexity with respect to the size of the underlying graphs and the length of the graph sequences is ensured.

```
input:  graph sequences $s = g_1 \dots g_n$ and $S = G_1 \dots G_m$, together with
        edit operations and their costs.
output: cost matrix $D(i, j)$ and pointers; $d(s, S) = D(n, m)$.

begin
1.  $D(0, 0) := 0$;
2.  for $i = 1$ to $n$ do $D(i, 0) := D(i-1, 0) + c(g_i \to \epsilon)$;
3.  for $j = 1$ to $m$ do $D(0, j) := D(0, j-1) + c(\epsilon \to G_j)$;
4.  for $i = 1$ to $n$ do
5.     for $j = 1$ to $m$ do
6.        $m_1 := D(i-1, j-1) + c(g_i \to G_j)$;
7.        $m_2 := D(i-1, j) + c(g_i \to \epsilon)$;
8.        $m_3 := D(i, j-1) + c(\epsilon \to G_j)$;
9.        $D(i, j) = \min(m_1, m_2, m_3)$;
10.       if $m_1 = D(i, j)$ then pointer$(i, j)$:=$(i-1, j-1)$
11.                 else if $m_2 = D(i, j)$ then pointer$(i, j)$:=$(i-1, j)$
12.                                   else pointer$(i, j)$:=$(i, j-1)$;
end
```

Fig. 8.6. Algorithm for edit distance, $d(s, S)$, of graph sequences $s = g_1 \dots g_n$ and $S = G_1 \dots G_m$.

8.4 Applications in Network Behavior Analysis

With new algorithms for graph sequence distance computation at our disposal, a number of additional, more refined, procedures for network behavior analysis can be designed. Three possibilities are discussed below.

8.4.1 Anomalous Event Detection Using a Library of Past Time Series

The basic idea of this approach is to have a database of past time series of graphs available with some of these time series including anomalous events. More precisely, for each sequence $S = G_1 \dots G_m$ in the database and each transition from G_i to G_{i+1} it is known whether this transition corresponds to an anomalous event. If the transition is known to be abnormal, it may also carry a label that further specifies of which type the abnormal event is. Now assume we are given an actual time series of graphs $s = g_1 \dots g_n$ and want to find out whether an abnormal event has occurred in the last time step, i.e., between g_{n-1} and g_n. Rather than just computing $d(g_{n-1}, g_n)$ and comparing this value to a predefined threshold, with the new graph sequence matching algorithms at our disposal, we can match the whole time series s, or a suffix of suitable length, to the sequences in the database, retrieve all sequences that have a small distance to s, and decide based on the type of these similar sequences whether the transition from g_{n-1} to g_n is abnormal. More precisely, if we retrieve sequence $S = G_1 \dots G_m$ from the database such that the pair G_{i-1}, G_i corresponds to the pair g_{n-1}, g_n, and if furthermore the transition from G_{i-1} to G_i is known to be anomalous, then we may conclude that also the transition from g_{n-1} to g_n is anomalous. Such a strategy, which compares actual input data to past events of known type, follows the paradigm of case-based reasoning, where past cases are used to solve an actual problem. For an introduction to, and overview of, case-based reasoning, see [107].

In the rest of this section we will discuss the strategy that was sketched in the last paragraph in greater detail. The first question that arises in the implementation of a scheme for anomalous change detection using a library of past cases concerns the length of sequence s. We assume here that there exist some deterministic, but unknown, rules that partly govern the network behavior. In particular, we assume that whether the transition from g_{n-1} to g_n is anomalous depends not only on g_{n-1} and g_n, but on some longer, but finite, history, i.e., on a suffix of sequence s of length $t \geq 2$. Finding an appropriate value of the relevant length t of the postfix of s to be used to decide whether the transition from g_{n-1} to g_n represents an anomalous event is the first important step to be solved. Clearly, if the value of t is too small then we may not have enough information to decide of which type the event leading from g_{n-1} to g_n is. On the other hand, if t is too large we may unnecessarily slow down the computation or introduce noise. In order to simplify our notation, we assume that $n = t$, i.e., we assume that the whole length of sequence s is relevant to classifying the transition from g_{n-1} to g_n as anomalous or not. The actual choice of a suitable value of parameter t will usually need some prior knowledge that depends on the actual application, and may require some experimental investigation.

Given a fixed value of parameter t, let us furthermore assume that we know that the transition from G_{i-1} to G_i in sequence S of the database is anomalous. Then we can compute the distance of $s = g_1 \ldots g_n$ and $G_{i-n+1} \ldots G_i$. If this distance $d(g_1 \ldots g_n, G_{i-n+1} \ldots G_i)$ is smaller than a given threshold θ we may conclude that the transition from g_{n-1} to g_n is anomalous. Moreover, we may conclude that the abnormality is of the same type as the one corresponding to the transition from G_{i-1} to G_i. Hence detection of anomalous events can be solved by means of the graph sequence matching algorithm introduced in this chapter. If there is more than one anomalous event recorded in the database, then all of these events can be used for comparison to sequence s. If more than one anomalous event from the database matches the actual sequence, then the closest match, i.e., the event corresponding to the sequence with the smallest distance, can be used to determine the type of abnormality.

When matching sequence $g_1 \ldots g_n$ with $G_{i-n+1} \ldots G_i$, we can either disable insertions and deletions or allow them, depending on the particular application. Disabling insertions and deletions can be easily accomplished by defining the costs of these edit operations to equal infinity. If insertions and deletions are disabled, then the two sequences under consideration must be of the same length, and computing the edit distance of two sequences of equal length, $s = g_1 \ldots g_n$ and $S = G_1 \ldots G_n$, reduces to computing the sum of the substitution costs $c(g_i, G_i)$, i.e., in this case

$$d(g_1 \dots g_n, G_1 \dots G_n) = \sum_{i=1}^{n} c(g_i, G_i) \, .$$

Note that this operation requires only $\mathcal{O}(nk)$ time, where parameter k represents the cost of computing $c(g_i, G_i)$.

If deletions and insertions are enabled, one may consider a time window longer than n in the sequences in the database to be matched with the input sequence s. That is, if $s = g_1 \dots g_n$ and the transition from G_{i-1} to G_i in database sequence S is known to be anomalous, then we may wish to compute $d(g_1 \dots g_n, G_{i-N+1} \dots G_i)$ rather than $d(g_1 \dots g_n, G_{i-n+1} \dots G_i)$ for some $N > n$. Again, finding an appropriate value of parameter N may require a priori knowledge about the problem domain and experimental investigation.

8.4.2 Prediction of Anomalous Events

We assume again that a database with past time series of graphs is at our disposal, where it is known for each transition from G_{i-1} to G_i in each sequence S whether the transition represents an anomalous event. Given an actual time series of graphs $s = g_1 \dots g_n$, we consider the task of predicting whether the transition from g_n to g_{n+1} will be anomalous. It is assumed, similarly to Section 8.4.1, that the behavior of the network to be monitored is to some degree deterministic, though the concrete rules that govern the network's behavior are unknown. Consequently, we may conclude that an anomalous event is likely to occur if there exists a sequence $S = G_1 \dots G_m$ in the database that contains a (contiguous) subsequence $G_{i-n+1} \dots G_i$ that is similar to s, and the transition from G_i to G_{i+1} represents an anomalous event. This strategy is based on the assumption that, given two sequences s and S, if the first n graphs are similar in both sequences, the elements at position $n + 1$ will also be similar.

To implement a scheme for prediction as sketched in the last paragraph, we need to fix again parameter t that defines the length of the relevant time window. This parameter has the same meaning as in Section 8.4.1. In order to simplify our notation we assume again that $t = n$. In analogy to Section 8.4.1 let the transition from G_i to G_{i+1} represent an anomalous event. Then we compute the distance $d(g_1 \dots g_n, G_{i-n+1} \dots G_i)$ and predict that the transition from g_n to g_{n+1} will be an anomalous one if this distance is smaller than some predefined threshold.

Similarly to Section 8.4.1, two different instances of this scheme seem feasible. We may choose either to disable or enable insertions and deletions. In the second case, it may be meaningful to consider a time window in the sequences in the database that is longer than n.

8.4.3 Recovery of Incomplete Network Knowledge

For a number of reasons we may sometimes have to deal with incomplete time series of graphs, where at certain points in time no graphs are available. For the sake of simplicity we consider just the case of a simple gap of length one in a given time series of graphs. That is, we assume that an actual time series of graphs $s = g_1 \dots g_{i-1} \emptyset g_{i+1} \dots g_n$ is given, where we have observed a graph at each point in time $1, \dots, i-1, i+1, \dots, n$, but we missed the observation at time i. The procedure described in the following can be easily extended from one to several gaps in a sequence, and from gaps of length one to gaps of length $r \geq 1$. The task considered in this subsection consists in recovering the state of the network at point i, i.e., in recovering graph g_i from $g_1 \dots g_{i-1} g_{i+1} \dots g_n$. For the procedure described below we assume that the database of past time series includes only sequences without gaps.

One possible procedure to recover graph g_i is to split incomplete sequence $s = g_1 \dots g_{i-1} \emptyset g_{i+1} \dots g_n$ into two continuous subsequences, $s_1 = g_1 \dots g_{i-1}$ and $s_2 = g_{i+1} \dots g_n$. Then we try to find a sequence $S = G_1 \dots G_m$ in the database that contains a subsequence $S_1 = G_{j+1} \dots G_{j+i-1}$ and another subsequence $S_2 = G_{j+i+1} \dots G_{j+n}$ such that s_1 matches S_1 well and s_2 matches S_2 well, i.e., both $d(s_1, S_1)$ and $d(s_2, S_2)$ are small. If two such subsequences exist, we can use the graph in between S_1 and S_2, i.e., G_{j+i}, to substitute it for the missing observation in sequence s. This procedure is based on the assumption that, given two sequences s and S, if the first i and the last j graphs are similar in both sequences, the elements in between will also be similar.

In the implementation of the proposed procedure we need to decide upon the lengths of sequences s_1 and s_2. To keep our notation simple we let s_1 and s_2 be of lengths $i-1$ and $n-i$, respectively, i.e., $s_1 = g_1 \dots g_{i-1}$ and $s_2 = g_{i+1} \dots g_n$, as introduced above. If we decide to disable insertions and deletions then there exists a simple solution that checks whether database sequence $S = G_1 \dots G_m$ contains sequences $S_1 = G_{j+1} \dots G_{j+i-1}$ and $S_2 = G_{j+i+1} \dots G_{j+n}$ such that s_1 has a small distance to S_1 and s_2 has a small distance to S_2. We assume that $n \leq m$ and compute

$$\begin{aligned} &d(g_1 \dots g_{i-1} \gamma g_{i+1} \dots g_n, G_{j+1} \dots G_{j+i-1} G_{j+i} G_{j+i+1} \dots G_{j+n}) \\ &\quad = \sum_{k=1}^{n} c(g_k, G_{j+k}) \ , \end{aligned}$$

for $j = 0, \dots, m-n+1$. Symbol $\gamma = g_i$ is a dummy symbol for which we define $c(\gamma \to g) = 0$ for any graph g. Clearly, whenever there exists a subsequence $G_{j+1} \dots G_{j+i-1} G_{j+i}\ G_{j+i+1} \dots G_{j+n}$ for which $d(g_1 \dots g_{i-1} \gamma g_{i+1} \dots g_n, G_{j+1} \dots G_{j+i-1} G_{j+i} G_{j+i+1} \dots G_{j+n})$ is smaller than some predefined threshold, graph G_{j+i} qualifies to be used as a suitable substitute for the missing graph g_i.

8.5 Conclusions

In this chapter we have proposed a novel class of algorithms for graph sequence matching. While both the matching of graphs and the matching of sequences of symbols have been individually addressed in the literature before, this is the first attempt to combine both concepts into a common framework that allows us to compare sequences in which the individual elements are given in terms of graphs. Our novel algorithms are able to decide, for two given sequences of graphs s and S, whether s is a (continuous) subsequence of S. Moreover, we have introduced a procedure that is able to compute the distance of two graph sequences based on a set of edit operations, each of which affects a complete graph in one of the two given sequences. Also, some applications of the proposed algorithms in the analysis of network behavior have been discussed. Compared to previous work in network analysis that uses the same kind of graph representation as in this chapter, it can be expected that the proposed algorithms will be able to be distinguished by increased robustness and precision in the detection of anomalous events, since they take more information into account when classifying an event as normal or abnormal. Moreover, there are a number of novel tasks in network analysis that can be addressed, thanks to the improved capabilities of the new tools proposed in this report. Examples include prediction of anomalous behavior and recovery of missing information. One crucial requirement, however, is the availability of a sufficiently large database that represents typical network behavior of both normal and abnormal type.

In Section 8.4.3 we have shown how a missing graph in a time series can be recovered. A variation of this task consists in the reconstruction of partial graphs. Here we assume that at time i only a partial graph g_i was observed and we are interested in reconstructing the missing portion of graph g_i. Obviously, this problem can be solved by the method described in Section 8.4.3 if we don't distinguish whether graph g_i is completely missing or was only partially observed. Another possible solution is, however, to apply some kind of graph interpolation operator that would reconstruct g_i from g_{i-1} and g_{i+1}, or from a larger window. That is, under this scheme we would not need any of the graph sequence stored in the database but rely only on graphs in the actual input sequence. This topic will be discussed in greater detail in Chapter 11.

Another open question is whether more elaborate versions of the algorithms proposed in this chapter can be developed that would be able to find appropriate values of some of the involved parameters automatically. For example, in all of the methods discussed in Section 8.4 it is necessary for the user, or the system developer, to fix the length t of the relevant sequences or subsequences that are to be taken into account in comparing the actual times series of graphs with the cases stored in the database. It would be an advantage from the user's point of view if appropriate values of this parameter could be automatically found by our graph sequence matching algorithm.

Part III

Properties of the Underlying Graphs

9

Distances, Clustering, and Small Worlds

9.1 Graph Functions

9.1.1 Distance

In previous chapters we have discussed distance between *graphs*. In this chapter we shall consider distance between *vertices in a graph*.

As we said in Chapter 2, the length of a path in a graph is the number of edges in it, and the *distance* between two connected vertices is the minimum length of paths joining them (a path attaining this length is often called a "shortest path" joining the vertices). In a network, with weights on the edges, one sometimes defines the length as the sum of the weights on the edges of a path; to distinguish between the two, we will refer to this as *weighted length*, and similarly define *weighted distance*.

We shall write $D(x, y)$ for the distance between vertices x and y. By definition, $D(x, x) = 0$.

9.1.2 Longest Distances

There are several definitions of the *diameter* of a graph. The key property used in any definition of graph diameter is *eccentricity*.

Definition 9.1. Given a graph $G = (V, E)$ the *eccentricity* of a vertex $v \in V$, denoted by $\varepsilon(v)$, is the maximum distance from v to any other vertex in the graph. That is, $\varepsilon(v) = \max_{u \in V} D(v, u)$.

Definition 9.2. Given a graph $G = (V, E)$, the *diameter* of G is

$$D(G) = \frac{\sum_{v \in V} \varepsilon(v)}{|V|}.$$

Graph diameter can be calculated using algorithms for all pairs of shortest paths. From the algorithms, averaging over all of the vertex eccentricities is straightforward.

Section 9.3 presents an example network for calculating vertex eccentricities and graph diameter.

Of course, the distance function in the definition of eccentricity could be replaced by the weighted distance. In that case we would refer to the *weighted eccentricity* and graph *weighted diameter*.

9.1.3 Average Distances

It is of interest to know the typical distance between two vertices in a graph. One obvious parameter is the *mean path length*, the mean value of $D(x, y)$, where x and y vary through the vertices. While this is a useful parameter when known, its calculation is impractical for large graphs such as those that arise in discussing intelligent networks. It is necessary to estimate the average. Such estimation must be carried out by sampling. Since calculating a median is much faster than calculating a mean, there are better sampling techniques available for accurately estimating medians than for means, so Watts and Strogatz considered the median path length as a measure.

The sampling technique for estimating medians was studied, and its accuracy estimated, by Huber [93]. One might think that the median would be a significantly less reliable measure of average distance than the mean. However, Huber [93] studied the sampling technique for estimating medians, and estimated its accuracy. The results are summarized in [184, pp. 29–30] and show that the median is quite a good estimator.

Another theoretical advantage is that medians can be calculated even if the graph is disconnected, provided the number of disconnected pairs of vertices is not too large. (A disconnected pair corresponds to an "infinite" path, which is longer than any finite path. Provided fewer than half the paths are "infinite," the median can be calculated.) Newman [135] suggests ignoring disconnected pairs, but we believe that the "infinite" distance technique is more appropriate.

To approximate a median of a set is straightforward, because the sample median is an unbiased estimator of the median (and of the mean, in normal or near-normal data). One simply takes a number of readings and finds their median.

If S is a set with n elements, its median is defined as that value $M(S)$ such that $n/2$ members of S have value $\leq M(S)$ and $n/2$ members have value $\geq M(S)$. Let us say that an estimate M of $M(S)$ is "of accuracy" δ if at least $\delta n/2$ members have value $\leq M$ and $\delta n/2$ members have value $\geq M$. Then Huber proved the following theorem:

Theorem 9.3. *[93] Suppose δ (usually near 1) and ϵ (usually small) are positive constants. Approximation of the median $M(S)$ of a set S by sampling s readings yields a value of accuracy at least δ with probability $1 - \epsilon$, where*

$$s = 8 \ln \frac{2}{\epsilon} \left(\frac{\delta}{1 - \delta} \right)^2 .$$

9.1.4 Characteristic Path Length

The median of a finite set of readings is a discrete-valued function: it takes either one of the reading values or the average of two of them. This leads to some minor inaccuracies.

Consider, for example, two sets of readings: A contains 49 readings 0 and 50 readings 1, while B contains 50 readings 0 and 49 readings 1. These sets are very similar, but the medians are 0 and 1 respectively. To avoid such chaotic leaps in value, the following definition is used.

The *characteristic path length* $L(G)$ of a graph G is calculated as follows. First, for each vertex x, the median $\overline{D}_x$ of all the values $D(x, y)$, y a vertex, is calculated. Then $L(G)$ is the mean of the values $\overline{D}_x$, for varying x. So, in the sampling procedure suggested by Huber, one calculates a number of values of $\overline{D}_x$ and takes their median.

9.1.5 Clustering Coefficient

The *neighborhood* of a set of vertices consists of all the vertices adjacent to at least one member of the set, excluding members of the set itself. If the set consists of a single vertex x, we denote the neighborhood of $\{x\}$ by $N(x)$. The graph generated by $N(x)$, denoted by $\langle N(x)\rangle$, has vertex set $N(x)$, and its edges are all edges of the graph with both endpoints in $N(x)$. We write $k(x)$ (the *degree* of x) and $e(x)$ for the numbers of vertices and edges, respectively, in $\langle N(x)\rangle$. Then the *clustering coefficient* γ_x of x is

$$\gamma_x = \frac{e(x)}{\binom{k(x)}{2}} = \frac{2e(x)}{k(x)(k(x)-1)}.$$

In other words, it equals the number of connections between the neighbors of x divided by the maximum possible number of connections.

The *clustering coefficient* of a *graph G* equals the mean of the clustering coefficients of all vertices of G, and is denoted $\gamma(G)$ or simply γ.

The extreme maximum value $\gamma(G) = 1$ occurs if and only if G consists of a number of disjoint complete graphs of the same order (each has $k + 1$ vertices, where every vertex has the same degree k). The extreme minimum value $\gamma(G) = 0$ is attained if and only if the graph G contains no triangles.

9.1.6 Directed Graphs

The theory of these measures is much the same if directed graphs are considered. However, in calculating the clustering coefficient in a directed graph, the formula

$$\gamma_x = \frac{e(x)}{2\binom{k(x)}{2}} = \frac{e(x)}{k(x)(k(x)-1)}$$

should be used. In other words, the result should be halved, corresponding to the fact that there are $k(k - 1)$ possible directed edges between k vertices.

9.2 Diameters

Graph diameter can be used as as a measure of difference between two graphs. Given two graphs G and H, simply calculate the difference between their respective diameters. The difference $f(G, H)$ is given by

$$f(G, H) = |D(G) - D(H)|.$$

We shall now discuss some theoretical properties of this distance measure and some sensitivities that this measure has for detecting change in networks.

9.2.1 A Pseudometric

It is a desirable property that a measure of distance between networks (like the ones defined in Chapter 4) should be a *metric*. Recall that a metric on set A is a function $d : A \times A \to \mathbb{R}$ that maps a product set $A \times A$ to the real numbers such that four axioms hold: nonnegativity, separation, symmetry, and the triangle inequality.

The axiom of separation states that for a given distance measure, $d(a, b) = 0$ if and only if $a = b$. In terms of graphs, two graphs are equivalent if they are *isomorphic*, that is, the two graphs have a one-to-one mapping of the vertices that preserves the edge connections [188]. It is certainly possible that two graphs G and H are not isomorphic but have the same graph diameter and therefore $f(G, H) = 0$. A simple example of this is the complete graph with n vertices, K_n. The complete graph K_n has diameter $D(K_n) = 1, \forall\, n \geq 2$, and it follows that $f(K_i, K_j) = 0, \forall\, i, j \geq 2$; however, K_i and K_j are not isomorphic if $i \neq j$.

Although the function $f(G, H) = |D(G) - D(H)|$ is not a metric, it is still a useful measure of distance between two graphs. To discuss this, we present the notion of a *pseudometric*.

Definition 9.4. A pseudometric on set A is a function $d : A \times A \to \mathbb{R}$ for which the following axioms hold for all $a, b, c \in A$:

(i) $d(a, b) \geq 0$,
(ii) $d(a, b) = 0$, if $a = b$,
(iii) $d(a, b) = d(b, a)$,
(iv) $d(a, b) \leq d(a, c) + d(c, b)$.

For the graph diameter distance measure, axiom (ii) of Definition 9.4 implies that if two graphs are isomorphic, denoted by $G \cong H$, then they have the same graph diameter, but having the same graph diameter does not imply that the two graphs are isomorphic. This constitutes a relaxation of the separation axiom in the definition of a metric.

Theorem 9.5. *The function $f(G, H) = |D(G) - D(H)|$ is a pseudometric.*

Proof. It must be shown that all axioms in Definition 9.4 hold for $f(G, H) = |D(G) - D(H)|$. For (i), by definition of absolute value, $|D(G) - D(H)| \geq 0, \forall\, G, H$, so $f(G, H) \geq 0$.

Next, to show that (ii) holds, the relation of graph isomorphism is used. If G and H are isomorphic, then there exists a bijection $g : V(G) \to V(H)$ such that $\{u, v\} \in E(G)$ implies that $\{g(u), g(v)\} \in E(H)$. Due to this bijection, $|V(G)| = |V(H)|$ and $\varepsilon(v) = \varepsilon(g(v))$ and therefore $f(G, H) = 0$ if G is isomorphic to H.

Again, by definition of absolute value, $f(G, H) = |D(G) - D(H)| = |D(H) - D(G)| = f(H, G)$, so (iii) holds.

Finally, it must be shown that the triangle inquality $f(G, H) \leq f(G, K)+f(K, H)$ holds for all graphs G, H, K. Without loss of generality it is assumed that $D(G) \geq D(H)$. Three cases are considered. In the first case, let $D(G) \geq D(H) \geq D(K)$. Then

$$|D(G) - D(K)| + |D(K) - D(H)| = D(G) - D(K) + D(H) - D(K),$$

but $D(H) - D(K) \geq 0$ and $-D(K) \geq -D(H)$, so

$$\begin{aligned} D(G) - D(K) + D(H) - D(K) &\geq D(G) - D(K) + 0 \\ &\geq D(G) - D(H) \\ &= |D(G) - D(H)|. \end{aligned}$$

For the second case, let $D(G) \geq D(K) \geq D(H)$. Then

$$\begin{aligned} |D(G) - D(K)| + |D(K) - D(H)| &= D(G) - D(K) + D(K) - D(H) \\ &= D(G) - D(H) \\ &= |D(G) - D(H)|. \end{aligned}$$

In the last case, let $D(K) \geq D(G) \geq D(H)$. Then

$$|D(G) - D(K)| + |D(K) - D(H)| = D(K) - D(G) + D(K) - D(H),$$

but $D(K) - D(G) \geq 0$ and $D(K) \geq D(G)$, so

$$\begin{aligned} D(K) - D(G) + D(K) - D(H) &\geq 0 + D(K) - D(H) \\ &\geq D(G) - D(H) \\ &= |D(G) - D(H)|. \end{aligned}$$

Axioms (i)–(iv) hold, and therefore $f(G, H) = |D(G) - D(H)|$ is a pseudometric.

It is important to note that it is possible to define an equivalence class for which the function f is a metric. If we define $\mathfrak{G}$ to be the set of all networks with diameter equal to $D(G)$ and write $D(\mathfrak{G})$ for the (common) diameter of members of $\mathfrak{G}$, then $f(\mathfrak{G}, \mathfrak{H}) = |D(\mathfrak{G}) - D(\mathfrak{H})|$ is a metric on the (quotient set of) equivalence classes $\{\mathfrak{G}\}$.

9.2.2 Sensitivity Analysis

In addition to being a pseudometric, if difference in graph diameter is to be a useful indicator of changes in network topology, it is important that small changes in a relatively dense unweighted network should not result in large changes in the diameter. To investigate the sensitivity of our distance measure, we look at the family of complete graphs and simple cycles.

The eccentricity of any vertex of the complete graph K_n is 1, so the diameter is $D(K_n) = 1$. If one edge is deleted, two vertices of the resulting graph G will have eccentricity 2, so $D(G) = \frac{n+2}{n}$. If k edges are deleted, where $k \leq \frac{n}{2}$, at most $2k$ vertices will have eccentricity 2, and the others will have eccentricity 1. So $D(G) \leq 2$.

In general, if a graph G has any vertex x of degree $n-1$, then $\varepsilon(x) = 1$. Any two other vertices y, z are joined by a path of length 2, namely yxz, so $\varepsilon(y) \leq 2$ and $\varepsilon(z) \leq 2$. So $D(G) < 2$. This means that even when $\frac{n}{2}$ edges are deleted from K_n, the resulting graph has diameter less than 2 unless the $\frac{n}{2}$ edges form a one-factor.

At the other extreme, consider the cycle C_n. Every vertex has eccentricity $\lfloor \frac{n}{2} \rfloor$, so this is the graph diameter. If C_n has vertices $x_1, x_2, \ldots, x_n$, let C_n^i denote the result of adding edge $x_n x_i$ (a *chord* of length i) to C_n. The addition of an edge reduces the graph diameter. As an example, we have

$$D(C_{10}) = 5,\ D(C_{10}^2) = 4.2,\ D(C_{10}^3) = 4.4,\ D(C_{10}^4) = 3.8,\ D(C_{10}^5) = 4.0,$$

while

$$D(C_9) = D(C_9^2) = 4,\ D(C_9^3) = D(C_9^4) = 3.56.$$

Theorem 9.6. *If* $n \equiv 0 \pmod 4$, *then*

$$D(C_n^i) \geq D\left(C_n^{\frac{n}{2}}\right) = \frac{3n}{8}.$$

If $n \equiv 2 \pmod 4$, *then*

$$D(C_n^i) \geq D\left(C_n^{\frac{n}{2}-1}\right) = \frac{3n}{8} + \frac{4}{8n}.$$

If $n \equiv 1 \pmod 4$, *then*

$$D(C_n^i) \geq D\left(C_n^{\lfloor \frac{n-1}{2} \rfloor}\right) = \frac{3n+1}{8}.$$

If $n \equiv 3 \pmod 4$, *then*

$$D(C_n^i) \geq D\left(C_n^{\lfloor \frac{n-3}{2} \rfloor}\right) = \frac{3n}{8} - \frac{14n-15}{8n}.$$

Proof. In the first case, it is easy to see that the graph diameter is smallest when the chord is of length $n/2$: in those cases the eccentricities of the vertices are $n/2-1$ (this is the eccentricity of two vertices, the vertices furthest from the endpoints of the chord), $n/2-2$ (four vertices adjacent to the two just mentioned), $n/2-3$ (four times vertices), $\ldots, n/4+1$ (four vertices), $n/4$ (two vertices, the endpoints of the chord). The average is as shown.

The other cases are handled similarly. It is interesting to note that when $n \equiv 2 \pmod 4$, $D\left(C_n^{\frac{n}{2}}\right) = D\left(C_n^{\frac{n}{2}-1}\right) + \frac{n-2}{2n}$.

Theorem 9.6 shows that the addition of an edge can reduce the diameter of C_n by approximately 25%, from $\lfloor \frac{n}{2} \rfloor$ to approximately $\frac{3n}{8}$. This discussion shows that the distance measure based on graph diameter is sensitive to small changes in sparse graphs and less sensitive for small changes in nearly complete graphs.

This analysis is pointless in the case of weighted networks, because the difference in diameter due to inclusion or deletion of one edge can be made as large as we please by increasing the weight of the edge.

9.3 An Example Network

Here an example network (graph) is used to demonstrate the calculation of vertex eccentricities and difference in graph diameter as presented in Section 9.2. First, the graph without an edge-weight function will be used, then an edge-weight ω will be introduced.

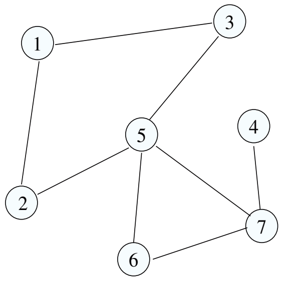


Fig. 9.1. Example graph.

Let G be the graph in Figure 9.1 with $V = \{1, 2, \ldots, 7\}$ and $E = \{\{1, 2\}, \{1, 3\}, \{2, 5\}, \{3, 5\}, \{4, 7\}, \{5, 6\}, \{5, 7\}, \{6, 7\}\}$.[1] The eccentricities for the example graph are $\varepsilon(1) = 4$, $\varepsilon(2) = 3$, $\varepsilon(3) = 3$, $\varepsilon(4) = 4$, $\varepsilon(5) = 2$, $\varepsilon(6) = 3$, $\varepsilon(7) = 3$. Taking an average of all the eccentricities, $D(G) \approx 3.143$.

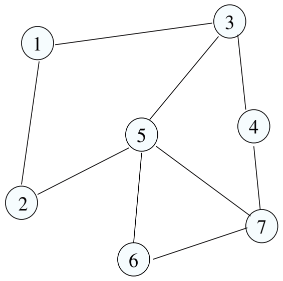


Fig. 9.2. Example graph with edge {3,4} added.

To demonstrate that graph diameter can detect small changes in graph topology, an edge is added to the graph in Figure 9.1. Figure 9.2 shows the graph with the edge $\{3, 4\}$ added. Let this graph be $H = (E \cup \{3, 4\}, V)$. It is left to the reader to verify that the graph diameter $D(H) \approx 2.714$ and the difference $f(G, H) \approx 0.429$.

[1]The example graph is undirected; therefore, the edges are just unordered pairs of adjacent vertices. It is trivial to convert the graph to directed as described in Section 9.1.

Now an edge-weight ω_1 is introduced on the original graph G to give $G_1 = (V, E, \omega_1)$. For simplicity, let $\omega_1(e) = 3, \forall e \in E$. This yields a graph diameter $D(G_1) \approx 9.429$. To demonstrate that graph diameter can detect a change in edge-weight, set $\omega_2 = \omega_1, \forall e \neq \{3, 5\} \in E$. Let $\omega_2(\{3, 5\}) = 1$ and $G_2 = (E, V, \omega_2)$. With this minimal change, $D(G_2) = 8.0$, and $f(G_1, G_2) \approx 1.429$.

9.3.1 Time Series Using f

The examples in Section 9.3 show that graph diameter can be used to detect small changes in both network topology (the underlying graph) and network traffic (the weighted case). In this section we use graph diameter to measure change in a time series of an actual communication network described in detail in Sections 3.4.2 and 4.2. Both unweighted and weighted time series are presented where the corresponding edge-weight is traffic. The time series are based on daily traffic activity.

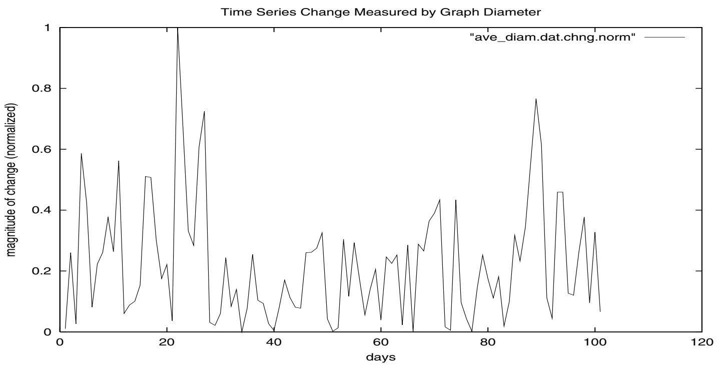


Fig. 9.3. Time series using comparison.

Figure 9.3 shows the times series change based on the difference in graph diameter normalized by the maximum change over the interval. Peaks in the time series represent potential abnormal change. Abnormal change would be based on deviations from some normal state of the network.

Figure 9.4 shows the time series change based on the difference in weighted graph diameter, normalized by the maximum change over the interval. As in the plot in Figure 9.3, peaks represent potential abnormal change. The peaks and valleys differ between the two plots, since different factors are compared for weighted and nonweighted graphs. The nonweighted plot would show topological changes where edges are either added or deleted. The weighted plot, based on edge traffic, can show both topological changes as well as abnormal changes in edge traffic. As demonstrated in Section 9.2.1 and Section 9.3, both are sensitive to small changes in the network.

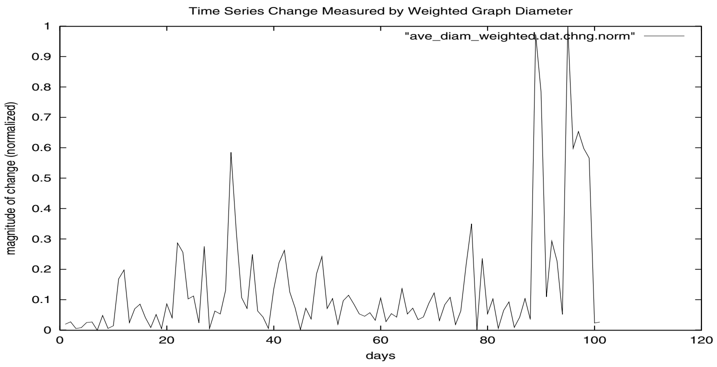


Fig. 9.4. Time series using weighted comparison.

9.4 Time Series Using *D*

Instead of considering time series of f (difference in graph diameters), one can use the diameter D directly and obtain an interesting time series of graphs. With the measure D, we simply use the characterization of a graph independent of any other graphs. This is the first time that we have addressed the problem of abnormal change detection from a characterization of an individual graph. Note that f (as a function of time) is effectively the rate of change of D in time.

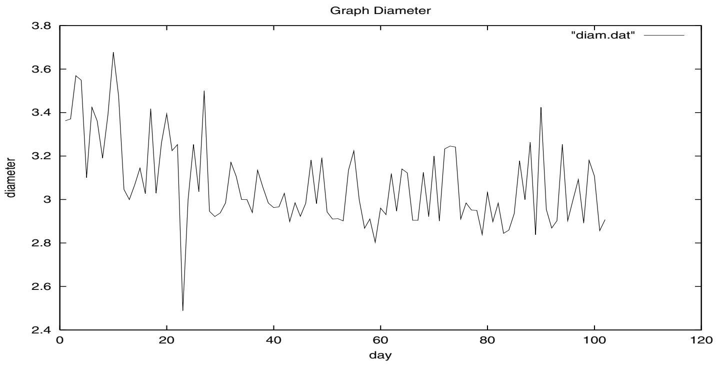


Fig. 9.5. Time series using only characterization.

As can be seen in Figure 9.5, there are still peaks and valleys in the time series plot of graph diameter, but these peaks and valleys have a different connotation than in the time series. This connotation is based on the difference in two graphs. From a communications network perspective, large graph diameter implies a longer average communication path between any two vertices. From the aspects of performability and redundancy, small graph diameter is desirable so as to minimize the longest average communication path link. So from this standpoint, an abnormal event could be defined as an event in which the graph diameter moves above some established threshold. We also notice the obvious correlation of significant events with the time series using the consecutive f-distances (see Figure 9.3).

9.5 Characteristic Path Lengths, Clustering Coefficients, and Small Worlds

9.5.1 Two Classes of Graphs

Two classes of graphs have been used as standards in studies relating to characteristic path lengths and clustering coefficients. They are lattice graphs and random graphs.

Lattice Graphs

Lattice graphs were defined by Watts and Strogatz [185] to provide a family of highly structured graphs. A *d-lattice* is analogous to a Euclidean lattice of dimension d. The n vertices are labeled with the integers modulo n, and where x is joined to all the vertices $(v - i^{d'})$ and $(v + i^{d'})$ for $1 \leq i \leq \frac{1}{2}k$, $1 \leq d' \leq d$, where k is some integer (it is usually assumed that $k \geq 2d$). Examples of a 1-lattice and a 2-lattice with $k = 4$ are shown in Figures 9.6 and 9.7.

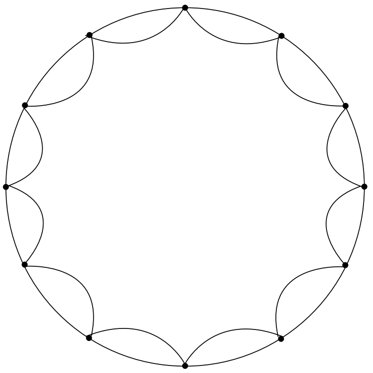


Fig. 9.6. A 1-lattice with $k = 4$.

Suppose G is a 1-lattice with n vertices, for which k is even and at least 2. Then

$$\gamma(G) = \frac{3(k-2)}{4(k-1)}.$$

The exact value of $L(G)$ depends on the remainder when n is divided by k, but approximately,

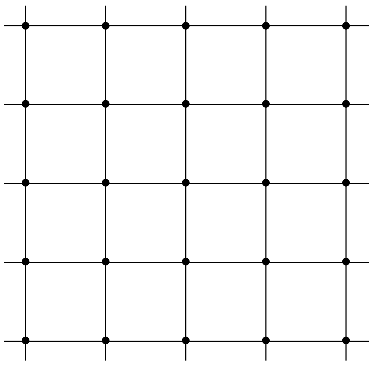


Fig. 9.7. A 2-lattice with $k = 4$ (joined as on a torus).

$$L(G) = \frac{n(n + k - 2)}{2k(n - 1)}$$

and

$$\gamma(G) = \frac{3(k - 2)}{4(k - 1)}.$$

If we wish to discuss directed graphs, we define the directed version of a d-lattice by replacing every edge xy in the undirected case by the pair of edges $\{xy, yx\}$. The above formulas remain true in the directed case.

Random Graphs

There are two standard models of random graphs in the mathematical literature (see, for example, [16]).

A *graph of type* $G(n, M)$ has n vertices and M edges. The M edges are chosen from the $\binom{n}{2}$ possibilities in such a way that any of the possible M-sets is equally likely to be the one chosen. (In statistical terminology, the edges are a simple random sample of size M from the possibilities.)

A *graph of type* $G(n, p)$ has n vertices. The probability that the vertex-pair xy is an edge is p, and the $\binom{n}{2}$ events "xy is an edge" are independent.

In studying most properties—in particular, the properties that will interest us—these two models of random graphs are interchangeable, provided M is approximately np. When this is true, it is more usual to study $G(n, p)$, since the independence makes mathematical discussions simpler.

The average degree of vertices in a $G(n, p)$ is clearly $p(n - 1)$. The clustering coefficient is approximately p; in other words, for a reasonably large random graph of average degree k, the clustering coefficient is approximately k/n.

9.5.2 Small-World Graphs

From the above results on clustering coefficients in completely random graphs and lattice graphs, it would seem that tight local clusters entail long average distances, and that reducing average distance requires abandoning local clustering.

However, Watts and Strogatz [185] discovered a class of graphs that combine strong local clustering with relatively short average distance. They started from a highly regular graph (a 1-lattice), and randomly changed a small fraction of the connections (they used a random process in which certain edges were deleted and replaced by edges with one endpoint the same as in the original and the other chosen randomly). In these graphs the clustering coefficient stays near $3/4$, while the average distance between vertices is is approximately $\ln(n)/\ln(k)$, so it is proportional to $\log(n)$ rather than to n.

Small-world networks evolve when links are made randomly in a sparse structured network. The underlying graph of such a network is an amalgam of local clusters and random long-range links. For example, in a network representing acquaintanceship, the clusters might be cliques of friends or coworkers, and long-range links might occur through a vacation or when close friends work for different companies. Watts and Strogatz conjectured that the underlying graphs of small-world networks would behave like their artificially constructed graphs, which they accordingly called *small-world graphs*.

The name “small-world network” comes from the buzz phrase “It’s a small world,” sometimes heard when strangers discover a common link between them. For example, when you board an airplane, you might find that your uncle works in the same office as the wife of the person sitting next to you. Among other applications, the theory has been used in social network theory to track how information and ideas spread through a community. It serves to model the spread of infectious diseases. Moreover, it appears that the nervous system may be wired along the same principle.

The small-world effect was first studied by Milgram [132] in 1967. He described an experiment in which people in Omaha, Nebraska, were given a letter intended for delivery to a stranger in Massachusetts and asked to pass it along to someone they thought might know the intended recipient, with instructions for that person to do the same. Only about 20% of the letters ever reached the “target.” For those that did, though, the surprise was how quickly they arrived: It rarely took more than six intermediate steps.

Recent mathematical discussion of the small-world phenomenon has focused on the underlying (undirected) graphs of small-world networks. The characteristic path lengths and clustering coefficients of these graphs have been studied. In particular, Watts and Strogatz applied their models to three real-world networks.

One example was the Internet Movie Database (IMDb), in which the vertices are movie actors, with a connection between any two who have appeared in the same movie. Analyzing the IMDb network, which then had an average of $k = 61$ connections per actor for $n = 225{,}226$ actors (the IMDb has increased considerably in size since their study), Watts and Strogatz found a clustering coefficient $C = 0.79$—surprisingly close to the theoretical value for a highly regular network—and an average distance between actors of 3.65, which is much closer to $\ln(n)/\ln(k) = 2.998$ than it is to $n/2k = 1846$.

The other two networks analyzed were electrical in nature: the power grid for the western United States and the neural system of the much-studied nematode *C. elegans*. (Nematodes are a class of worms comprising 10,000 known species. The parasitic kinds, such as hookworm, infect nearly half the world's human population.) The power grid has $n = 4941$ vertices, consisting of generators, transformers, and substations, with an average of $k = 2.67$ high-voltage transmission lines per vertex. *C. elegans* has a mere $n = 282$ neurons (not counting 20 neurons in the worm's throat, which biologists have not yet completely mapped), with an average of $k = 14$ connections—synapses and gap junctions—for each neuron.

The power grid has a rather low clustering coefficient $\gamma = 0.08$. But this is 160 times what would be expected for a random network of the same size. The average distance between vertices is 18.7, far below the 925 predicted for a perfectly regular network. (Actually, this is an unfair comparison, since Watts and Strogatz's regular network is essentially 1-dimensional, whereas the power grid is inherently 2-dimensional. It would be fairer to compare the power grid to a hexagonal honeycomb, for which most vertices have three links. The average distance between two vertices in an $m \times m$ honeycomb, which has $n = 2m(m + 2)$ vertices altogether, is approximately $2m$. For $n = 4941$, that is approximately 98.) For *C. elegans*, the clustering coefficient is 0.28 (as against 0.05 for the random model), and the average distance is 2.65 (with 2.25 calculated for the random model and 10 for the corresponding lattice graph with $k = 14$).

Extensive surveys of small-world phenomena appear in the literature [6, 184].

9.6 Enterprise Graphs and Small Worlds

Throughout this book we are studying enterprise networks, such as communications networks and other intelligent networks. These networks often have a large amount of initial structure imposed, but with use they evolve in a similar way to small-world networks. Enterprise networks often have a large amount of initial structure imposed, but with use they evolve in a similar way to small-world networks. We refer to the underlying graph of such a network as an *enterprise graph*. We suspected that the small-world model would apply to these graphs, and shall present evidence for this hypothesis.

9.6.1 Sampling Traffic

In our previous investigations (see, for example, Sections 3.4.2 and 4.2) we have used sample network traffic collected from the network management system of a large-enterprise data network. A traffic probe was installed on a single physical link in the network and traffic parameters were logged over 24-hour periods in daily log files. A traffic log file contains information on the logical originators (O) and destinations (D) of traffic (derived from network address information) and the volume of traffic transmitted between OD pairs. To reduce the overall number of OD pairs in the data set, the 45,000 individual users (network addresses) were clustered by the business domain they belonged to on the data network. The aggregated logical flows of traffic between the

325 business domains observed over this physical link in a day were then represented as a directed and labeled graph. Vertex-weight identified the business domains of logical nodes communicating over the physical link with edge-weight denoting the total traffic transmitted between corresponding OD pairs over a 24-hour period.

Successive log files collected over subsequent days produced a time series of corresponding directed and labeled graphs representing traffic flows between business domains communicating over the physical link in the network. Log files were collected continuously over a period of several months, from July 19 to October 25, 1999, and weekends and public holidays were removed to produce a final set of 90 log files representing the successive business days' traffic.

9.6.2 Results on Enterprise Graphs

To test our hypothesis, we used our data to produce two samples of enterprise graphs by removing weights and directions (see Sections 3.4.2 and 4.2). The resulting graph represented connections in the network. Time series of both the original graphs (with 45,000 vertices corresponding to individual users) and the domain graphs (with 325 vertices corresponding to the business domains) were studied.

Both characteristic path length and clustering coefficient can be discussed in the case of directed graphs. The theory of both these measures is much the same in the directed case. However, in calculating the clustering coefficient in a directed graph, the formula

$$\gamma_x = \frac{e(x)}{2\binom{k(x)}{2}} = \frac{e(x)}{k(x)(k(x)-1)}$$

should be used. In other words, the result should be halved, corresponding to the fact that there are $k(k-1)$ possible directed edges between k vertices.

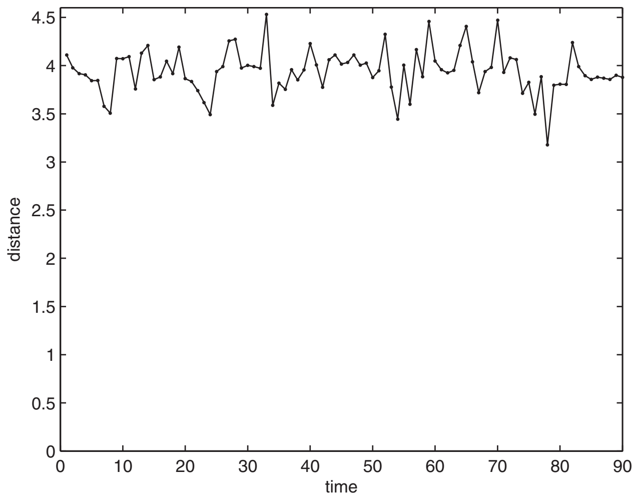


Fig. 9.8. Characteristic path lengths for original 45,000 users.

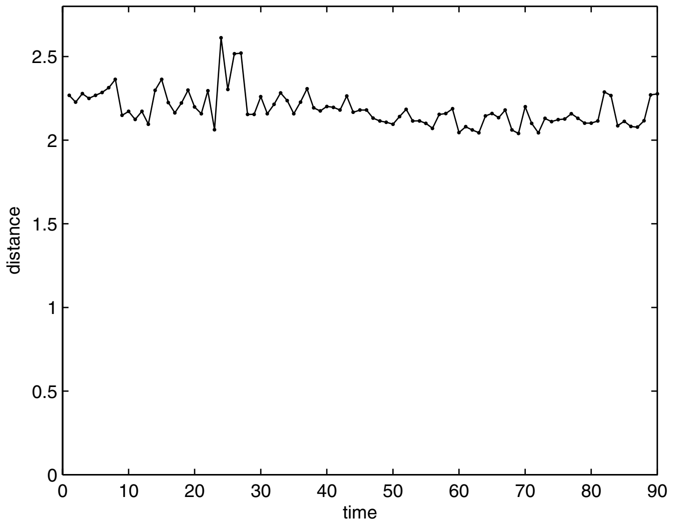


Fig. 9.9. Characteristic path lengths for 325 domains.

Much work has been done on random directed graphs. To have a directed benchmark for highly structured graphs, we define the directed version of a d-lattice by replacing every edge xy in the undirected case by the pair of edges $\{xy, yx\}$. The formulas for characteristic path length and clustering coefficient apply to the directed lattices.

We measured the characteristic path lengths for the directed versions of our two series of enterprise graphs. In the original graphs, the characteristic path length ranges approximately from 3 to 4.5, averaging between 3.8 and 3.9. In the domain graph series the average is about 2.2; the length is never less than 2, and there were only three days when it was greater than 2.35. This is consistent with the hypothesis that enterprise graphs behave like small-world graphs in the directed case also. The time series of characteristic path lengths are shown in Figures 9.8 and 9.9.

In both series of graphs, the clustering coefficient ranges approximately from 0.58 to 0.75, averaging about 0.67. In the domain graph series there is very little variation; the original graph series is a little less tight. These figures are very close to the lattice graph figure, 0.749. This is also consistent with the small-world hypothesis. The time series of clustering coefficients are shown in Figures 9.10 and 9.11.

We produced a histogram of the values γ_x for all vertices x on the first day (day 0) of the series. These histograms are shown in Figures 9.12 and 9.13.

In our examples of enterprise graphs, there were very few situations in which communication was not symmetric, that is, if there was a directed edge from x to y, there was usually a directed edge from y to x. This would not necessarily be true in some enterprise graphs, for example in a highly structured social network where instructions or data are broadcast from a source that does not accept answering messages. While the characteristic path length would be higher in such networks (due to the existence

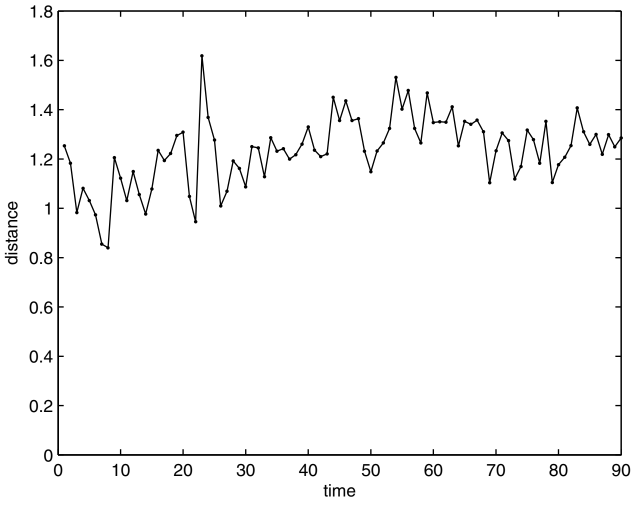


Fig. 9.10. Clustering coefficients for original 45,000 users.

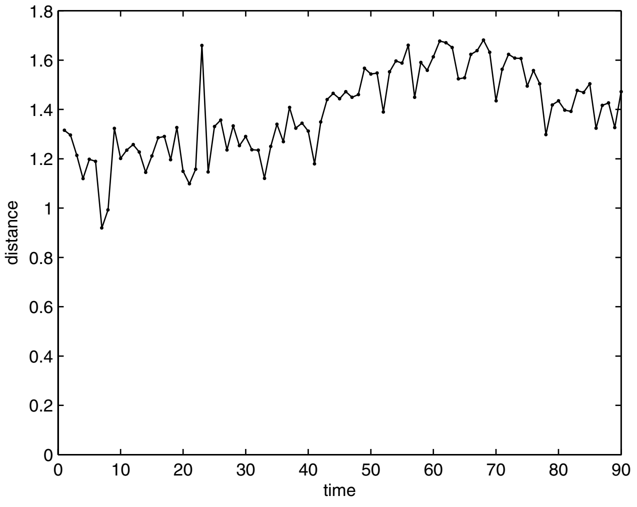


Fig. 9.11. Clustering coefficients for 325 domains.

of a larger number of "infinite" paths), and the clustering coefficient would be lower, we predict that the underlying graph will still fit the "small-world" pattern.

9.6.3 Discovering Anomalous Behavior

We do not derive a measure of change between snapshots of the network from the characteristic path length or clustering coefficient. However, the relative consistency

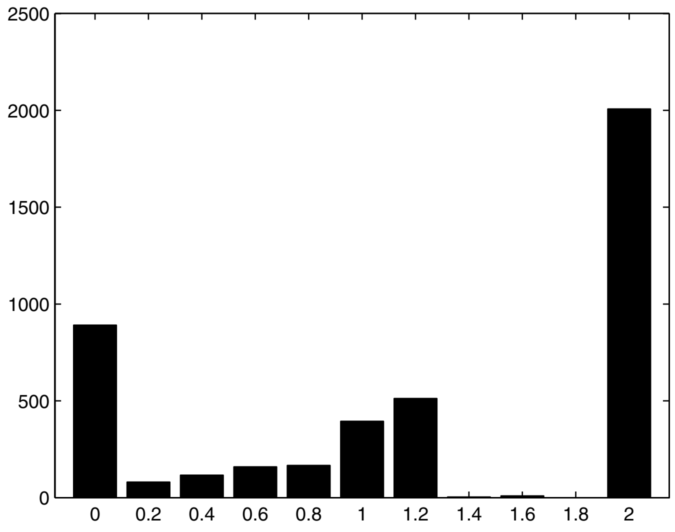


Fig. 9.12. Clustering coefficients of vertices, day 0 (original 45,000 users).

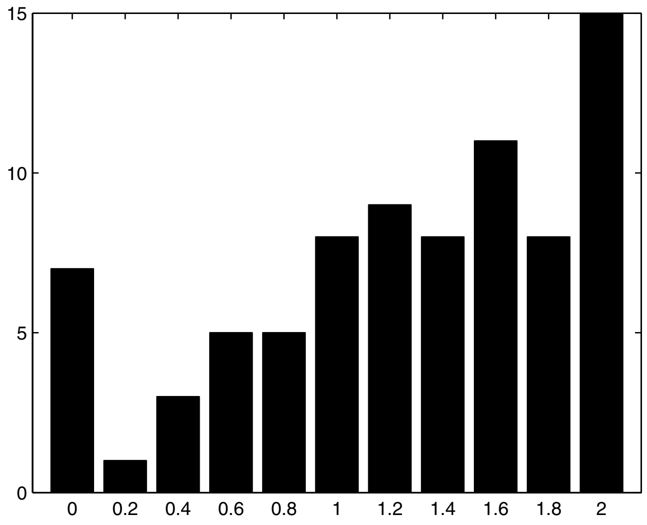


Fig. 9.13. Clustering coefficients of vertices, day 0 (325 domains).

of the characteristic path length, particularly for the domain graphs, suggests that one should study those days with a markedly different characteristic length, to see whether they correspond to other marked differences in the network. Therefore the "small-world" measures provide a significant tool for network management.

We would also suggest that the upward trend in the clustering coefficient be investigated. If it in fact reflects a seasonal or other consistent change in the network, this would be important, since other measures that have been tested on these data do not reflect such a consistent change.

10

Tournament Scoring

10.1 Introduction

Various methods have been used to assess the relative strengths of players in round-robin tournaments. These methods involve the digraph representing the results of the tournament. (Somewhat confusingly, this digraph is also called a "tournament.") In this chapter we shall generalize these techniques to provide a graph measure that we call *modality*. We propose techniques to measure rapid change in the structure of the network by looking at changes in modality.

The ideas of matrix theory will be used in discussing tournaments. In addition to standard references such as [92] and [114], another useful reference for nonnegative matrices is [145]. We shall use the notation $\|v\|$ to denote the (Euclidean) *norm* of the vector v. Thus, if v has n elements,

$$\|v\| = \sqrt{\sum_{i=1}^{n} v_i^2}.$$

Let I_n, J_n, and e_n represent an identity matrix of order n, an $n \times n$ matrix with every entry 1, and a vector of length n with every entry 1, respectively. If the order n is clear from the context, it will be omitted. A zero matrix or vector will be denoted by O or 0 respectively.

10.2 Tournaments

10.2.1 Definitions

In general, a *tournament* is an oriented complete graph. In other words, it is a digraph in which every pair of vertices is joined by exactly one arc.

In sports scheduling, a tournament is an event in which a number of players compete two at a time. A *round-robin* tournament is one in which every pair of players competes exactly once, so a (graphical) tournament represents the results of a (scheduling) round-robin tournament in which ties are impossible: the players are the vertices and $x \rightarrow y$ whenever x beats y. We shall use the graphical and scheduling terminologies interchangeably.

10.2.2 Tournament Matrices

The win–loss outcomes of the matches in a tournament can conveniently be recorded in a *tournament matrix* $A = [a_{ij}]$, where $a_{ij} = 1$ if $i \rightarrow j$ and $a_{ij} = 0$ otherwise; in particular, A has zero diagonal. Such a matrix is called a *tournament matrix*. A square zero–one matrix A is a tournament matrix if and only if

$$A + A^T = J - I.$$

As an example, consider the 6-player tournament in which

- 1 beats 2, 3, 4, 5;
- 2 beats 3, 4, 5, 6;
- 3 beats 5, 6;
- 4 beats 3, 6;
- 5 beats 4, 6;
- 6 beats 1.

This tournament has matrix

$$M = \boxed{\begin{matrix} 0&1&1&1&1&0 \\ 0&0&1&1&1&1 \\ 0&0&0&0&1&1 \\ 0&0&1&0&0&1 \\ 0&0&0&1&0&1 \\ 1&0&0&0&0&0 \end{matrix}}\,.$$

A matrix is called *reducible* if it can be written in the form

$$M = \boxed{\begin{array}{c|c} X & Y \\ \hline O & Z \end{array}}\,,$$

where the submatrices X and Z are square, and *irreducible* otherwise. A tournament is called *reducible* if there is some ordering of the players for which the tournament matrix is reducible, and called *irreducible* otherwise.

10.3 Ranking Tournaments

10.3.1 The Ranking Problem

The classical way to rank participants in a tournament is to allocate points for victories. For example, in the example tournament, players 1 and 2 have score 4; players 3, 4, 5 have score 2; and player 6 has score 1. There is an immediate problem: how do we rank players 1 and 2 in this example? Our first intuition is that 1 should outrank 2, because 1 beat 2 when they met. Is this appropriate, or should they be considered equal? The second problem we notice is the anomalous behavior of player 6. This player won only one match. However, it was against one of the two strongest players. Should this performance receive extra credit? In general, should we give more credit to a player who beats "better" opponents?

10.3.2 Kendall–Wei Ranking

We outline a method due to Wei [187] and Kendall [102] that takes into account the quality of the opponents. We assume that the tournament is irreducible (if the tournament is reducible, all players in the upper part are ranked above all players in the lower part). We assume that there is a nonnegative quantity that we shall call the (relative) *strength* of a player, and that the players should be ranked according to strength. So the object of a scoring system is to estimate the strengths of players. We denote the strength of player i by w_i^*, and (assuming there are n players) define the *strength vector* of a tournament to be

$$w^* = (w_1^*, w_2^*, \dots, w_n^*) .$$

For convenience, we assume that the strength vector is normalized, so that the sum of strengths of the players adds to 1. Suppose we have an estimate of all the strengths of the players. One could define the strength of player i to depend on the sum of the strengths of the players that i beats. The Kendall–Wei method approximates w^* by finding a sequence of vectors $w^1, w^2, \dots, w^n$. In w^1, the strength of player i is proportional to the number of matches won in the tournament: in the example, we take

$$v^1 = (4, 4, 2, 2, 2, 1)$$

and define w^1 to be $v^1/\|v^1\|$. When v^{j-1} is determined, we define

$$v_i^j = \sum_{i \to k} v_k^{j-1} .$$

In other words, the strength of player i *in the* j*th iteration* is derived from the sum of the $(j-1)$*st iteration* strengths of the players beaten by i. This is a reasonable approximation to the suggested principle, and also is easy to compute: $v^1 = Ae$, and $v^j = Av^{j-1}$, so in general $v^n = A^n e$. Then $w^n = v^n/\|v^n\|$. If the sequence (w^n) approaches a limit, then that limit is a reasonable value for w^*. Proceeding with the example, we obtain

$$\begin{aligned}
v^2 &= (10, 7, 3, 3, 3, 4), \\
v^3 &= (16, 13, 7, 7, 7, 10), \\
v^4 &= (34, 31, 17, 17, 17, 16), \\
v^5 &= (82, 67, 33, 33, 33, 34), \\
v^6 &= (166, 133, 67, 67, 67, 82).
\end{aligned}$$

So w^6 is approximately (.65, .52, .26, .26, .26, .32). At this stage player 1 is significantly stronger than player 2 and player 6 has moved clearly into third place. It is not obvious that convergence occurs, but we would expect the order of strength of the players to stay unchanged in future iterations. An important question is, of course, whether convergence takes place.

10.3.3 The Perron–Frobenius Theorem

The *spectral radius* of a real square matrix A, denoted by $\rho(A)$ or simply ρ, is the largest of the absolute values of eigenvalues of A. The matrix A is called *primitive* if all its entries are nonnegative and if there is some positive integer power of A all of whose entries are positive. The following theorem guarantees convergence of the strength vector of a tournament matrix. It comes from a theorem of Perron [142], generalized by Frobenius [74] (see [114, p. 538]).

Theorem 10.1 (Perron–Frobenius). *Suppose A is an irreducible matrix with all entries nonnegative. Then $\rho(A)$ is an eigenvalue of multiplicity one for A, and there is a corresponding eigenvector v all of whose entries are positive. Moreover, any nonnegative eigenvector of A is a multiple of v. If, furthermore, A is primitive, then every other eigenvalue of A has absolute value less than $\rho(A)$.*

The multiple of v with norm 1, the eigenvector $v/\|v\|$, is called the *Perron vector* of A, and denoted by $v(A)$. If the eigenvalue λ of largest magnitude of a matrix A is unique, then it is known that $\lim_{k\to\infty} A^k e/\|A^k e\|$ exists, and in fact is an eigenvector corresponding to λ. In fact this is the *power method*, a standard technique for finding the eigenvalue of largest magnitude of A when that eigenvalue is unique (see, for example, [75, p. 209]). This means that the Kendall–Wei method always gives a result whenever there are four or more players, since all tournament matrices of size greater than three are primitive.

10.4 Application to Networks

10.4.1 Matrix of a Network

Consider a network in which the nodes communicate with each other, such as an internet or intranet. With such a network we associate a graph G, the *underlying graph* of the network, whose vertices correspond to users or sets of users. (The vertices might be individual nodes, or they might correspond to groups of users, or to servers.) Vertices are joined by an edge if information can travel between the vertices. With such a network we associate a weighted graph, called a *snapshot* of the network, in which edge $\{i, j\}$ is assigned a weight p_{ij} representing the number of packets of information that travel between vertices i and j in a given time period. (The vertices might be individual nodes, or they might correspond to groups of users, or to servers.) Such a snapshot might represent all transactions in one day, or in one week, or any other appropriate period. Thus the snapshots form a time series of graphs. We have discussed these series in [25].

Typically p_{ij} will be a count of the number of packets of information, but this is not necessary. We could use any function p_{ij} provided it is a monotone increasing function of the number of packets, and provided $p_{ij} = 0$ if and only if no information is transmitted. As defined, p_{ij} is symmetric, $p_{ij} = p_{ji}$, but one could equally well define p_{ij} to be the number of packets sent from i to j, and associate a digraph rather than a graph with the network. We shall discuss possible variations of p_{ij} in Section 10.6 below.

We now define the *snapshot matrix* corresponding to a snapshot to be the matrix $A = (p_{ij})$. A snapshot is called *reducible* if there is some ordering of the vertices for which the snapshot matrix is reducible, and called *irreducible* otherwise. Because of the symmetry of the matrix, a reducible snapshot is decomposable: the network decomposes into two or more components, with no communication between the components.

We define the underlying graph of a snapshot as the subgraph of G obtained by deleting edges (i, j) for which $p_{ij} = 0$. The underlying graph of a decomposable snapshot is disconnected, and the components that do not communicate correspond to the components of the underlying graph.

All the analysis in Section 10.3 can be carried out for matrices more general than tournament matrices. In fact, Kendall's original analysis put an entry $\frac{1}{2}$ in every diagonal position, and Thompson [170] showed that this $\frac{1}{2}$ may be replaced by any value r with $0 \le r < 1$. Moreover, one could multiply the matrix by a positive constant without affecting the Perron vector. In particular, consider a snapshot matrix of a communication network. The Perron vector of such a matrix will be defined and will measure relative "strength" of the nodes; the greater the entry in the ith position, the greater the "strength" of node i. What is this "strength"? From the discussion of the Kendall–Wei method, we see that the first approximation measures the amount of information that the node communicates with the rest of the network. Subsequent approximations weight this with the communicativeness of those that the node contacts, so the information will be disseminated more rapidly. If we were to treat the amount of information communicated to or originating from a node as a random variable, the first approximation to "strength" is the value of the variable at that node; the largest value corresponds to the mode of the distribution. Consequently, we refer to the measure as the *modality* of the node. The Perron vector will be called the *modality vector* of the network.

10.5 Modality Distance

10.5.1 Defining the Measure

We use the modality vector to define a measure of difference between snapshots. The *modality distance* between two snapshots with matrices A and B is

$$\|v(A) - v(B)\|.$$

Various problems exist where a measure of difference between two graphs is required; see, for example, the discussion in [157]. In order to make numerical comparisons between differences in different situations, it is desirable that measures of graph difference have metric properties (see [30]), and the modality distance is clearly a metric: if there are n nodes, then modality distance is derived directly from the Euclidean distance metric in n-dimensional space.

10.5.2 Applying the Distance Measure

The modality distance method was tested on two datasets comprising 102 and 284 days respectively, of TCP/IP traffic collected from a number of probes on an enterprise network excluding weekends and public holidays. The 284-days data were collected about a year after the 102-days data. In the 102-days data set, the network administrators identified three days (22, 64, and 89) on which they thought the network had changed or behaved anomalously. Only on day 64 was there a suggested reason: the introduction of a web-based personnel and financial management system. There may have been other network changes in this period but none were notable enough to be mentioned by network administrators. From Figure 10.1, one can see that modality distance obviously produces an evident change point on day 64. All tests were performed on a PC with Pentium R4 processor run at 3.00 GHz. Using Java-based REDBACK software, for 102-days data set 7 seconds of CPU time was used, while the 284-days data set required 102 seconds of processing time. The running times seem to be reasonable assuming that eigenvalues of large matrices were computed. Also, we chose the diagonal parameter r from [170] equal to 0.1, and we replaced every other zero entry of the adjacency matrices by 1.0×10^{-5}.

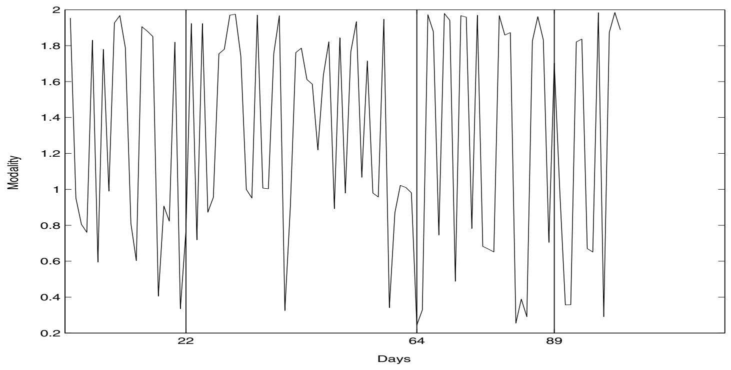


Fig. 10.1. Modality distance for 102 days data.

We also compared the modality distance results with a number of other time series graph distance measures. For example, the MCS distances [25, 30] using edges (Figure 10.2) and vertices (Figure 10.3) for the 102-days data set produced obvious large changes in network topology behavior on all three anomalous days. Naturally, since graphs in time series are labeled, this MCS-distance computations run in linear time and produced numerical distances in about 7 seconds on the same PC.

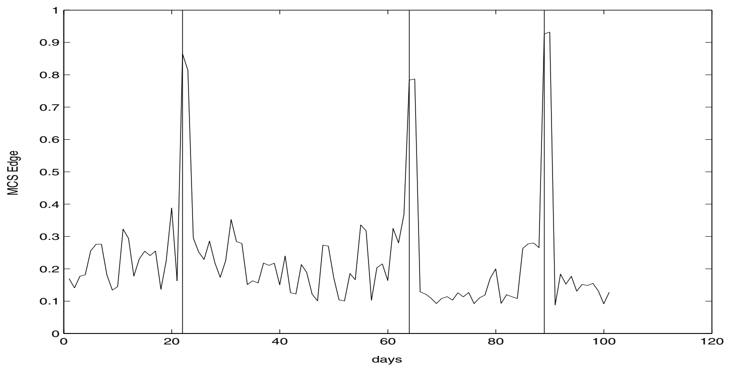


Fig. 10.2. MCS distance (edges) for 102-days data.

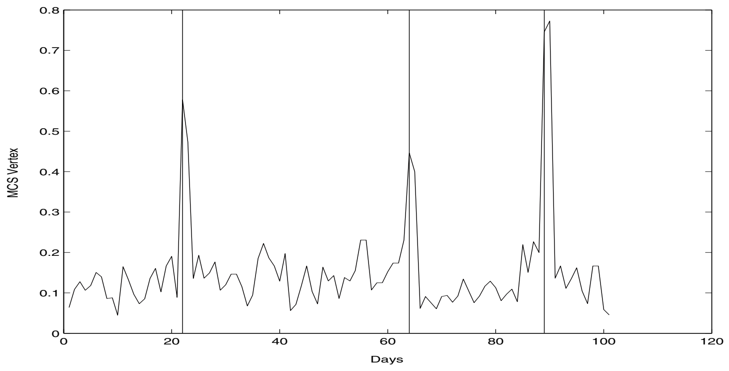


Fig. 10.3. MCS distance (vertices) for 102-days data.

Modality distance results for the 284-days data set are shown in Figure 10.4. We did not have any identified anomalous change points, but we could postprocess the obtained numerical time series of doubles with algorithms for change point detection and/or forecasting of network behavior.

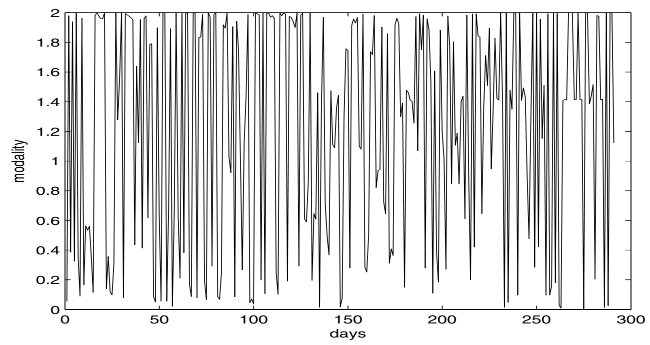


Fig. 10.4. Modality distance for 294-days data.

10.6 Variations in the Weight Function

The first possible variation is to consider the modality vector of the digraph corresponding to the network. This may give more information in cases in which the network corresponds to a hierarchy, in which messages commonly travel from "superiors" to "inferiors." Increased modality distance could suggest a change in the hierarchy.

As we mentioned previously, the function p_{ij} need not be a simple count of the number of packets of information. One might well argue that a large relative change in the amount of communication between two vertices is more important than a large absolute change. In this case, p_{ij} could be taken as a logarithmic function. For example, if n_{ij} is the number of packets exchanged between vertices i and j, one might define $p_{ij} = 1 + \ln(n_{ij})$ when $n_{ij} > 0$. The 1 is added so that p_{ij} is positive; a zero entry will then correspond to zero communication.

Another possible change in the model is to incorporate a measure of the total communication involving a vertex, or the number of packets emanating from the vertex. The entry p_{ii} could be adjusted to become a measure of this.

The appropriateness of these possible modifications would depend on the application. One possible direction for future research is to test these ideas on various classes of real data.

10.7 Conclusion

In this chapter we proposed an application of the tournament scoring in analysis of communication networks. Tournament matrices record the win–loss outcomes of the matches in tournaments. Kendall–Wei ranking is an approach to ranking of tournament outputs using the quality of the players. The strength vector is approximated by a recursive sequence of vectors where the strength of a player i is proportional to the sum of the strengths of all the players that were beaten by i. The question of the convergence of that sequence is solved by a version of the Perron–Frobenius theorem for primitive matrices. That version states that for primitive matrices each eigenvalue is strictly dominated by the spectral radius (which is a simple zero of the characteristic polynomial). We introduced the snapshot matrices of individual members of time series of weighted digraphs that are defined and used for the implementation of the tournament scoring. Modality distance is defined and used to measure distance between consecutive members of the time series of digraphs. Some future directions for generalizing the measure (by varying the weight function) are discussed.

Part IV

Prediction and Advanced Distance Measures

11

Recovery of Missing Information in Graph Sequences

11.1 Introduction

Various procedures for the detection of anomalous events and network behavior were introduced in previous chapters of this book. In the current chapter we are going to address a different problem, viz., the recovery of missing information. Procedures for missing information recovery are useful in computer network monitoring in situations in which one or more network probes have failed. Here the presence, or absence, of certain nodes and edges is not known. In these instances, the network management system would be unable to compute an accurate measurement of network change. The techniques described in this chapter can be used to determine the likely status of this missing data and hence reduce false alarms of abnormal change.

This chapter is organized as follows. In the next section we address the problem of recovering missing information in a computer network using three different heuristic procedures that exploit graph context in time. In Section 11.3 an alternative approach to the recovery of missing information in a computer network is proposed. This approach makes use of decision tree classifiers. Finally, Section 11.4 draws conclusions and discusses potential future work.

11.2 Recovery of Missing Information in Computer Networks Using Context in Time

We consider a graph sequence $g_1, g_2, \ldots, g_{t-1}, g_t$ and assume that our knowledge of g_t is incomplete. That is, there exist one or several nodes and/or edges in g_t that were not observed in the graph sequence acquisition process. In other words, we don't know whether these nodes and/or edges are actually present in g_t. Assuming that all graphs under consideration have unique node labels, which is justified in the considered application, the behavior of an individual node x in the time series of graphs can be analyzed, and its presence or absence in some or all previous graphs in the sequence under consideration can be used to predict its presence or absence in graph g_t. A similar argument can be applied to the edges of graph g_t.

In the present section we will present three heuristic procedures for the recovery of missing information in a computer network based on graph context in time. We start by introducing the basic concepts and our notation in Section 11.2.1. Then the three strategies for information recovery will be described in Sections 11.2.2 to 11.2.4.

11.2.1 Basic Concepts and Notation

We consider graphs with unique node labels. To represent graphs with unique node labels in a convenient way, we drop set V and define each node in terms of its unique label. Hence a graph with unique node labels is represented by a 3-tuple $g = (L, E, \beta)$, where L is the set of node labels occurring in g, $E \subseteq L \times L$ is the set of edges, and $\beta : E \rightarrow L^{'}$ is the edge-labeling function. The terms "node label" and "node" will be used synonymously in the remainder.

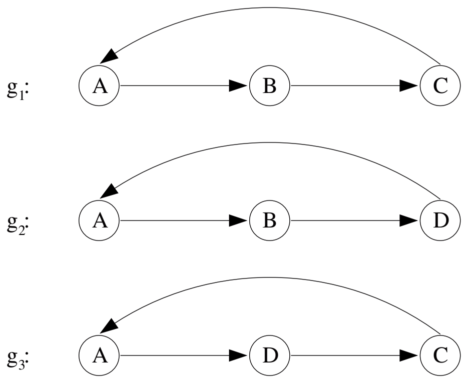


Fig. 11.1. An example of a graph sequence $s = g_1, g_2, g_3$.

In this chapter we will especially consider time series of graphs, i.e., graph sequences $s = g_1, g_2, \ldots, g_N$. The notation $g_i = (L_i, E_i, \beta_i)$ will be used to represent an individual graph g_i in sequence s; $i = 1, \ldots, N$. Motivated by computer network analysis applications we assume the existence of a universal set of node labels, or nodes, $\mathcal{L}$, from which all node labels that occur in a sequence s are drawn. That is, $L_i \subseteq \mathcal{L}$ for $i = 1, \ldots, N$ and $\mathcal{L} = \bigcup_{i=1}^{N} L_i$.[1] Given sequence $s = g_1, g_2, \ldots, g_N$, a subsequence of s is obtained by deleting the first i and the last j graphs from s, where $0 \leq i + j \leq N$. Thus $s' = g_{i+1}, \ldots, g_{N-j+1}$ is a subsequence of sequence s.[2]

As an example, consider sequence $s = g_1, g_2, g_3$, where graphs g_1, g_2, and g_3 are depicted in Figure 11.1. These graphs are formally represented as follows:

[1]In the computer network analysis application $\mathcal{L}$ will be, for example, the set of all unique IP host addresses in the network. Note that in one particular graph g_i, usually only a subset is actually present.

[2]The notation used here differs slightly from that of Chapter 8, where a subsequence as defined above is referred to as a *continuous* subsequence.

- $g_1 = (L_1, E_1, \beta_1)$; $L_1 = \{A, B, C\}$; $E_1 = \{(A, B), (B, C), (C, A)\}$;
- $g_2 = (L_2, E_2, \beta_2)$; $L_2 = \{A, B, D\}$; $E_2 = \{(A, B), (B, D), (D, A)\}$;
- $g_3 = (L_3, E_3, \beta_3)$; $L_3 = \{A, D, C\}$; $E_3 = \{(A, D), (D, C), (C, A)\}$.

We assume that $\beta_1 = \beta_2 = \beta_3 =$ const and omit the edge labels in Figure 11.1. In this example we have $\mathcal{L} = \{A, B, C, D\}$.

Given a time series of graphs $s = g_1, g_2, \ldots, g_N$ and its corresponding universal set of node labels $\mathcal{L}$, we can represent each graph $g_i = (L_i, E_i, \beta_i)$ in this series as a 3-tuple $(\gamma_i, \delta_i, \widehat{\beta_i})$ where:

- $\gamma_i : \mathcal{L} \to \{0, 1\}$ is a mapping that indicates whether node l is present in g_i. If l is present in g_i, then $\gamma_i(l) = 1$; otherwise $\gamma_i(l) = 0$.[3]
- $\delta_i : \mathcal{L}' \times \mathcal{L}' \to \{0, 1\}$ is a mapping that indicates whether edge (l_1, l_2) is present in g_i; here we choose $\mathcal{L}' = \{l \mid \gamma_i(l) = 1\}$, i.e., $\mathcal{L}'$ is the set of nodes that are actually present in g_i.
- $\widehat{\beta_i} : \mathcal{L}' \times \mathcal{L}' \to L'$ is a mapping that is defined as follows:
  $\widehat{\beta_i}(e) = \begin{cases} \beta_i(e), & \text{if } e \in \{(l_1, l_2) \mid \delta_i(l_1, l_2) = 1\}, \\ \text{undefined otherwise.} \end{cases}$

The definition of $\widehat{\beta_i}(e)$ means that each edge e that is in fact present in g_i will have label $\beta_i(e)$.

The 3-tuple $(\gamma_i, \delta_i, \widehat{\beta_i})$ that is constructed from $g_i = (L_i, E_i, \beta_i)$ will be called the *characteristic representation* of g_i, and denoted by $\chi(g_i)$. Clearly, for any given graph sequence $s = g_1, g_2, \ldots, g_N$ the corresponding sequence $\chi(s) = \chi(g_1), \chi(g_2), \ldots, \chi(g_N)$ can be easily constructed and is uniquely defined. Conversely, given $\chi(s) = \chi(g_1), \chi(g_2), \ldots, \chi(g_N)$ we can uniquely reconstruct $s = g_1, g_2, \ldots, g_N$.

As an example, consider graphs g_1, g_2, and g_3 in Figure 11.1. As mentioned before, $\mathcal{L} = \{A, B, C, D\}$. The following characteristic representations are obtained:

- $\chi(g_1) = (\gamma_1, \delta_1, \widehat{\beta_1})$ where
  $\gamma_1 : A \mapsto 1, B \mapsto 1, C \mapsto 1, D \mapsto 0$
  $\delta_1 : (A, B) \mapsto 1, (B, C) \mapsto 1, (C, A) \mapsto 1, (B, A) \mapsto 0, (C, B) \mapsto 0, (A, C) \mapsto 0$
  $\widehat{\beta_1} : (A, B) \mapsto$ const, $(B, C) \mapsto$ const, $(C, A) \mapsto$ const;
  $\widehat{\beta_1}(x, y)$ undefined for any other $(x, y) \in \{A, B, C\} \times \{A, B, C\}$
- $\chi(g_2) = (\gamma_2, \delta_2, \widehat{\beta_2})$ where
  $\gamma_2 : A \mapsto 1, B \mapsto 1, C \mapsto 0, D \mapsto 1$
  $\delta_2 : (A, B) \mapsto 1, (B, D) \mapsto 1, (D, A) \mapsto 1, (B, A) \mapsto 0, (D, B) \mapsto 0, (A, D) \mapsto 0$
  $\widehat{\beta_2} : (A, B) \mapsto$ const, $(B, D) \mapsto$ const, $(D, A) \mapsto$ const;
  $\widehat{\beta_2}(x, y)$ undefined for any other $(x, y) \in \{A, B, D\} \times \{A, B, D\}$
- $\chi(g_3) = (\gamma_3, \delta_3, \widehat{\beta_3})$ where
  $\gamma_3 : A \mapsto 1, B \mapsto 0, C \mapsto 1, D \mapsto 1$
  $\delta_3 : (A, D) \mapsto 1, (D, C) \mapsto 1, (C, A) \mapsto 1, (D, A) \mapsto 0, (C, D) \mapsto 0, (A, C) \mapsto 0$

[3]One can easily verify that $\{l \mid \gamma_i(l) = 1\} = L_i$.

$\widehat{\beta_3} : (A, D) \mapsto \text{const}, (D, C) \mapsto \text{const}, (C, A) \mapsto \text{const};$
$\widehat{\beta_3}\,(x, y)$ undefined for any other $(x, y) \in \{A, C, D\} \times \{A, C, D\}$

In this chapter we will pay particular attention to graph sequences with missing information. There are two possible cases of interest. First it may not be known whether node l is present in graph g_i. In other words, in $\chi\,(g_i)$, it is not known whether $\gamma_i\,(l) = 1$ or $\gamma_i\,(l) = 0$. Secondly, it may not be known whether edge (l_1, l_2) is present in g_i, which is equivalent to not knowing, in $\chi\,(g_i)$, whether $\delta_i\,(l_1, l_2) = 1$ or $\delta_i\,(l_1, l_2) = 0$.

To cope with the problem of missing information and in order to make our notation more convenient, we extend functions γ and δ in the characteristic representation $\chi\,(g)$ of graph $g = (L, E, \beta)$ by including the special symbol ? in the range of values of each function to indicate the case of missing information. That is, we write $\gamma\,(l) =?$ if it is unknown whether node l is present in g. Similarly, the notation $\delta\,(l_1, l_2) =?$ will be used to indicate that it is not known whether edge (l_1, l_2) is present in g. To keep our notation simple we will not explicitly distinguish between functions γ and δ as originally introduced and their extended versions that include the symbol ? in their range of values, and we will refer to the 3-tuple $\chi\,(g) = (\gamma, \delta, \widehat{\beta})$ as the characteristic representation of graph g regardless of whether the original functions or their extended versions are used.

Since any existing edge $(l_1, l_2) \in E$ requires the existence of both incident nodes $l_1, l_2 \in L$, we assume condition $\gamma\,(l_1) = \gamma\,(l_2) = 1$ if $\delta\,(l_1, l_2) = 1$ always being fulfilled to ensure the consistency of any graph g.

11.2.2 Recovery of Missing Information Using a Voting Procedure

In Sections 11.2.2 to 11.2.4 we introduce three simple heuristic procedures that are all based on the idea of using information about the behavior of a particular node or an edge along the time axis in order to predict its presence or absence in a particular graph.

Consider graph sequence $s = g_1, g_2, \ldots, g_t$ and let $\mathcal{L}$ denote the underlying universal set of node labels. Furthermore, consider graph $g_t = (L_t, E_t, \beta_t)$ with characteristic representation $\chi\,(g_t) = (\gamma_t, \delta_t, \widehat{\beta_t})$ and assume that $\gamma_t\,(l) =?$ for some $l \in \mathcal{L}$. In order to make a statement about the possible presence or absence of node l in graph g_t we consider a subsequence, or time window, $s' = g_{t-M}, \ldots, g_{t-1}$ of length M. The length of the sequence, M, is actually a parameter that can be tuned to the considered application. The basic idea is to utilize information about node l in the graphs belonging to subsequence s', in order to make a statement about the presence or absence of l in graph g_t. A simple approach consists in computing the relative frequency of occurrence of node l in subsequence s' and using this value for the decision to be made. Let k_1 be the number of graphs in subsequence s' in which node l is actually present. In other words, k_1 is the number of graphs g in subsequence s' for which $\gamma\,(l) = 1$. Clearly $0 \leq k_1 \leq M$. Similarly, assume that k_0 is the number of graphs g in subsequence s' for which $\gamma\,(l) = 0$. Similarly to k_1, we observe $0 \leq k_0 \leq M$. Obviously, there are $0 \leq M - (k_0 + k_1) \leq M$ graphs g in subsequence s' where $\gamma\,(l) =?$. Given parameters k_0 and k_1, we can use the following rule to make a decision as to the presence of node l in g_t:

$$\gamma_t(l) = \begin{cases} 0 \text{ if } k_0 > k_1, \\ 1 \text{ if } k_1 > k_0 . \end{cases} \tag{11.1}$$

In case $k_0 = k_1$, a random decision is in order. Alternatively, such a tie can be broken by some other scheme, for example by computing k_0 and k_1 for another value of M. Equation (11.1) can be interpreted as a majority voting rule. We decide $\gamma_t(l) = 0$ if the majority of graphs in subsequence s' excludes node l, and we decide $\gamma_t(l) = 1$ if node l is present in the majority of graphs in s'. Note that a graph g in sequence s where $\gamma(l) = ?$ will not be considered under this voting rule.

A potential problem with the decision rule according to equation (11.1) occurs if $k_0 + k_1 = 0$, i.e., if in each graph g of subsequence s' we have $\gamma_t(l) = ?$. In this case one has to resort to making a random decision, or possibly enlarge the length of subsequence s', i.e., increase the value of parameter M. The second possibility is motivated by the expectation to find some graph g in subsequence $g_1, \ldots, g_{t-M-1}$ in which $\gamma(l) = 1$ or $\gamma(l) = 0$, resulting in a value $k_0 > 0$ and/or $k_1 > 0$.

The decision scheme implied by equation (11.1) can be combined with a rejection mechanism. Under such a rejection mechanism a decision as to $\gamma_t(l) = 0$ or $\gamma_t(l) = 1$ will be made only if a certain level of confidence is reached. For example, one could replace the condition $k_0 > k_1$ by the more rigorous condition $k_0 > k_1 + \vartheta$, where $\vartheta > 0$ is a user-defined parameter. This means that a mere majority is no longer sufficient in order to decide for $\gamma_t(l) = 0$. The decision $\gamma_t(l) = 0$ will be made only if counters k_0 and k_1 differ by an amount larger than ϑ from each other. Similarly, the condition $k_1 > k_0$ would be replaced by $k_1 > k_0 + \vartheta$. In case neither $k_0 > k_1 + \vartheta$ nor $k_1 > k_0 + \vartheta$ holds, the system would abstain from making a decision and leave the value of $\gamma_t(l)$ undecided, i.e., yield $\gamma_t(l) = ?$ as the final result. Note that the original decision rule as given in equation (11.1) is a special case of the decision rule with rejection if $\vartheta = 0$. Intuitively, it may be expected that with an increasing value of ϑ, the relative number of correct decisions will increase, but the number of abstentions will increase as well.

Similar decision procedures can be derived for edges e in graph g_t. That is, given edge e with $\delta(e) = ?$, we count the number of graphs g in subsequence s' with $\delta(e) = 0$ and the number of graphs g with $\delta(e) = 1$ and decide, based on these two numbers, whether $\delta_t(e) = 0$ or $\delta_t(e) = 1$.

In case information about more than one node l or one edge e is missing in graph g_t, we can apply the procedures described above to all affected nodes and edges of g_t in parallel. In the extreme case, these procedures can even be applied when information about the complete graph g_t is missing, i.e., when $\gamma_t(l) = ?$ for all nodes l of g_t and $\delta_t(e) = ?$ for all edges e of g_t.

We conclude this subsection with an example. A graph sequence $s = g_1, g_2, g_3, g_4$ is shown in Figure 11.2. Assume that $t = 4$ and $M = 3$. This means that we use subsequence $s' = g_1, g_2, g_3$ for reconstructing missing information in g_4. First we note that $\mathcal{L} = \{A, B, C, D\}$. In Figure 11.2, dashed circles and lines represent missing information. For example, in graph g_1, we observe $\gamma_1(A) = \gamma_1(D) = 1$, $\gamma_1(B) = \gamma_1(C) = 0$, $\delta_1(A, D) = 1$ and $\delta_1(D, A) = 0$. In graph g_2, $\gamma_2(A) = \gamma_2(D) = 1$, $\gamma_2(B) = 0$, $\gamma_2(C) = ?$, $\delta_2(A, D) = 1$ and $\delta_2(D, A) = 0$, $\delta_2(A, C) = \delta_2(C, A) = \delta_2(C, D) = \delta_2(D, C) = ?$. Moreover, $\gamma_3(A) = 0$, $\gamma_3(B) = \gamma_3(C) = 1$, $\gamma_3(D) = ?$, $\delta_3(B, C) = \delta_3(C, B) = 1$, $\delta_3(C, D) = \delta_3(D, C) = \delta_3(B, D) = \delta_3(D, B) = 0$. In graph g_4, we observe $\gamma_4(l) = ?$ for any $x \in \mathcal{L}$, and $\delta_4(l_1, l_2) = ?$ for any $(l_1, l_2) \in \mathcal{L} \times \mathcal{L}$. This means that we don't have any information about graph g_4. Assume the task is to recover the missing information for graph g_4. Applying rule (2.2.1) results in the following decisions: $\gamma_4(A) = \gamma_4(D) = 1$; $\gamma_4(B) = 0$; for node C we have $k_0 = k_1 = 1$ and therefore make a random decision, for example, $\gamma_4(C) = 0$. Furthermore, we get for the edges of g_4 $\delta_4(A, D) = 1$ and $\delta_4(D, A) = 0$. A graphical representation of graph g_4 after recovering the missing information is shown in Figure 11.3.

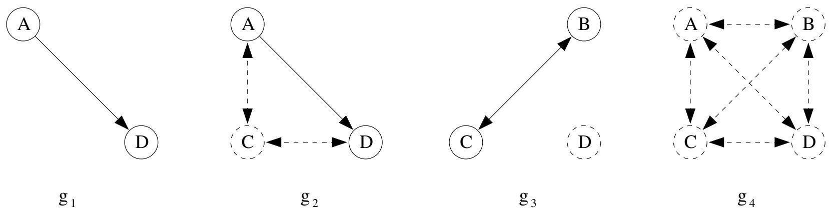


Fig. 11.2. Example of a graph sequence $s = g_1, g_2, g_3, g_4$ with missing information.

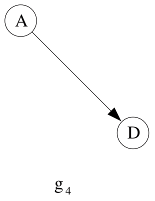


Fig. 11.3. Result of information recovery procedure when applied to g_4 in Figure 11.2.

11.2.3 Recovery of Missing Information Using Reference Patterns

The method described in Section 11.2.2 is based on a simple voting scheme that computes the number of graphs g in subsequence s' where $\gamma(l) = 0$ and compares this number with the number of graphs where $\gamma(l) = 1$. The particular order of the values $\gamma(l) = 0$ and $\gamma(l) = 1$ in subsequence s' is not relevant. In Section 11.2.3 we develop a more refined decision rule in which this order is taken into account. We address again the problem of making a decision as to $\gamma_t(l) = 0$ or $\gamma_t(l) = 1$, given $\gamma_t(l) = ?$ in sequence s.

As a generalization of the scenario considered in Section 11.2.2 we assume the existence of a reference set

$$R = \{s_1, \ldots, s_n\} \tag{11.2}$$

of graph subsequences

$$s_j = g_{j,1}, \ldots, g_{j,M} \tag{11.3}$$

of length M for each $j = 1, \dots, n$.

Each element s_j of the reference set is a sequence of graphs of length M. These sequences are used to represent information about the "typical behavior" of the nodes and edges in a graph sequence of length M. This information will be used to make a decision as to $\gamma_t(l) = 0$ or $\gamma_t(l) = 1$ whenever $\gamma_t(l) =?$ occurs.

To generate reference set R, we can utilize graph sequence $g_1, \dots, g_{t-1}$. Each sequence in R is of length M, by definition (see equation (11.3)), where M is a parameter to be chosen dependent on the particular application. Let's assume that $M \leq t-1$. Then we can extract all subsequences of length M from sequence $g_1, \dots, g_{t-1}$, and include them in reference set R. This results in

$$R = \{s_1 = g_1, \dots, g_M \ ; \ s_2 = g_2, \dots, g_{M+1} \ ; \ s_{t-M} = g_{t-M}, \dots, g_{t-1}\} \ . \tag{11.4}$$

From each sequence $s_i = g_i, \dots, g_{i+M-1}$ in set R we can furthermore extract, for each node $l \in \mathcal{L}$, the sequence $\gamma_i(l), \dots, \gamma_{i+M-1}(l)$. Assume for the moment that $\gamma_i(l), \dots, \gamma_{i+M-1}(l) \in \{0, 1\}$, which means that none of the elements $\gamma_i(l), \dots, \gamma_{i+M-1}(l)$ is equal to ?. Then $(\gamma_i(l), \dots, \gamma_{i+M-1}(l))$ is a sequence of binary numbers, 0 or 1, that indicate whether node l occurs in a particular graph in sequence s_i. Such a sequence of binary numbers will be called a *reference pattern*. Obviously $(\gamma_i(l), \dots, \gamma_{i+M-1}(l)) \in \{0, 1\}^M$. Because there are 2^M different binary sequences of length M, there exist at most 2^M different reference patterns for each node $l \in \mathcal{L}$. Note that a particular reference pattern $x = (x_1, \dots, x_M) \in \{0, 1\}^M$ may have multiple occurrences in set R.

In order to make a decision as to $\gamma_t(l) = 0$ or $\gamma_t(l) = 1$, given $\gamma_t(l) =?$, the following procedure can be adopted. First, we extract from graph sequence $s = g_1, \dots, g_t$ the sequence $(\gamma_{t-M+1}(l), \dots, \gamma_t(l))$ where, according to our assumption, $\gamma_t(l) =?$. Assume furthermore that $\gamma_{t-M+1}(l), \dots, \gamma_t(l) \in \{0, 1\}$, i.e., none of the elements in sequence $(\gamma_{t-M+1}(l), \dots, \gamma_t(l))$, except $\gamma_t(l)$, is equal to ?. Sequence $(\gamma_{t-M+1}(l), \dots, \gamma_t(l))$ will be called the *query pattern*. Given the query pattern, we retrieve from the reference set R all reference patterns $x = (x_1, \dots, x_M)$ where $x_1 = \gamma_{t-M+1}(l)$, $x_2 = \gamma_{t-M+2}(l), \dots, x_{M-1} = \gamma_{t-1}(l)$. Any reference pattern x with this property is called a *reference pattern matching the query pattern*, or a *matching reference pattern* for short. Clearly, a reference pattern that matches the query pattern is a sequence of 0's and 1's of length M, where the first $M-1$ elements are identical to corresponding elements in the query pattern. The last element in the query pattern is equal to ?, by definition, while the last element in any matching reference pattern is either 0 or 1. Let k be the number of reference patterns that match the query pattern. Furthermore, let k_0 be the number of matching reference patterns with $x_M = 0$, and let k_1 be the number of matching reference patterns with $x_M = 1$; note that $k = k_0 + k_1$. Now we can apply the following decision rule:

$$\gamma_t(l) = \begin{cases} 0 \text{ if } k_0 > k_1, \\ 1 \text{ if } k_1 > k_0 \ . \end{cases} \tag{11.5}$$

In case $k_0 = k_1$ a random decision is in order, similarly to the case $k_0 = k_1$ in equation (11.1). Intuitively, under this decision rule we consider the history of node l

over a time window of length M and retrieve all cases recorded in set R that match the current history. Then a decision is made as to $\gamma_t(l) = 0$ or $\gamma_t(l) = 1$, depending on which case occurs more frequently in the reference set. From the intuitive point of view, this approach is based on the assumption that not only the number of occurrences, but also the patterns of presence and absence of a node l in a time window of a certain length has a correlation with the occurrence of l in g_t.

As an example, consider graph sequence $s = g_1, \ldots, g_9$ shown in Figure 11.4. Let $t = 9$ and assume we want to make a decision as to $\gamma_9(A) = 0$ or $\gamma_9(A) = 1$. Let $M = 2$. Then we get the following reference set:

$$R = \{(g_1, g_2), (g_2, g_3), \ldots, (g_7, g_8)\} \ . \tag{11.6}$$

From this set, we can extract the following reference patterns for node A:

$(0, 0)$: 1 instance extracted from (g_1, g_2)
$(0, 1)$: 2 instances extracted from (g_2, g_3) and (g_4, g_5)
$(1, 0)$: 2 instances extracted from (g_3, g_4) and (g_7, g_8)
$(1, 1)$: 2 instances extracted from (g_5, g_6) and (g_6, g_7)

The query pattern in this example is $(\gamma_8(A), \gamma_9(A)) = (0, ?)$. There are three matching reference patterns, namely the single instance of $(0, 0)$ and the two instances of $(0, 1)$. Hence $k = 3$, $k_0 = 1$, $k_1 = 2$, and the result will be $\gamma_9(A) = 1$.

The method derived until now is based on the assumption that none of the reference patterns for node l, extracted from set R, contains the symbol ?. From the practical point of view, this implies that whenever the situation $\gamma_i(l) =?$ is actually encountered, we have to discard the corresponding reference pattern, which may result in a set of reference patterns too small to be meaningful. However, the restriction that symbol ? must not occur in a reference pattern can be overcome as described below. Consider reference set R as defined in equation (11.4) and assume that in fact $\gamma_i(l) =?$ for some i, $1 \le i < t$. Then there will be reference patterns for node l that include symbol ?. Let $x = (x_1, \ldots, x_{i-1}, ?, x_{i+1}, \ldots, x_M)$ be such a reference pattern. In order to eliminate symbol ? from the reference pattern, we replace x by two new reference patterns x^0 and x^1 where x^0 is obtained from x by replacing symbol ? by symbol 0, and x^1 is obtained by replacing ? by symbol 1. That is, $x^0 = (x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_M)$ and $x^1 = (x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_M)$. This schema can be iteratively applied in case there is more than one position in x equal to ?. Generally, if there are r, $1 \le r \le M$, positions equal to ?, we will replace the original reference pattern by 2^r new reference patterns, taking all combinations into account to substitute symbol 0 or symbol 1 for symbol ?. As an example, consider reference pattern $x = (?, ?, ?)$, which will be replaced by eight new reference patterns, $x^{000} = (0, 0, 0)$, $x^{001} = (0, 0, 1)$, ..., $x^{111} = (1, 1, 1)$.

Once all occurrences of symbol ? have been eliminated from all reference patterns for node l, we assign a weight to each new reference pattern. The weight is equal to $1/2^r$, where r is the number of symbols equal to ? in the original reference pattern (equivalently, 2^r is the number of new reference patterns obtained by substitution from the original reference pattern). In the previous example, where $x = (?, ?, ?)$, each of the new reference patterns $x^{000}, x^{001}, \ldots, x^{111}$ gets a weight equal to $1/8$. In case

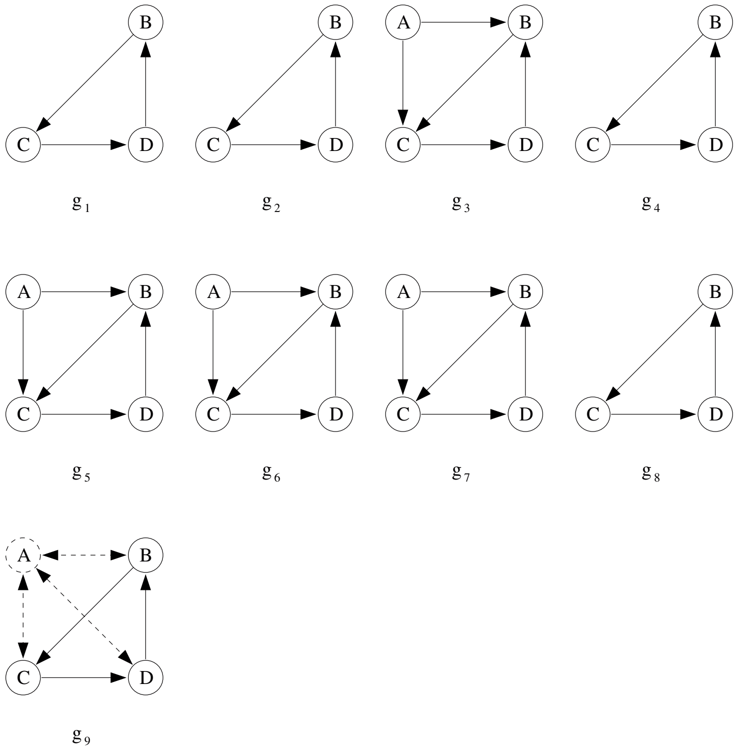


Fig. 11.4. An example used to demonstrate the method introduced in Section 11.2.3.

no substitution operation was applied to a reference pattern x (which means that this reference pattern never included an occurrence of symbol ?) we assign weight 1 to x.

Once all symbols ? have been eliminated and weights have been assigned to all reference patterns of node l, we apply the following modified rule in order to decide as to $\gamma_t(l) = 0$ or $\gamma_t(l) = 1$, given $\gamma_t(l) =?$. The numbers k_0 and k_1 in equation (11.5) are no longer used to reflect the number of matching reference patterns with $x_M = 0$ and $x_M = 1$, respectively, but are now equal to the sum of weights of all reference patterns with $x_M = 0$ and $x_M = 1$, respectively. With this modified definition of k_0 and k_1, equation (11.5) is applied.

As an example, consider the graph sequence in Figure 11.4 and assume $\gamma_6(A) =?$ rather than $\gamma_6(A) = 1$. For this situation we get the following reference patterns and weights for node A:

(0, 0) : weight 1, resulting from (g_1, g_2) contributing with weight 1
(0, 1) : weight 2.5, resulting from (g_2, g_3) and (g_4, g_5), each contributing with weight 1, and (g_6, g_7) contributing with weight 0.5
(1, 0) : weight 2.5, resulting from (g_3, g_4) and (g_7, g_8), each contributing with weight 1, and (g_5, g_6) contributing with weight 0.5
(1, 1) : weight 1, resulting from (g_5, g_6) and (g_6, g_7), each contributing with weight 0.5

In order to derive a decision concerning $\gamma_9(A) = ?$, we obtain $k_0 = 1$ and $k_1 = 2.5$ under the generalized decision schema. Hence $\gamma_9(A) = 1$, which is identical to the decision made before.

There is still a restriction in the decision procedure derived until now, in the sense that query pattern $(\gamma_{t-M+1}(l), \ldots, \gamma_t(l))$ must not include any occurrence of symbol ?, except for $\gamma_t(l) = ?$. Yet this restriction can be overcome by a procedure that eliminates all occurrences of ? in a query pattern, similar to their elimination in a reference pattern. Assume that our query pattern is $y = (y_1, \ldots, y_M)$, where in addition to $y_M = ?$, we have r, $1 \le r \le M - 1$, other symbols equal to ?. In such a situation, we iteratively replace each occurrence of ? first by symbol 0 and then by symbol 1. Consequently, we get 2^r new query patterns, none of which contains symbol ? at a position other than M. For example, for query pattern $y = (?, ?, ?)$ we derive four new query patterns $y^{00} = (0, 0, ?)$, $y^{01} = (0, 1, ?)$, $y^{10} = (1, 0, ?)$, and $y^{11} = (1, 1, ?)$. (Notice that symbol ? is not replaced at position M, due to our assumption that the last symbol in each query pattern is equal to ?.) For each new query pattern we retrieve all matching reference patterns $x = (x_1, \ldots, x_M)$ from set R and simply count, using variables k_0 and k_1, how many reference patterns exist with $x_M = 0$ and $x_M = 1$, respectively. Then equation (11.5) is applied. Notice that in this schema no weights are needed for the new reference patterns that are generated. If there are weights assigned to the matching reference patterns that are retrieved, these weights will be summed up in the computation of k_0 and k_1, as explained before.

As an example, assume that the graph sequence shown in Figure 11.4 (with $\gamma_6(A) = 1$) is being extended by an additional graph g_{10} with $\gamma_{10}(A) = ?$. Let $t = 10$, $M = 2$, and assume we want to make a decision regarding $\gamma_{10}(A)$. Now the query sequence becomes $y = (y_1, y_2) = (\gamma_9(A), \gamma_{10}(A)) = (?, ?)$. The method described above yields $y^0 = (0, ?)$ and $y^1 = (1, ?)$. Given the (unweighted) reference patterns derived from Figure 11.4, where graph g_9 is excluded, we find $k_0 = 3$ (due to one instance of (0, 0) and two instances of (1, 0)) and $k_1 = 4$ (due to two instances of both (0, 1) and (1, 1)), which results in $\gamma_{10}(A) = 1$.

We can extend this example by including (g_8, g_9) in reference set R. Because $\gamma_9(A) = ?$ we use the weighting schema to derive the reference patterns for node A. As a result we obtain:

$(0, 0)$: weight 1.5, resulting from (g_1, g_2) contributing with weight 1, and (g_8, g_9) contributing with weight 0.5
$(0, 1)$: weight 2.5, resulting from (g_2, g_3) and (g_4, g_5), each contributing with weight 1, and (g_8, g_9) contributing with weight 0.5
$(1, 0)$: weight 2, resulting from (g_3, g_4) and (g_7, g_8), each contributing with weight 1
$(1, 1)$: weight 2, resulting from (g_5, g_6) and (g_6, g_7), each contributing with weight 1

From the original query pattern $y = (?, ?)$ we derive $y^0 = (0, ?)$ and $y^1 = (1, ?)$. To calculate k_0 we have to add the weights of reference patterns (0, 0) and (1, 0), and for k_1 the weights of (0, 1) and (1, 1), which results in $k_0 = 3.5$ and $k_1 = 4.5$. Hence we set $\delta_{10}(A) = 1$.

The procedures described so far aim at the recovery of missing information concerning graph nodes. However, using function $\delta(e)$, defined in Section 11.2.1, rather than $\gamma(l)$, they can be adapted to the recovery of missing edge information in a straightforward manner.

Some final remarks on the recovery scheme using reference patterns concern the actual implementation of the proposed method. Because of the exponential number of reference patterns, it is advisable to keep parameter M rather small. On the other hand, if M gets too small we take only a short history of node l into account when making our decision. Therefore, a suitable compromise has to be found, preferably through experimental evaluation over a range of possible parameter values. Depending on the length t of sequence $s = g_1, g_2, \ldots, g_t$ and on M, it may happen that certain binary sequences will not occur as reference patterns at all. In this case $k_0 = k_1 = 0$ in equation (11.5). Hence a random decision has to be made. As an alternative to making a random decision, one could repeat the procedure using a smaller value of M. Generally, if M gets smaller, a higher number of reference patterns is obtained (assuming constant t). Another possibility to increase the number of reference patterns is to pool the reference patterns of all nodes $l \in \mathcal{L}$. That is, rather than keeping an individual set of reference patterns for each node $l \in \mathcal{L}$, there is only one global set of reference patterns, derived from, and applicable to, all nodes $l \in \mathcal{L}$ in the same way. Such a pooling procedure seems adequate if all nodes in the network are believed to behave in the same way. As a final remark in this section, we want to point out that regardless of how many matching reference patterns exist for a given query pattern, one can always resort to the decision rule presented in Section 11.2.2 to avoid making a random decision in case $k_0 = k_1$. Using parameter ϑ in a similar way as described in Section 11.2.2 it is also possible to implement a rejection schema for the method described in Section 11.2.3.

11.2.4 Recovery of Missing Information Using Linear Prediction

Linear prediction is an established methodology in signal processing [136]. It can be used in particular to estimate future values of a function as a linear combination of previous samples.

Given a time series of numbers, $y_1, \ldots, y_{t-1}$, an estimate of y_t, $\widehat{y_t}$, can be calculated as follows:

$$\widehat{y_t} = \sum_{i=1}^{M} \alpha_i y_{t-i} \ . \tag{11.7}$$

In this formula the last M values $y_{t-M}, \ldots, y_{t-1}$ are used for the prediction of y_t, using M real-valued weighting coefficients $\alpha_1, \ldots, \alpha_M$. In linear prediction one chooses the weighting factors α_i in such a way that the estimate $\widehat{y_t}$ results in an error as small as possible. Hence, defining the error as

$$e_t = y_t - \widehat{y_t} = y_t - \sum_{i=1}^{M} \alpha_i y_{t-i} \ , \tag{11.8}$$

and summing the squared error over a past time window of length T yields the quantity

$$E = \sum_{i=0}^{T-1} e_i^2 = \sum_{i=0}^{T-1} \left(y_i - \sum_{j=1}^{M} \alpha_j y_{i-j} \right)^2 \ . \tag{11.9}$$

Now the aim is to choose the coefficients α_i in such a way that E is minimized. The minimum of E occurs when the derivative with respect to each parameter $\partial E / \partial \alpha_i$ is equal to zero. After some intermediate steps, the following solution is obtained [136]:

$$\boldsymbol{\alpha} = \Phi^{-1} \boldsymbol{\varphi_0} \ , \tag{11.10}$$

where:

- $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_M)'$ is the desired vector of coefficients
- Φ^{-1} is the inverse of the quadratic $M \times M$ matrix

$$\Phi = \begin{bmatrix} \varphi_{1,1} & \cdots & \varphi_{1,M} \\ \vdots & \ddots & \vdots \\ \varphi_{M,1} & \cdots & \varphi_{M,M} \end{bmatrix}$$

  with elements

$$\varphi_{r,s} = \sum_{i=0}^{T-1} y_{i-r} y_{i-s}$$

  for $r, s = 0, 1, \ldots, M$
- $\boldsymbol{\varphi_0} = \left(\varphi_{1,0}, \ldots, \varphi_{M,0}\right)'$ is a column vector of dimension M

It can be shown that matrix Φ is symmetric, and this symmetry can be exploited to speed up the inversion process [136].

In order to apply linear prediction to the recovery of missing information about node l in a time series of graphs $s = g_1, g_2, \ldots, g_t$ we just need to select appropriate values for parameters M and T and substitute values $\gamma_1(l), \ldots, \gamma_{t-1}(l)$ for $y_1, \ldots, y_{t-1}$ in the previous equations. Similarly, we can use sequence $\delta_1(e), \ldots, \delta_{t-1}(e)$ for the recovery of missing edge information. It is possible to dynamically recompute all α_i

values whenever the case $\gamma_t(l) =?$ or $\delta_t(e) =?$ occurs. Alternatively, one can compute these coefficients just a single time and use them in equation (11.7) whenever the case $\gamma_t(l) =?$ or $\delta_t(e) =?$ occurs. It has to be noted that in contrast with the methods described in Sections 11.2.2 and 11.2.3, linear prediction will not tolerate missing information in any of the graphs g_i used to derive the coefficients α_i. That is, the cases $\gamma_i(l) =?$ or $\delta_i(e) =?$ are not admissible if graph g_i is being used in the computation of $\boldsymbol{\alpha}$.

11.3 Recovery of Missing Information Using a Machine Learning Approach

The discipline of machine learning is concerned with the automatic inference of some function f from a set of training, or learning, data [134]. In the application considered in this chapter, our aim is to construct a function that computes $\gamma_t(l)$ and/or $\delta_t(e)$, given some data extracted from time series $g_1, g_2, \ldots, g_t$ as input. From the general point of view, machine learning is a rather broad discipline with several subfields that include, for example, statistical classification [67] and neural networks [148]. In the following we will consider one particular approach to decision making that is based on decision tree classifiers [146].

In Section 11.3.1 we will first review basic concepts of decision tree classifiers. Then, in Section 11.3.2, we will derive a procedure for the recovery of missing information using *within-graph context*. The procedure makes a decision as to $\gamma_t(l) = 0$ or $\gamma_t(l) = 1$, given $\gamma_t(l) =?$, using only nodes from graph g_t, but no nodes from other graphs $g_1, g_2, \ldots, g_{t-1}$ of the considered time series. More general scenarios will be discussed in Section 11.3.3.

11.3.1 Decision Tree Classifiers

Decision tree classifiers are normally used for the purpose of object classification. An object $\boldsymbol{x}$ is given in terms of the values of d different features and represented by means of a d-dimensional vector, i.e., $\boldsymbol{x} = (x_1, \ldots, x_d)$. The feature values x_i, $1 \leq i \leq d$, can be numerical or nonnumerical. It is possible that one or several feature values are unknown. To classify an object means to assign it to a class Ω_i out of a number of given classes $\Omega_1, \ldots, \Omega_c$.

Let us consider a simple example where the task consists in classifying fruits, depending on their color, size, shape, and taste. Here we have four features, *color*, *size*, *shape*, *taste* (i.e., $d = 4$), and we assume that these features can assume the following values:

- $color \in \{green, yellow, red\}$
- $size \in \{big, medium, small\}$
- $shape \in \{round, elongated\}$
- $taste \in \{sweet, sour\}$

Hence a fruit is characterized by a 4-dimensional vector $\boldsymbol{x} = (x_1, \ldots, x_4)$, where x_1 is the value of feature $color$, x_2 is the value of feature $size$, x_3 is the value of feature $shape$, and x_4 is the value of feature $taste$. We note that all feature values are nonnumerical in this example. A particular example of a fruit is $\boldsymbol{x} = (yellow, small, round, sour)$. Object $\boldsymbol{y} = (yellow, big, round, ?)$ reflects the case in which no information about the feature $taste$ is available.

A *decision tree* is a tree with the following properties:

- Each leaf of the tree represents exactly one object class.
- Each nonleaf node n of the tree, including the root node, represents a test, which uses exactly one feature. For each possible outcome of the test, there is one edge leading from node n to a child node of n.

An example of a decision tree that can be used to classify fruits is shown in Figure 11.5. The different object classes underlying our example are

- $\Omega_1 =$ *watermelon*
- $\Omega_2 =$ *apple*
- $\Omega_3 =$ *grape*
- $\Omega_4 =$ *grapefruit*
- $\Omega_5 =$ *lemon*
- $\Omega_6 =$ *banana*
- $\Omega_7 =$ *cherry*

Given a decision tree, such as the one shown in Figure 11.5, and an unknown input object $\boldsymbol{x}$ to be classified, we simply apply the tests represented by the nonleaf nodes of the tree, starting with the test represented by the root, and traverse the tree top-down toward the leaves, according to the outcome of each test. Once a leaf has been reached, the class Ω_i represented by that leaf is assigned to the unknown input object $\boldsymbol{x}$. As an example, consider the object $\boldsymbol{x} =$ (*yellow, small, round, sour*). Classification of this object leads to the leaf node that represents object class $lemon$. Therefore, object $\boldsymbol{x}$ is classified as $lemon$. Note that for this decision the value of the attribute *taste* has not been used.

From the decision tree shown in Figure 11.5 it can be concluded that the same class may occur at different leaf nodes. This simply means that objects of the same class may have different feature values, or intuitively speaking, different appearance. An example in Figure 11.5 is the class *grape*, which occurs two times. We also note that the same test might occur multiple times, at different nonleaf nodes, in the same decision tree. For example, there are three different nonleaf nodes in Figure 11.5 that all test attribute $size$.

Given a decision tree, such as the one shown in Figure 11.5, and an unknown object, for example $\boldsymbol{x} =$ (*yellow, small, round, sour*), the classification of $\boldsymbol{x}$ is accomplished easily by a straightforward traversal of the decision tree. A more difficult question is how the decision tree is obtained. Clearly, one possibility is to have the decision tree built by a human expert, in a "manual" fashion based on his or her expertise. However, such a manual decision tree construction has clear limitations, for example if many features or many classes are involved. Also, for certain applications, there may be a lack of human expertise. In the following we introduce a procedure that allows us to infer a decision tree automatically from a set of examples. This set of examples is called a *learning* or *training set* in machine learning, and it is conceptually similar to the reference set R used in Section 11.2.3.

A training set is a set of objects, $\boldsymbol{x} = (x_1, \ldots, x_d)$, where the class of each object in the training set is known. There are several algorithms for the inference of a decision tree from a learning set that are similar to each other. In the following we describe an approach closely related to C4.5 [146]. It is a procedure that recursively splits the training set into smaller subsets, according to the possible outcomes of the tests represented by the nodes of the decision tree, i.e., the values of a chosen attribute. The procedure starts with the whole training set and terminates once a subset contains only elements that belong to the same class.

A pseudocode description of the procedure for decision tree learning is given in Figure 11.6. As an example, consider set $L = \{\boldsymbol{x_1}, \ldots, \boldsymbol{x_9}\}$ shown in Figure 11.7. We observe that L contains elements from different classes. Hence case 1 applies, and the algorithm generates a node for set L; this node will actually become the root node of the decision tree. Assume that the *best feature* is *color*. We assign this feature as a test to the node corresponding to L. The different values of *color*, viz. *green*, *yellow*, and *red*, split L into three subsets, $L_1 = \{\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{x_3}\}$, $L_2 = \{\boldsymbol{x_4}, \boldsymbol{x_5}, \boldsymbol{x_6}\}$, and $L_3 = \{\boldsymbol{x_7}, \boldsymbol{x_8}, \boldsymbol{x_9}\}$, respectively. For a graphical illustration see Figure 11.8. For each subset we generate an edge that leaves the node corresponding to L. Then we recursively apply procedure decision-tree-inference to each of L_1, L_2, L_3. At the node corresponding to L_1, case 1 applies. Assume *best feature* is $size$. This splits L_1 into $L_{11} = \{x_1\}$, corresponding to $size = big$, $L_{12} = \{x_2\}$, corresponding to $size = medium$, and $L_{13} = \{x_3\}$, corresponding to $size = small$. We generate the corresponding edges and continue with L_{11}. Since this subset contains only a single element, case 2 applies. We generate a leaf node for L_{11} and label it with $\Omega_1 = watermelon$. Similarly, we generate a leaf node for L_{12} and a leaf node for L_{13} and label it with $\Omega_2 = apple$ and $\Omega_3 = grape$, respectively. It is easy to verify that by continuing this procedure we get the tree shown in Figure 11.8, assuming that the following features will be chosen as *best feature*: $shape$ for L_2, $size$ for L_3, $size$ for $L_{21} = \{x_4, x_5\}$, and $taste$ for $L_{32} = \{x_8, x_9\}$. Dropping sets L, L_1, L_2, L_3, L_{11}, and so on from the tree and keeping only the *best feature* as a test at each nonleaf node renders the decision tree shown in Figure 11.5.

An important question in decision tree induction is how the *best feature* is found at each nonleaf node. The basic idea is to seek the feature that contributes most toward the *purity* of the resulting subsets. At any stage of the execution of the decision tree induction algorithm shown in Figure 11.6, a training set L is called *pure* if all its elements are from the same class. On the other hand, it is *impure* if different classes are represented in L. A quantity that is suitable to formally model the concept of purity is entropy. The entropy of training set L is given by

$$E(L) = -\sum_{i=1}^{c} p(\Omega_i) \log_2 p(\Omega_i) \ . \tag{11.11}$$

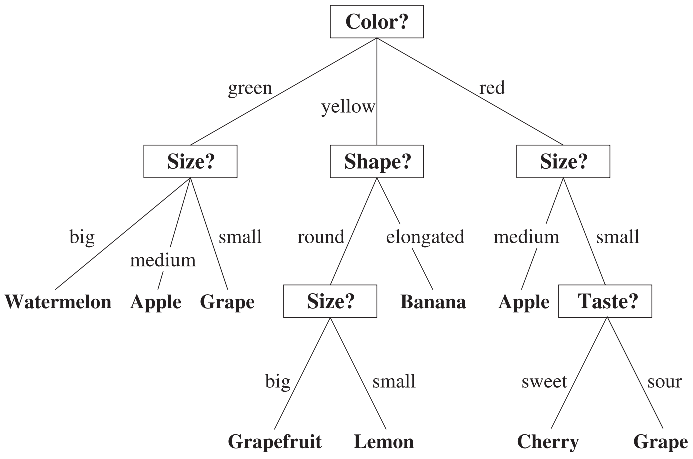


Fig. 11.5. Example of decision tree.

```
decision_tree_inference($L$)

input: learning set $L$ where the class of each object is known
output: decision tree
begin
case 1: the learning set $L$ includes objects from different classes; in this case do
  1. generate a decision tree node $N$ for $L$
  2. choose the best feature $x_i$ assign it as a test to $N$, and divide set $L$ into disjoint subsets $L_1, L_2,...,L_k$ corresponding to the different values $v_1,v_2,...,v_k$ of $x_i$
  3. for each value $v_j$, of $x_i$ do
     (a) generate an edge to the child node of $N$ corresponding to the value $v_j$
     (b) execute decision_tree_inference($L_j$)
case 2: the learning set $L$ includes objects from only a single class, $\Omega_i$; in this case generate a leaf node for $L$ and assign class $\Omega_i$ to it
end
```

Fig. 11.6. Procedure for decision tree inference.

In this formula, c is the number of classes and $p(\Omega_i)$ is the probability of class Ω_i occurring in L. This probability is usually computed by dividing the number of elements from Ω_i in L by the total number of elements in L. For example, if $L = \{x_1, x_2, x_3, x_4\}$, $x_1, x_2 \in \Omega_1$, $x_3 \in \Omega_2$, $x_4 \in \Omega_3$ then $p(\Omega_1) = 0.5$, $p(\Omega_2) = p(\Omega_3) = 0.25$. It is known that $E(L) \geq 0$, and $E(L) = 0$ if and only if all elements in L are from the same class. On the other hand, the maximum value of $E(L)$ occurs if and only if the probabilities of all classes Ω_i in L are the same, which means that $p(\Omega_i) = 1/c$ for $i = 1, \ldots, c$. Note that maximum and minimum purity coincides with minimum and maximum entropy, respectively.

Given a training set L, in order to find the *best feature* at a particular node in the decision tree we probe each feature x_i by computing the weighted entropy of the successor nodes that result if L is split into subsets $L_1, L_2, \ldots, L_k$ depending on the k different values of x_i. More precisely, the expression

$$E = \sum_{j=1}^{k} \frac{E(L_j)}{|L_j|} \tag{11.12}$$

is computed for each feature x_i, and the feature that minimizes E is taken as the *best feature*. Clearly, this minimization strategy is equivalent to maximizing the purity among the training subsets, which makes sense because we require each leaf node to be eventually produced being completely pure.

As an example, consider the decision tree in Figure 11.8 and assume we want to find the *best feature* for the root node, which corresponds to the training set $L = \{x_1, \ldots, x_9\}$. Evaluation of feature *color* gives

$$E = \frac{1}{3}E(L_1) + \frac{1}{3}E(L_2) + \frac{1}{3}E(L_3) = \frac{1}{3}\left(-\log\frac{1}{3} - \log\frac{1}{3} - \log\frac{1}{3}\right) = -\log\frac{1}{3}\,.$$

We compute E in the same manner for all other features, i.e., *size, shape*, and *taste*, and choose that feature as best feature that yields the smallest value of E.[4] The same procedure is repeated at all other nonleaf nodes of the decision tree. Note that for any of the other nonleaf nodes we test all features, even in case a feature was already chosen as *best feature* at a predecessor of the current node in the tree.

$\mathbf{x}_1$ = (green, big, round, sweet) = Watermelon
$\mathbf{x}_2$ = (green, medium, round, sour) = Apple
$\mathbf{x}_3$ = (green, small, round, sweet) = Grape
$\mathbf{x}_4$ = (yellow, big, round, sour) = Grapefruit
$\mathbf{x}_5$ = (yellow, small, round, sour) = Lemon
$\mathbf{x}_6$ = (yellow, small, elongated, sweet) = Banana
$\mathbf{x}_7$ = (red, medium, round, sweet) = Apple
$\mathbf{x}_8$ = (red, small, round, sweet) = Cherry
$\mathbf{x}_9$ = (red, small, round, sour) = Grape

Fig. 11.7. Training set for inference of the decision tree shown in Figure 11.5.

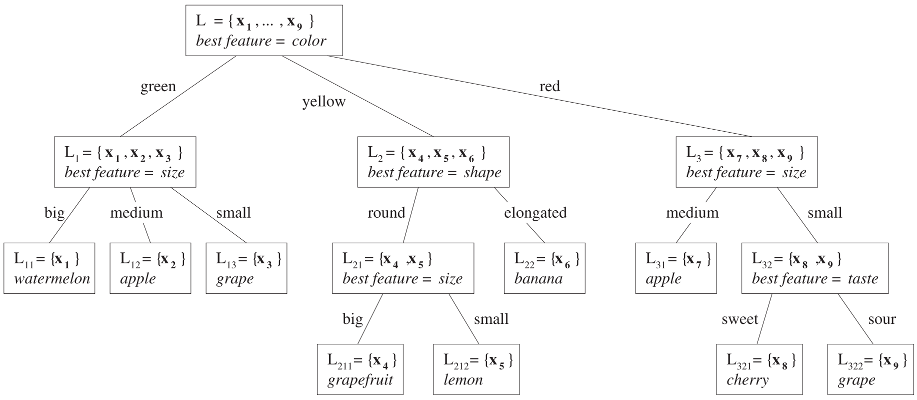
There are more issues that need to be addressed before a decision tree classifier can be actually applied to a practical problem. One of these issues is how to deal with

[4]Note that for the decision tree shown in Figure 11.8, *best feature* was chosen randomly. That is, the entropy minimization procedure based on equation (11.12) was not used in Figure 11.8.

unknown feature values that may occur in the training set and/or the unknown input objects to be classified. Furthermore, it may happen during decision tree construction that elements from different classes end up in the same leaf node. In this case there exist no features that allow us to discriminate between these elements. Such a case, where two identical objects belong to different classes, is not uncommon in real-world applications.[5]

Another issue is decision tree pruning in order to avoid overfitting. Usually, the aim of the algorithm described in Figure 11.6 is to produce a decision tree that is used as a classifier on future input objects. In particular, the classifier should work well on new input objects that are not included in the training set. That is, we have to expect that a new input object is different from any of the training objects used to build the tree. It is well known that decision trees that are overadapted, or overfit, to the given training set tend to have a rather poor performance on new, unseen data.[6] To avoid overfitting, some pruning strategies are available. They typically cut off some branches after a decision tree has been generated, or they avoid generation of such branches from the beginning.

For a detailed treatment all of these issues we refer to [146] and Chapter 3 on decision tree learning in [134]. There are several software packages available that include all functionality needed to implement decision tree classifiers for a variety of applications, including techniques to deal with unknown feature values and to avoid overfitting.



$L = \{\mathbf{x}_1, \dots, \mathbf{x}_9\}$
best feature = color
green
yellow
red
$L_1 = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$
best feature = size
$L_2 = \{\mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6\}$
best feature = shape
$L_3 = \{\mathbf{x}_7, \mathbf{x}_8, \mathbf{x}_9\}$
best feature = size
big
medium
small
round
elongated
medium
small
$L_{11} = \{\mathbf{x}_1\}$
watermelon
$L_{12} = \{\mathbf{x}_2\}$
apple
$L_{13} = \{\mathbf{x}_3\}$
grape
$L_{21} = \{\mathbf{x}_4, \mathbf{x}_5\}$
best feature = size
$L_{22} = \{\mathbf{x}_6\}$
banana
$L_{31} = \{\mathbf{x}_7\}$
apple
$L_{32} = \{\mathbf{x}_8, \mathbf{x}_9\}$
best feature = taste
big
small
sweet
sour
$L_{211} = \{\mathbf{x}_4\}$
grapefruit
$L_{212} = \{\mathbf{x}_5\}$
lemon
$L_{321} = \{\mathbf{x}_8\}$
cherry
$L_{322} = \{\mathbf{x}_9\}$
grape

Fig. 11.8. Example of decision tree induction, using the training set given in Figure 11.7.

11.3.2 Missing Information Recovery by Means of Decision Tree Classifiers: A Basic Scheme

In this section we describe how the network information recovery problem can be cast as a classification problem that can then be solved by means of a decision tree

[5]An example is optical character recognition, where digit 0 and character O may have identical appearance.

[6]The decision tree in Figure 11.8 is actually an example of overfitting.

classifier. We use the same terminology as in Section 11.2 and assume we want to make a decision as to $\gamma_t(l) = 0$ or $\gamma_t(l) = 1$, given $\gamma_t(l) = ?$. Actually, this decision problem can be transformed into a classification problem as follows. The network at time t, g_t corresponds to the unknown object to be classified. Network g_t is described by means of a feature vector $\boldsymbol{x} = (x_1, \ldots, x_d)$, and the decision as to $\gamma_t(l) = 0$ or $\gamma_t(l) = 1$ can be interpreted as a two-class classification problem, where $\gamma_t(l) = 0$ corresponds to class Ω_0 and $\gamma_t(l) = 1$ corresponds to class Ω_1. As features $x_1, \ldots, x_d$ that represent the unknown object $\boldsymbol{x}$, i.e., graph g_t, one can use, in principle, any quantity that is extractable from graphs $g_1, \ldots, g_t$. In the present section we consider the case that these features are extracted from graph g_t exclusively. Assume that the universal set of node labels is given by $\mathcal{L} = \{l_0, l_1, \ldots, l_D\}$, and assume furthermore that it is node label l_0 for which we want to make a decision as to $\gamma_t(l_0) = 0$ or $\gamma_t(l_0) = 1$, given $\gamma_t(l_0) = ?$. Then we set $d = D$ and use the D-dimensional binary feature vector $(\gamma_t(l_1), \ldots, \gamma_t(l_D))$ to represent graph g_t. In other words, $\boldsymbol{x} = (\gamma_t(l_1), \ldots, \gamma_t(l_D))$. This feature vector is to be classified as either belonging to class Ω_0 or Ω_1. The former case corresponds to deciding $\gamma_t(l_0) = 0$, and the latter to $\gamma_t(l_0) = 1$. Intuitively, using $(\gamma_t(l_1), \ldots, \gamma_t(l_D))$ as a feature vector for the classification of g_t means we make a decision as to the presence or absence of l_0 in g_t depending on the presence or absence of all other nodes from $\mathcal{L}$ in g_t.

For the implementation of the classification procedure described in the last paragraph, we need a training set. For the training set we can use all previous graphs in the given time series, i.e., $g_1, \ldots, g_{t-1}$. From each graph g_i, we extract the D-dimensional feature vector

$$\boldsymbol{x_i} = (\gamma_i(l_1), \ldots, \gamma_i(l_D)) \ . \tag{11.13}$$

So our training set becomes $L = \{\boldsymbol{x_1}, \ldots, \boldsymbol{x_{t-1}}\}$. As pointed out in Section 11.3.1 we do need to assign the proper class to each element of the training set. This can be easily accomplished by assigning class Ω_0 to $\boldsymbol{x_i}$ if $\gamma_i(l_0) = 0$; otherwise, if $\gamma_i(l_0) = 1$ we assign class Ω_1 to $\boldsymbol{x}_i$.

Given such a training set constructed from $g_1, \ldots, g_{t-1}$, we can now apply the procedure described in Figure 11.6 to infer a decision tree from training set L. Once the decision tree has been produced, it is easy to classify feature vector $\boldsymbol{x_t}$ (see equation (11.13)), which describes g_t, as belonging to Ω_0 or Ω_1.

As mentioned in Section 11.3.1, decision tree classifiers are able to deal with unknown attribute values. This is important in our application because we must expect that not only information about node l_0 in g_t is missing, but also about other nodes l_i in g_t, where $i \in \{1, \ldots, D\}$. Similarly, in building the decision tree from training set $L = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{t-1}\}$, there may be graphs g_i, $i \in \{1, \ldots, t-1\}$, for which it is not known for some nodes whether they are present in g_i. Hence some of the $\gamma_i\left(l_j\right)$ may be unknown. Using a state-of-the-art decision tree software package will allow us to deal with missing feature values without the necessity of taking any additional precautions. In other words, it doesn't matter, neither during decision tree inference nor while classifying an unknown input object, whether there are unknown feature values or not. The system will be able to correctly handle any case.

The procedure described in this section is based on two assumptions. The first assumption is that there is some kind of correlation between the occurrence of a node l in graph g_t and the occurrence of some (or all) other nodes in the same graph. In other words, we assume that the behavior of node l is dependent, in some way, on the behavior of the other nodes. Note, however, that we don't need to make any assumptions as to the mathematical nature of this dependency. Our second assumption is that there is some stationarity in the dependency between l and the other nodes. Using graphs $g_1, \dots, g_{t-1}$ as a training set to derive a classifier that makes a decision pertaining to graph g_t will work well only if the dependency between l and the other nodes in g_t is of the same nature as in $g_1, \dots, g_{t-1}$.

In a practical setting it may be computationally too demanding to infer a decision tree at each point of time t from $g_1, \dots, g_{t-1}$, because decision tree induction procedures typically work in batch mode. That is, as time progresses from t to $t+1$, and a new decision tree for g_{t+1} is built from $g_1, \dots, g_t$, we can't use the tree produced for g_t before, but need to generate the decision tree for g_{t+1} completely from scratch. Hence it may be preferable to do an update of the actual decision tree only after a certain period of time has elapsed. In the decision tree updating process it may also be advisable to use only part of the network history. This means that for the construction of the decision tree for g_t, we don't use $g_1, \dots, g_{t-1}$, but focus on only the M most recent graphs $g_{t-M}, \dots, g_{t-1}$. This is particularly advisable if there is evidence that the behavior of the network is not perfectly stationary, but changing over time.

11.3.3 Possible Extensions of the Basic Scheme

In Section 11.3.2 we have presented a basic scheme of applying a decision tree classifier to the recovery of missing information in a computer network. In the current section we discuss a number of possible extensions. All extensions are based on the decision tree induction and traversal procedures described in Section 11.3.1. They differ only in the feature vector $\boldsymbol{x} = (x_1, \dots, x_d)$ used to represent the underlying network.

The first possible extension discussed in this section concerns network edges. It is easy to see that information about network edges can be integrated in a feature vector in a straightforward way. If $E = \{e_1, \dots, e_{D'}\} = \mathcal{L} \times \mathcal{L}$ is the set of all potential edges in the network then we can extend equation (11.13) as follows:

$$\boldsymbol{x}_i = \left(\gamma_i(l_1), \dots, \gamma_i(l_D), \delta_i(e_1), \dots, \delta_i\left(e_{D'}\right)\right) . \qquad (11.14)$$

Such an extension would allow us to use not only node information, but also information about the presence or absence of edges in the process of recovering information about node l_0 in graph g_t. All other steps remain the same as described in Section 11.3.2. However, a note of caution regarding computational complexity is in order here, because such an extension will increase the dimensionality of the feature vector from $\mathcal{O}(D)$ to $\mathcal{O}\left(D^2\right)$, which leads to a corresponding increase of complexity in decision tree inference.

Our next extension concerns the recovery of missing edge data. That is, we consider the problem of making a decision as to $\delta_t(e) = 0$ or $\delta_t(e) = 1$, given $\delta_t(e) = ?$ for some

$e \in \mathcal{L} \times \mathcal{L}$. To address this problem we can proceed similarly to Section 11.3.2. The only difference is the way we assign class Ω_0 or Ω_1 to the feature vector $\boldsymbol{x}_i$ representing graph g_i in the training set. Here we use the presence or absence of edge e in graph g_i as the criterion for assigning $\boldsymbol{x}_i$ to Ω_0 or Ω_1, i.e., if $\delta_i(e) = 0$ then we assign graph g_i to Ω_0; otherwise, if $\delta_i(e) = 1$ we assign $\boldsymbol{x}_i$ to Ω_1. Note that this schema can be applied using either only node information in the feature vector (see equation (11.13)), or both node and edge information (see equation (11.14)). A third possibility is to use only edge information. In this case the feature vector becomes $\boldsymbol{x}_i = \left(\delta_i(e_1), \ldots, \delta_i\left(e_{D'}\right)\right)$.

Obviously, in our decision procedure we are not confined to including only information about graph g_i in feature vector $\boldsymbol{x}_i$. In order to classify graph g_t as belonging to either Ω_0 or Ω_1, which corresponds to deciding $\gamma_t(l_0) = 0$ or $\gamma_t(l_0) = 1$, respectively, we can include information about previous graphs in the feature vector as well. For example, by including information about graph g_i and g_{i-1} in $\boldsymbol{x}_i$, equation (11.13) turns into

$$\boldsymbol{x}_i = \left(\gamma_i(l_1), \ldots, \gamma_i(l_D), \gamma_{i-1}(l_1), \ldots, \gamma_{i-1}(l_D)\right) . \tag{11.15}$$

Note that the order of the features in equation (11.15) is arbitrary. However it has to be the same for all graphs g_i. A feature vector as defined by equation (11.15) can be extended by including information about graphs $g_{i-2}, g_{i-3}, \ldots$. The scenario considered in Section 11.2 is obtained if we choose $\boldsymbol{x}_i = \left(\gamma_{i-1}(l), \gamma_{i-2}(l), \ldots, \gamma_{i-d}(l)\right)$. Furthermore, one can include information about the edges in all these graphs.

As a general rule, however, feature vectors should be chosen in a way such that they include only information that is useful to enhance the discriminatory power of the resulting decision tree. If too much information is included in a feature vector, it will not only slow down the classification and decision tree induction processes, but may result in overfitting.[7] To find appropriate features for a given application, careful experimental evaluation is needed.

11.4 Conclusions

In this chapter the problem of missing information recovery has been investigated. In Section 11.2, three heuristic schemes were proposed that all use context in time, i.e., the behavior of a node or an edge in previous graphs in the sequence under observation, in order to predict its presence or absence in the actual graph. Next, in Section 11.3, we have developed a machine-learning-based method to solve the same problem. This method can utilize context in time as well as intragraph context, which means that not only the history of a node or an edge can be used to infer information about its possible presence or absence in the actual graph, but also information about the presence or absence of certain other nodes or edges in the graph under consideration.

The information recovery schemes introduced in Sections 11.2 and 11.3 can be extended in various ways. First of all, we have not addressed the problem of edge label recovery. That is, it may be known that edge e exists in graph g_t, but its label $\beta_t(e)$ may be unknown. Here we can imagine the development of some kind of extrapolation

[7] This phenomenon is also known as the "curse of dimensionality" [67, 134].

schema that computes $\beta_t(e)$ as a linear or nonlinear combination of L previous label values $\beta_{t-L}(e), \ldots, \beta_{t-1}(e)$.

Throughout Sections 11.2 and 11.3 we have always considered the problem of recovering missing information in graph g_t based on information extracted from $g_1, \ldots, g_{t-1}$ (Section 11.2), or $g_1, \ldots, g_t$ (Section 11.3). This problem is an instance of an *online* problem, where we seek a decision immediately after observation g_t has been made. In an *offline* scenario, by contrast, we may be allowed to use all graphs $g_1, \ldots, g_N$ in the whole time series for the recovery of the missing information in g_t, where $t < N$. It is easy to see that the framework proposed in Sections 11.2 and 11.3 can be extended to the offline case in a straightforward way. If not only past graphs $g_1, \ldots, g_{t-1}$ or $g_1, \ldots, g_t$ but also future graphs $g_{t+1}, \ldots, g_N$ are available, information extracted from the future graphs can be used in the procedures introduced in Section 11.2 and 11.3 in the same way as information extracted from the past graphs.

We furthermore note that the online algorithms introduced in Sections 11.2 and 11.3 can be used not only for the purpose of recovering missing information, but also for *predicting* the presence or absence of nodes in a graph. All algorithms of Section 11.2 can be used in their original form for this purpose, while in Section 11.3 we only need to ensure our decision tree being built from data occurring in $g_1, \ldots, g_{t-1}$, but not in g_t, when making a prediction about g_t.

In Section 11.3 we have cast our missing information recovery problem as a classification problem and used a decision tree classifier to solve it. We would like to mention that a variety of other classification methods exist [67, 148]. For those classification methods, however, in particular for statistical classifiers, it is to be expected that the number of features (i.e., the number of nodes and edges in graphs $g_1, \ldots, g_t$) becomes too large, which may lead to poor classifier performance due to overfitting. However, to overcome the problem of too many features, methods for feature selection may be applicable; see Chapter 8 in [186], for example. So statistical classifiers could be trained after suitable reduction of the number of relevant features. Such a procedure may eventually make a large number of classification procedures available for solving our problem of missing information recovery.

12

Matching Hierarchical Graphs

12.1 Introduction

In general, the computation of graph similarity is a very costly task. In the context of this book, however, we focus on a special class of graphs that allow for low-order polynomial-time matching algorithms. The considered class of graphs is characterized by the constraint that each node has a unique node label. This constraint is met in all computer network monitoring and abnormal event detection applications considered in this book.

Future applications of graph matching may require one to deal with graphs consisting of tens or even hundreds of thousands of nodes. For these applications low-order polynomial matching algorithms, such as those considered in previous chapters, may be still too slow. In this chapter we introduce a hierarchical graph representation scheme that is suitable for reducing the size of the graphs under consideration. Other reduction schemes have been proposed in [109], for example. There are also some conceptual similarities with hierarchical quadtree, or pyramid, representations in image processing [4]. The basic idea underlying the proposed hierarchical representation scheme is to contract some nodes of the given graph and represent them as a single node at a higher level of abstraction. There are no particular assumptions about the criteria that control the selection of nodes to be contracted into a single node at a higher abstraction level. For the contraction process, any algorithm that clusters nodes of a graph, including heuristic selection strategies or the algorithms discussed in Chapter 7, may be chosen. Properties of the nodes that are contracted are stored as attributes with the corresponding node at the higher level of abstraction. This process can be carried out in a hierarchical, iterative fashion, which will allow us to eventually contract any arbitrarily large set of nodes into a single node.

Because of the reduced number of nodes, computing the similarity of two graphs at a higher level of abstraction can be expected to be much faster than the corresponding computation on the original graphs. It is, however, desirable that the graph contraction procedure, as well as the chosen graph distance measure, have some monotonicity properties. That is, if graph g_1 is more similar to g_2 than to g_3 at the original, full graph resolution level, then this property should be maintained for the representation at any higher level of abstraction. In this chapter we study several of these properties. While the general monotonicity property, as stated above, can't be guaranteed, we will derive upper and lower bounds of the graph similarity measure at higher levels of abstraction. It will be shown that under certain conditions these bounds are tight, i.e., they are identical to the real similarity value.

In the next section, the proposed graph abstraction scheme is presented. Then in Section 12.3, our new graph similarity measures will be defined and upper and lower bounds for graph distance at higher levels of abstraction derived. Next, potential applications of the proposed graph contraction scheme and the similarity measures in the domain of computer network monitoring will be discussed. In Section 12.5 the results of an experimental study will be presented. Finally, a summary and conclusions will be provided in Section 12.6.

12.2 Hierarchical Graph Abstraction

In this chapter we consider graphs $g = (V, E, \alpha, \beta)$ with unique node labels, and use the following graph edit distance:

$$d(g_i, g_j) = |V_i| + |V_j| - 2|V_i \cap V_j| + |E_i| + |E_j| - 2|E_i \cap E_j|\,. \tag{12.1}$$

This edit distance is identical to the edit distance introduced in Chapter 4 for the case that we neglect edge weight and are just interested in whether an edge is present between a given pair of nodes.

We start our graph abstraction process by partitioning the set of nodes V into a set of subsets, or clusters, $C = \{c_1, \ldots, c_n\}$, where $c_i \subseteq V$, $c_i \cap c_j = \emptyset$, $\bigcup_{i=1}^{n} c_i = V$ for $i \neq j$; $i, j = 1, \ldots, n$.

Definition 12.1. Given a graph g and an arbitrary partitioning C, a *hierarchical abstraction* of g is the graph $\bar{g} = (\bar{V}, \bar{E}, \bar{\alpha}, \bar{\beta})$ where:

(i) $\bar{V} = C$, i.e., each node in $\bar{g}$ represents a cluster of nodes in g (hence $\bar{V} = \{c_1, \ldots, c_n\}$);
(ii) $\bar{E} = \bar{V} \times \bar{V}$, i.e., $\bar{g}$ is fully connected;
(iii) $\bar{\alpha}(v) = (nodes(v), edges(v))$ for each $v \in \bar{V}$, such that
 – $nodes(v) = |c|$, where v represents c
 – $edges(v) = |\{e \mid e = (x, y) \in E \wedge x \in c \wedge y \in c\}|$, where v represents c.
 That is, each node in $\bar{g}$ gets two attributes, $nodes(v)$ and $edges(v)$, assigned to it. The attribute $nodes(v)$ is equal to the number of nodes in graph g that belong to the cluster represented through v, while $edges(v)$ is equal to the number of edges in that cluster in graph g; and
(iv) $\bar{\beta}(e) = |\{(x, y) \mid (x, y) \in E \wedge x \in c_i \wedge y \in c_j \wedge e = (c_i, c_j)\}|$ for each $e \in \bar{E}$. That is, if e is an edge in $\bar{g}$ originating at the node representing cluster c_i and terminating at the node representing cluster c_j, then we count the number of edges in g that lead from a node in c_i to a node on c_j.

Example 12.2. A graph g_i and its hierarchical abstraction $\bar{g}_i$ are shown in Figure 12.1. For these graphs we observe that $V_i = \{1, 2, 3, 4, 5\}$ and $E_i = \{(1, 2), (1, 4), (2, 1), (2, 4), (2, 5), (3, 1), (4, 3)\}$.

We assume that V_i is partitioned into $C = \{\{1, 2\}, \{3, 4\}, \{5\}\}$, i.e., $c_1 = \{1, 2\}$, $c_2 = \{3, 4\}$, $c_3 = \{5\}$. The hierarchical abstraction $\bar{g}_i = (\bar{V}, \bar{E}, \bar{\alpha}, \bar{\beta})$ is then given by

$$\begin{aligned}
&\bar{V}_i = \{\{1, 2\}, \{3, 4\}, \{5\}\} = \{c_1, c_2, c_3\},\\
&\bar{E}_i = \{(c_1, c_2), (c_1, c_3), (c_2, c_1)\},\\
&\text{nodes}_i : c_1 \to 2, c_2 \to 2, c_3 \to 1,\\
&\text{edges}_i : c_1 \to 2, c_2 \to 1, c_3 \to 0,\\
&\bar{\beta}_i : (c_1, c_2) \to 2, (c_1, c_3) \to 1, (c_2, c_1) \to 1.
\end{aligned}$$

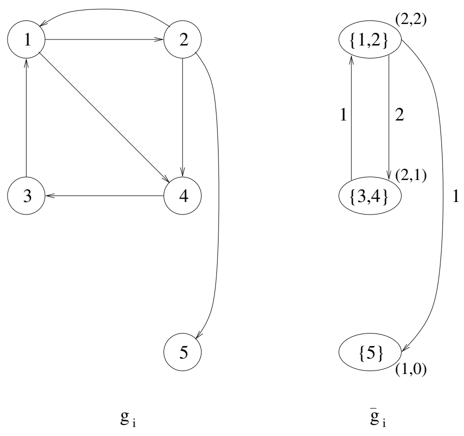


Fig. 12.1. A graph g_i and its hierarchical abstraction $\bar{g}_i$.

All edges e that have an attribute value $\beta(e) = 0$ are not included in Figure 12.1. In the graphical representation of $\bar{g}_i$ in Figure 12.1, the pairs (x, y) displayed next to the nodes correspond to the node attributes, i.e., $x = \text{nodes}(v)$, $y = \text{edges}(v)$. Similarly, the numbers next to the edges correspond to the edge attributes.

12.3 Distance Measures for Hierarchical Graph Abstraction

In this section we introduce two distance measures for the hierarchical graph abstraction introduced in Section 12.2, and discuss relationships with the measure defined in equation (12.1). Throughout this section we assume that $g_i = (V_i, E_i, \alpha_i, \beta_i)$ and $g_j = (V_j, E_j, \alpha_j, \beta_j)$ are two given graphs. The nodes and edges of both graphs come from (possibly larger) sets V and E, respectively, i.e., $V_i \cup V_j \subseteq V$, $E_i \cup E_j \subseteq E$, and $C = \{c_1, \ldots, c_n\}$ is a partition of V. The graphs $\bar{g}_i = (\bar{V}_i, \bar{E}_i, \bar{\alpha}_i, \bar{\beta}_i)$ and $\bar{g}_j = (\bar{V}_j, \bar{E}_j, \bar{\alpha}_j, \bar{\beta}_j)$ are the hierarchical abstractions of g_i and g_j, respectively, both based on the partition C. In Section 12.4, we will consider not only pairs, but whole sets of graphs $G = \{g_1, \ldots, g_m\}$, and compute the distance of various pairs of graphs from set G. For reasons of efficiency, it is advantageous to consider one global partitioning C for all graphs from G. Otherwise, if individual partitionings are applied, not all pairs $\bar{g}_i$ and $\bar{g}_j$ will be comparable under the considered distance measures.

The first distance measure is defined as follows:

$$D_l(\bar{g}_i, \bar{g}_j) = \sum_{v \in \bar{V}} |\text{nodes}_i(v) - \text{nodes}_j(v)| + \sum_{v \in \bar{V}} |\text{edges}_i(v) - \text{edges}_j(v)| + \sum_{e \in \bar{E}} |\bar{\beta}_i(e) - \bar{\beta}_j(e)| \,. \tag{12.2}$$

Example 12.3. A graph g_j and its hierarchical abstraction $\bar{g}_j$ are shown in Figure 12.2. We assume that $V = V_i \cup V_j$, $E = E_i \cup E_j$ and $C = \{\{1, 2\}, \{3, 4\}, \{5, 6\}\}$. It is easy to verify that $d(g_i, g_j) = 13$ and $D_l(\bar{g}_i, \bar{g}_j) = 9$. The distance $D_l(\bar{g}_i, \bar{g}_j)$ is obtained by summing the absolute differences of all pairs of corresponding attribute values. For *nodes* we get the value two, for *edges* the value three, and for $\bar{\beta}(e)$ the value four.

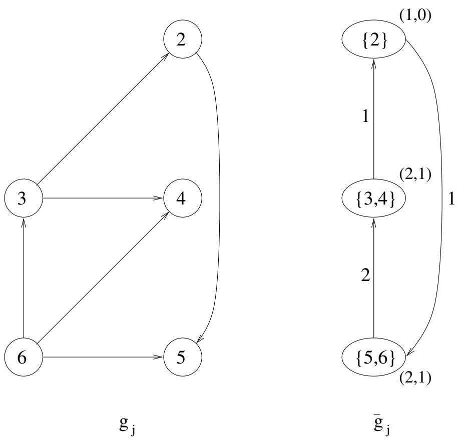


Fig. 12.2. Another graph g_j, and its hierarchical abstraction $\bar{g}_j$.

The fact that $D_l(\bar{g}_i, \bar{g}_j) \leq d(g_i, g_j)$ is not a coincidence. It can be easily proven that $D_l(\bar{g}_i, \bar{g}_j)$ is a lower bound of $d(g_i, g_j)$ for any pair of graphs g_i and g_j, and any partitioning C.

Lemma 12.4. *Let g_i, g_j, $\bar{g}_i$ and $\bar{g}_j$ be graphs as introduced above. Then*

$$D_l(\bar{g}_i, \bar{g}_j) \leq d(g_i, g_j).$$

Proof. The proof is based on the observation that the term $|V_i| + |V_j| - 2|V_i \cap V_j|$ in equation (12.1) is equal to the number of nodes that are in either g_i or g_j, but not in both. Similarly, $|E_i| + |E_j| - 2|E_i \cap E_j|$ is equal to the number of edges either in g_i or g_j, but not in both. In equation (12.2), node v (corresponding to one of the clusters c_k) includes exactly $nodes_i(v)$ nodes from g_i and $nodes_j(v)$ nodes from g_j. Hence there must be at least $|nodes_i(v) - nodes_j(v)|$ nodes that are not in both g_i and g_j. Summing up over all nodes $v \in \bar{V}$ (i.e., clusters $c_k \in C$) yields a lower bound of the expression $|V_i|+|V_j|-2|V_i \cap V_j|$. Similarly, the sum of the second and the third terms in equation (12.2) yields a lower bound of $|E_i| + |E_j| - 2|E_i \cap E_j|$.

It can be shown that under certain conditions the lower bound given by equation (12.2) is exact.

Lemma 12.5. *Let g_i, g_j, $\bar{g}_i$, and $\bar{g}_j$ be the same as in Lemma 12.4. Furthermore, let $V_i \subseteq V_j$ and $E_i \subseteq E_j$. Then*

$$D_l(\bar{g}_i, \bar{g}_j) = d(g_i, g_j).$$

Proof. From our assumptions it follows that $|V_i \cap V_j| = |V_i|$ and $|E_i \cap E_j| = |E_i|$. Hence $|V_i|+|V_j|-2|V_i \cap V_j| = |V_j|-|V_i|$, $|E_i|+|E_j|-2|E_i \cap E_j| = |E_j|-|E_i|$, and $d(g_i, g_j) = |V_j| - |V_i| + |E_j| - |E_i|$. Obviously, the right-hand side of this equation is identical to the right-hand side of equation (12.2) under the assumption $|V_i| \subseteq |V_j|$ and $|E_i| \subseteq |E_j|$.

The second graph distance measure is defined as follows:

$$D_u(\bar{g}_i, \bar{g}_j) = \sum_{v \in \bar{V}} \text{NODES}(v) + \sum_{v \in \bar{V}} \text{INTRACLUSTER-EDGES}(v) + \sum_{e \in \bar{E}} \text{INTERCLUSTER-EDGES}(e)\,, \tag{12.3}$$

where

$$\text{NODES}(v) = \begin{cases} nodes_i(v) + nodes_j(v), & \text{if } nodes_i(v) + nodes_j(v) < |c|, \\ 2|c| - nodes_i(v) - nodes_j(v), & \text{otherwise}\,, \end{cases}$$

$$\text{INTRACLUSTER-EDGES}(v) = \begin{cases} edges_i(v) + edges_j(v), & \text{if } edges_i(v) + edges_j(v) < |\text{EDGES}(v)|, \\ 2|\text{EDGES}(v)|\text{-}edges_i(v) - edges_j(v), & \text{otherwise}\,, \end{cases}$$

and

$$\text{INTERCLUSTER-EDGES}(e) = \begin{cases} \bar{\beta}_i(e) + \bar{\beta}_j(e), & \text{if } \bar{\beta}_i(e) + \bar{\beta}_j(e) < |\text{EDGES}(e)|, \\ 2|\text{EDGES}(e)| - \bar{\beta}_i(e) - \bar{\beta}_j(e), & \text{otherwise}\,. \end{cases}$$

In this definition, c denotes the cluster that corresponds to node v, EDGES(v) is the set of all edges in set E that belong to cluster c, and EDGES(e) is the set of all edges in E that start and end at the same cluster as edge e. Formally,

$$\text{EDGES}(v) = \{e \mid e = (x, y) \in E \wedge x \in c \wedge y \in c\}\,,$$

and

$$\text{EDGES}(e) = \{(x, y) \mid (x, y) \in E \wedge x \in c_i \wedge y \in c_j \wedge e = (c_i, c_j)\}\,.$$

Example 12.6. For the graphs shown in Figures 12.1 and 12.2, we obtain $D_u(\bar{g}_i, \bar{g}_j) = 13$. Note that in all of the quantities NODES(v), INTRACLUSTER-EDGES(v), and INTERCLUSTER-EDGES(e) the second condition always evaluates to true. The first term in equation (12.3) evaluates to two, while values five and six are obtained for the second and third terms, respectively.

Next we show that the measure $D_u(\bar{g}_i, \bar{g}_j)$ is an upper bound on $d(g_i, g_j)$.

Lemma 12.7. *Let g_i, g_j, $\bar{g}_i$ and $\bar{g}_j$ be graphs as introduced above. Then*

$$d(g_i, g_j) \leq D_u(\bar{g}_i, \bar{g}_j)\,.$$

Proof. If the number of nodes of V that belong to cluster c is greater than the number of nodes of g_i in cluster c plus the number of nodes of g_j in cluster c, then the intersection of nodes of g_i and g_j is possibly empty and the expression $|V_i| + |V_j| - 2|V_i \cap V_j|$ in equation (12.1) is bounded from above by $nodes_i(c) + nodes_j(c)$. Otherwise, some nodes from g_i and g_j must be the same, i.e., some nodes must occur in both g_i and g_j. The number of these nodes is equal to $nodes_i(c)+nodes_j(c)-|c|$. Hence the expression $|V_i| + |V_j| - 2|V_i \cap V_j|$ becomes equal to $|nodes_i(c) + nodes_j(c) - 2(nodes_i(c) + nodes_j(c) - |c|)| = 2|c| - nodes_i(c) - nodes_j(c)$. A similar argument holds for the edges, i.e., for the the attributes $edges_i(c)$, $edges_j(c)$, $\bar{\beta}_i(e)$, and $\bar{\beta}_j(e)$. Summing over all clusters c and all edges in E provides an upper bound of $d(g_i, g_j)$.

In Example 12.6 we note that $D_u(\bar{g}_i, \bar{g}_j) = d(g_i, g_j)$. This is no coincidence because the proof of Lemma 12.7 implies that the upper bound $D_u(\bar{g}_i, \bar{g}_j)$ is equal to the actual distance $d(g_i, g_j)$ if $|V_i| + |V_j| \geq |V|$ and $|E_i| + |E_j| \geq |E|$. This is summarized in the following lemma.

Lemma 12.8. *Let g_i, g_j $\bar{g}_i$, and $\bar{g}_j$ be defined as in Lemma 12.7 and let $|V_i|+|V_j| \geq |V|$ and $|E_i| + |E_j| \geq |E|$. Then*

$$D_u(\bar{g}_i, \bar{g}_j) = d(g_i, g_j)\,.$$

A consequence of this lemma is that for any two graphs g_i, g_j and their hierarchical abstractions $\bar{g}_i$, $\bar{g}_j$, the quantity $D_u(\bar{g}_i, \bar{g}_j)$ is always equal to $d(g_i, g_j)$ if we set $V = V_i \cup V_j$ and $E = E_i \cup E_j$.

In the remainder of this section we will investigate the problem of how the upper and lower bounds $D_u(\bar{g}_i, \bar{g}_j)$ and $D_l(\bar{g}_i, \bar{g}_j)$ depend on the way we partition the set V. Let $C = \{c_1, \ldots, c_n\}$ and $\bar{C} = \{\bar{c}_1, \ldots, \bar{c}_m\}$ be two different partitionings of set V. We call C *finer* than $\bar{C}$ if for each c_i there exists a $\bar{c}_j$ such that $c_i \subseteq \bar{c}_j$. Let g_i and g_j be two graphs, $\bar{g}_i$ and $\bar{g}_j$ their hierarchical abstractions based on partition C, and $\bar{G}_i$ and $\bar{G}_j$ their hierarchical abstractions based on partition $\bar{C}$, where C is finer than $\bar{C}$. Then we can prove that $D_u(\bar{g}_i, \bar{g}_j)$ and $D_l(\bar{g}_i, \bar{g}_j)$ are better approximations of $d(g_i, g_j)$ than $D_u(\bar{G}_i, \bar{G}_j)$ and $D_l(\bar{G}_i, \bar{G}_j)$, respectively.

Lemma 12.9. *Let $\bar{g}_i$, $\bar{g}_j$, $\bar{G}_i$, and $\bar{G}_j$ be as defined above. Then*

$$D_l(\bar{G}_i, \bar{G}_j) \leq D_l(\bar{g}_i, \bar{g}_j) .$$

Proof. Assume that the cluster $c \in \bar{C}$ is split into clusters $c_1, \ldots, c_k \in C$ when we refine the partition $\bar{C}$ to the partition C. Clearly, $|c| = \sum_{l=1}^{k} |c_l|$. The contribution of cluster c to the first term of $D_l(\bar{G}_i, \bar{G}_j)$ is equal to $|nodes_i(c) - nodes_j(c)|$, which can be rewritten as $|\sum_{l=1}^{k} nodes_i(c_l) - \sum_{l=1}^{k} nodes_j(c_l)|$; see equation (12.2). On the other hand, for clusters $c_1, \ldots, c_k$ we get a contribution equal to $\sum_{l=1}^{k} |nodes_i(c_l) - nodes_j(c_l)|$ to the first term in $D_l(\bar{g}_i, \bar{g}_j)$. Applying a similar argument to the second and third terms in equation (12.2) and using the well-known relation $|\sum_{l=1}^{k} a_l - \sum_{l=1}^{k} b_l| \leq \sum_{l=1}^{k} |a_l - b_l|$, which holds for any set of real numbers a_l, b_l, concludes the proof.

Lemma 12.10. *Let $\bar{g}_i$, $\bar{g}_j$, $\bar{G}_i$, and $\bar{G}_j$ be the same as in Lemma 12.9. Then*

$$D_u(\bar{g}_i, \bar{g}_j) \leq D_u(\bar{G}_i, \bar{G}_j) .$$

Proof. The proof is based on observing that in the computation of NODES(v) in equation (12.3), whenever the second case evaluates to true for a partition $\bar{C}$, it will also evaluate to true for any other partition C that is finer than $\bar{C}$. On the other hand, if the first case evaluates to true for $\bar{C}$, then either the first or the second case may evaluate to true for any of the clusters in C. Moreover, we observe that the value under the second condition is always less than or equal to the value obtained under the first condition. Applying a similar argument to INTRACLUSTER-EDGES(v) and INTERCLUSTER-EDGES(e) yields the proof.

Summarizing all results derived in this section, we obtain the following theorem:

Theorem 12.11. *Let all quantities be as introduced above. Then:*

$$\begin{aligned}
&(i) \quad D_l(\bar{G}_i, \bar{G}_j) \leq D_l(\bar{g}_i, \bar{g}_j) \leq d(g_i, g_j) \leq D_u(\bar{g}_i, \bar{g}_j) \leq D_u(\bar{G}_i, \bar{G}_j) , \\
&(ii) \quad d(g_i, g_j) = D_l(\bar{g}_i, \bar{g}_j) = D_l(\bar{G}_i, \bar{G}_j), \quad \text{if } V_i \subseteq V_j \text{ and } E_i \subseteq E_j , \\
&(iii) \quad d(g_i, g_j) = D_u(\bar{g}_i, \bar{g}_j) = D_u(\bar{G}_i, \bar{G}_j), \quad \text{if } V = V_i \cup V_j, E = E_i \cup E_j .
\end{aligned}$$

We notice that whenever the condition in (ii) is satisfied, the condition in (iii) will also be satisfied. Hence in this case $D_l(\bar{g}_i, \bar{g}_j) = D_l(\bar{G}_i, \bar{G}_j) = d(g_i, g_j) = D_u(\bar{g}_i, \bar{g}_j) = D_u(\bar{G}_i, \bar{G}_j)$.

12.4 Application to Computer Network Monitoring

Through the hierarchical abstraction process described in Section 12.2, the number of nodes in a graph can be reduced. In fact, it can be made arbitrarily small. In the extreme case, a large graph will be represented by a single node only. Applying equations (12.2) and (12.3) to a pair of graphs $\bar{g}_i, \bar{g}_j$, which result from g_i and g_j through the proposed abstraction process, yields upper and lower bounds for $d(g_i, g_j)$. The closer $\bar{g}_i$ and $\bar{g}_j$ are to the original, full-resolution graphs g_i and g_j, i.e., the more details are included in the abstract graph representation, the closer will be the upper and lower bounds to the actual value $d(g_i, g_j)$. Note that only two numbers, the number of corresponding nodes and the number of corresponding edges from the original graph, need to be stored with a node at an abstract level. For edges at an abstract level, only one number is needed, representing the number of corresponding edges in the original graph.

If an abnormal event at time $t + 1$ is defined by the condition $d(g_t, g_{t+1}) \geq \theta$, where θ is a threshold that depends on the considered application and the underlying network, then rather than considering $d(g_t, g_{t+1})$ one can compute $D_l(\bar{g}_t, \bar{g}_{t+1})$ and $D_u(\bar{g}_t, \bar{g}_{t+1})$, where $\bar{g}_t$ and $\bar{g}_{t+1}$ are obtained from g_t and g_{t+1} through the proposed graph abstraction procedure. Clearly, if $D_l(\bar{g}_t, \bar{g}_{t+1}) \geq \theta$ then we conclude that an abnormal event has occurred. Similarly, if $D_u(\bar{g}_t, \bar{g}_{t+1}) < \theta$ then we conclude that no abnormal event has occurred. In either case we need not compute $d(g_t, g_{t+1})$, and it can be expected that computing $D_l(\bar{g}_t, \bar{g}_{t+1})$ and $D_u(\bar{g}_t, \bar{g}_{t+1})$ is faster than the computation of $d(g_t, g_{t+1})$. On the other hand, if $D_l(\bar{g}_t, \bar{g}_{t+1}) < \theta$ and $D_u(\bar{g}_t, \bar{g}_{t+1}) \geq \theta$, then we need to calculate $d(g_t, g_{t+1})$. Alternatively we can compute $D_l(\tilde{g}_t, \tilde{g}_{t+1})$ and $D_u(\tilde{g}_t, \tilde{g}_{t+1})$ for graphs $\tilde{g}_t$ and $\tilde{g}_{t+1}$ that are closer to the original level of resolution than $\bar{g}_t$ and $\bar{g}_{t+1}$, expecting either $D_l(\tilde{g}_t, \tilde{g}_{t+1}) \geq \theta$ or $D_u(\tilde{g}_t, \tilde{g}_{t+1}) < \theta$.

In Chapter 7 (intra)graph clustering algorithms have been described. These algorithms are able to identify clusters of nodes within a graph such that nodes in the same cluster are similar to each other, while nodes in different clusters are dissimilar. For the graph abstraction process described in Section 12.2, no such clustering algorithm is needed. In fact, any partition of the underlying set of nodes can be used as the basis of graph abstraction. In the simplest case, one can just assign unique labels from 1 to N to all servers in the underlying network, and then partition the set $\{1, \ldots, N\}$ into a given number of disjoint subsets.

Computing $D_l(\bar{g}_t, \bar{g}_{t+1})$ and $D_u(\bar{g}_t, \bar{g}_{t+1})$, we get lower and upper bounds of $d(g_t, g_{t+1})$, respectively, as discussed before. Note that in the case that all nodes and edges of the entire network appear in the union g_t and g_{t+1}, the upper bound $D_u(\bar{g}_t, \bar{g}_{t+1})$ is the exact value. In this case two arbitrarily large graphs g_t and g_{t+1} can be contracted to a single node each, and the upper bound will still be the exact value, i.e., $D_u(\bar{g}_t, \bar{g}_{t+1}) = d(g_t, g_{t+1})$. As an example consider the graphs in Figures 12.1 and 12.2. If we contract g_i into a single node v, we get $nodes_i(v) = 5$, $edges_i(v) = 7$. Similarly, if g_j is contracted into a single node, then $nodes_j(v) = 5$, $edges_j(v) = 6$. Moreover, under the assumption that the union of g_i and g_j includes the entire set of nodes and edges of the network, we observe that $|c| = 6$ and $|\mathrm{EDGES}(v)| = 12$. Hence $D_u(\bar{g}_i, \bar{g}_j) = (12 - 10) + (24 - 13) = 13 = d(g_i, g_j)$.

12.5 Experimental Results

The aim of the experiments described in this section is to verify the theoretical results derived in Section 12.3, to measure the tightness of upper and lower bounds, and to quantitatively evaluate the computational savings that can be achieved through the proposed graph abstraction scheme. In the experiments described in this section, we first generate a graph g_1, with 10,000 nodes. Each node is connected, on average, to 1,000 other nodes via an undirected, unlabeled edge. Edges are randomly distributed in g_1. The set of integers $\{1, \ldots, 10{,}000\}$ is used to label the nodes and each node has a unique label. A second graph g_2 is obtained from g_1 by randomly deleting $n\%$ of the nodes of g_1, together with their incident edges. Additionally $n'\%$ of the remaining edges are deleted. Next, hierarchical abstractions of both g_1 and g_2 are generated, consisting of 1,000, 100, 10, and 1 node. These hierarchical abstractions are all based on the same partition of the nodes of g_1. For example, to generate a hierarchical abstraction with 1,000 nodes, i.e., with a cluster size of ten nodes each, the first cluster is given by the nodes with a label from $\{1, \ldots, 10\}$, the second cluster by the nodes with a label from $\{11, \ldots, 20\}$, and so on. From the way g_1 and g_2 are generated, it is obvious that the conditions of Lemmas 12.5 and 12.8 are fulfilled. Hence, we expect that $D_l(\bar{g}_1, \bar{g}_2) = d(g_1, g_2) = D_u(\bar{g}_1, \bar{g}_2)$ for any of the considered hierarchical abstractions $\bar{g}_i$. As a matter of fact, this expectation is confirmed in Figure 12.3, where the x-axis corresponds to the different levels of abstraction (i.e., number of clusters, which is 1,000, 100, 10, 1), and the y-axis represents the distances $d(g_1, g_2)$, $D_l(\bar{g}_1, \bar{g}_2)$, $D_u(\bar{g}_1, \bar{g}_2)$. All three distances coincide for any considered level of abstraction, which confirms that both upper and lower bounds are identical to the real graph distance. In Figure 12.3, values $n = 10$ and $n' = 5$ are used. In Figure 12.4, the results of four similar experimental runs are shown for values $(n = 20, n' = 10)$, $(n = 30, n' = 15)$, $(n = 40, n' = 20)$, and $(n = 50, n' = 25)$. In each case the values of $d(g_1, g_2)$, $D_l(\bar{g}_1, \bar{g}_2)$, and $D_u(\bar{g}_1, \bar{g}_2)$ coincide. Hence only four straight lines are observed in this figure. Clearly, with an increasing number of nodes and edges being deleted from g_1, the distance between g_1 and g_2 increases. This effect can be clearly observed in Figure 12.4. The point to be stressed about Figure 12.4 is that, similarly to Figure 12.3, all three measures $D_l(\bar{g}_1, \bar{g}_2)$, $d(g_1, g_2)$, and $D_u(\bar{g}_1, \bar{g}_2)$ are identical, as stated in Lemmas 12.5 and 12.8.

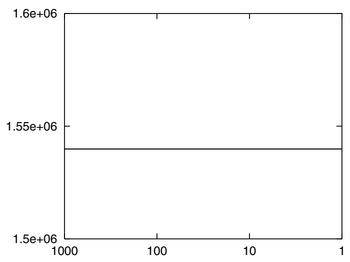


Fig. 12.3. Experimental data illustrating Lemmas 12.5 and 12.8.

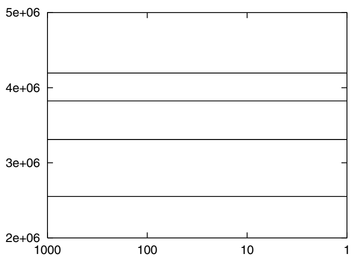


Fig. 12.4. Further illustration of Lemmas 12.5 and 12.8.

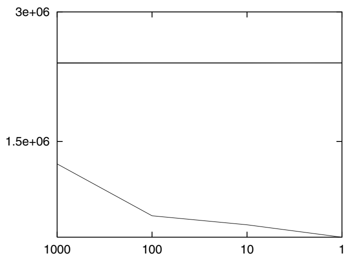


Fig. 12.5. Experimental data illustrating the upper and lower bound ($m = 60$).

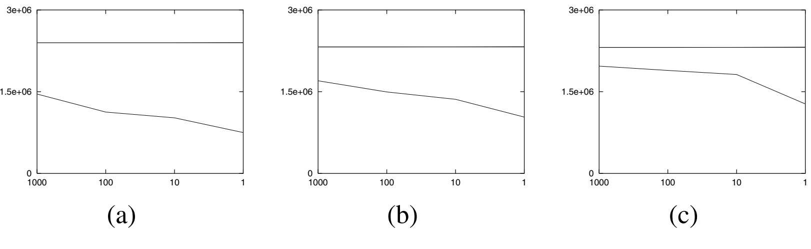


Fig. 12.6. Further illustration of upper and lower bound: (a) $m = 70$, (b) $m = 80$, (c) $m = 90$.

The aim of the next set of experiments is to analyze the behavior of the upper and lower bounds in case the conditions of Lemmas 12.5 and 12.8 are no longer satisfied. For this purpose, we start again with a graph g_1 that is generated in exactly the same way as described in the previous paragraph. Next we randomly delete 50% of the nodes of g_1 together with their incident edges. The resulting graph is referred to as g_3. Next, graph g_4 is generated by randomly deleting m% of all nodes together with their incident edges

Table 12.1. Data corresponding to Figure 12.5

	1000	100	10	1	m
D_u	2,409,516	2,410,042	2,410,088	2,411,390	
D_l	1,239,038	638,308	534,130	390,094	60
d	2,407,580				

Table 12.2. Data corresponding to Figure 12.6.

	1000	100	10	1	m
D_u	2,397,782	2,398,060	2,398,146	2,400,294	
D_l	1,457,062	1,126,264	1,020,270	749,790	70
d	2,396,262				
D_u	2,322,436	2,322,510	2,322,814	2,325,836	
D_l	1,700,350	1,495,504	1,360,116	1,034,514	80
d	2,321,440				
D_u	2,313,272	2,313,318	2,313,494	2,317,408	
D_l	1,969,420	1,890,110	1,815,788	1,275,640	90
d	2,312,800				

from g_1 ($m = 60, 70, 80, 90$). Clearly, when we match graphs g_3 and g_4, the conditions of Lemmas 12.5 and 12.8 are not necessarily satisfied any longer. Similarly to the first set of experiments, hierarchical abstractions of g_3 and g_4 were generated consisting of 1,000, 100, 10, and 1 node. In Figure 12.5, the distances $d(g_3, g_4)$, $D_l(\bar{g}_3, \bar{g}_4)$, and $D_u(\bar{g}_3, \bar{g}_4)$ are shown for $m = 60$. While the lower bound is significantly smaller than the real distance, the upper bound is quite tight. As a matter of fact, $d(g_3, g_4)$ visually coincides with $D_u(\bar{g}_3, \bar{g}_4)$ in Figure 12.5. To see that $d(g_3, g_4)$ is not identical to $D_u(\bar{g}_3, \bar{g}_4)$, the information provided in Figure 12.5 is shown in tabular form in Table 12.1. In Figure 12.6 and Table 12.2 the corresponding values are given for $m = 70, 80$, and 90. As m increases, graphs g_3 and g_4 become more similar to each other. In any case, the upper bound is very close to the real distance even for the maximum degree of compression, where both graphs are represented through a single node only. We also observe that both upper and lower bounds become tighter as the distance $d(g_3, g_4)$ decreases.

The motivation of the third set of experiments is to measure the computational savings that can be achieved by means of the proposed hierarchical graph abstraction scheme. We assume that the sensors, or devices, that yield the graph data not only provide us with the graphs at the full level of resolution, but also with hierarchical abstractions. Hence the time needed to generate hierarchical abstractions from a graph at the full resolution level is not taken into account in the experiments described in the following. To analyze the computational efficiency of the proposed graph similarity measures, we select graphs g_3 and g_4 (with $m = 60$) and their hierarchical abstractions, as described in the last paragraph, and measure the time needed to compute $d(g_3, g_4)$, $D_l(\bar{g}_3, \bar{g}_4)$, and $D_u(\bar{g}_3, \bar{g}_4)$. The results are shown in Table 12.3. The computation of $d(g_3, g_4)$ is performed on the original graphs g_3 and g_4, and is independent of the cluster size in the hierarchical abstraction. It turns out that the computation of both $D_l(\bar{g}_3, \bar{g}_4)$ and $D_u(\bar{g}_3, \bar{g}_4)$ is extremely fast when compared to $d(g_3, g_4)$. From this observation we can conclude that distance measure D_u provides an excellent compromise between speed and precision. On one hand, it is extremely fast to compute, and on the other, it returns values very close to the real graph distance. As a matter of fact, a speedup on the order of 10^8 can be observed over the computation of $d(g_3, g_4)$ for the case of maximum graph compression, while the precision of the upper bound is still within a tolerance of 0.2%.

Table 12.3. Computational time of the distance measures in msec.

	1000	100	10	1	m
Time D_u	61.6	0.5632	0.00568	0.000076	
Time D_l	19.52	0.1552	0.001304	0.0000552	60
Time d			36,000		

12.6 Conclusions

In this chapter we have described a hierarchical graph abstraction procedure that contracts clusters, or groups, of nodes into single nodes. On this hierarchical representation, graph similarity can be computed more efficiently than on the original graphs. Two distance measures for contracted graphs are introduced, and it is shown that they provide lower and upper bounds, respectively, for the distance of graphs at the original level of resolution. The proposed methods can be used to very significantly speed up the computation of graph similarity in the context of computer network monitoring and abnormal change detection. It can be proven that under special conditions, upper and/or lower bounds are exact.

References

1. *CAIDA Cooperative Association for Internet Data Analysis*. Available at **http://www.caida.org/Tools**.
2. S. Aidarous and T. Plevyak. *Telecommunications Network Management into the 21st Century: Techniques, Standards, Technologies, and Applications*. IEEE Press, Piscataway, N.J., 1994.
3. J.T. Astola and T.G. Campbell. On computation of the running median. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(4):572–574, April 1989.
4. D. Ballard and C. Brown. *Computer Vision*. Prentice Hall, 1982.
5. S. Ballew. *Managing IP Networks with Cisco routers*. O'Reilly and Associates, Cambridge, Mass, first edition, October 1997.
6. A.L. Barabási. *Linked: The New Science of Networks*. Perseus Publishing, Cambridge MA, 2002.
7. P. Barford and D. Plonka. Characteristics of network traffic flow anomalies. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pages 69–73, San Francisco, California, USA, 2001. ACM Press.
8. R.A. Becker, S.G. Eick, and A.R. Wilks. Visualizing network data. *IEEE Transactions on Visualization and Computer Graphics*, 1(1):16–21, 1995.
9. Y. Bejerano and R. Rastogi. Robust monitoring of link delays and faults in IP networks. In *Proceedings of the IEEE INFOCOM*, San Francisco, California, March 2003.
10. C. Berge. *Graphs and Hypergraphs*. North-Holland, 1973.
11. A. Bierman and R. Iddon. RFC 2074: Remote network monitoring MIB protocol identifiers, January 1997.
12. N.L. Biggs, E.K. Lloyd, and R.J. Wilson. *Graph Theory, 1736-1936*. Clarendon Press, Oxford, 1976.
13. U. Black. *Network Management Standards: The OSI, SNMP and CMOL Protocols*. McGraw Hill, 1992.
14. P. Bodensiek. *Intranet Publishing*. QUE, 1996.
15. C. Bohoris, G. Pavlou, and H Cruickshank. Using mobile agents for network performance monitoring. In *Proceedings of the IEEE/IFIP Network Operations and Management Symposium*, pages 637–652, April 2000.
16. B. Bollobás. *Random Graphs*. Academic Press, London, 1985.
17. R. Boutaba, K.E. Guemhioui, and P. Dini. An outlook on intranet management. *IEEE Communications Magazine*, pages 92–99, October 1997.

18. Y. Breitbart, C.Y. Chan, M. Garofalakis, R. Rastogi, and A. Silberschatz. Efficiently monitoring bandwidth and latency in IP networks. In *Proceedings of the IEEE INFOCOM*, pages 933–942, Anchorage, Alaska, April 2001.
19. D. Breitgand, D. Raz, and Y. Shavitt. SNMP GetPrev: An efficient way to access data in large MIB tables. *IEEE Journal on Selected Areas in Communications*, 20(4):656–667, 2002.
20. H. Bunke. On a relation between graph edit distance and maximal common subgraph. *Pattern Recognition Letters*, 18:689–694, 1997.
21. H. Bunke. Error correcting graph matching: on the influence of the underlying cost function. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21:917–922, 1999.
22. H. Bunke, P. Dickinson, and M. Kraetzl. Matching sequences of graphs with applications in computer network analysis. In *Proceedings of the 8th World Multiconference on Systemics, Cybernetics and Informatics*, pages 270–275, Orlando, FL, 2004.
23. H. Bunke, P. Dickinson, and M. Kraetzl. Recovery of missing information in graph sequences. In L. Brun and M. Vento, editors, *Proceedings of the 5th International Workshop GBR2005, Springer, LNCS 3434*, Graph Based Representations in Pattern Recognition, pages 312–321, 2005.
24. H. Bunke and S. Günter. Weighted mean of a pair of graphs. *Computing*, 67(3):209–224, November 2001.
25. H. Bunke, M. Kraetzl, P. Shoubridge, and W.D. Wallis. Measuring change in large enterprise data networks. In *Proceedings of the International Conference on Information, Decision and Control*, pages 53–58, Adelaide, 2002.
26. H. Bunke, M. Kraetzl, P.J. Shoubridge, and W.D. Wallis. Measuring abnormal change in large data networks. In *Proceedings of the International Conference on Information, Decision and Control*, pages 53–58, Adelaide, 2002.
27. H. Bunke and B. Messmer. Recent advances in graph matching. *International Journal of Pattern Recognition and Artificial Intelligence*, 11(1):169–203, 1997.
28. H. Bunke, A. Muenger, and X. Jiang. Combinatorial search versus genetic algorithms; A case study based on the generalized mean graph problem. *Pattern Recognition Letters*, 20:1271–1277, 1999.
29. H. Bunke and A. Sanfeliu. *Syntactic and Structural Pattern Recognition – Theory and Applications*. World Scientific, 1990.
30. H. Bunke and K. Shearer. A graph distance metric based on the maximal common subgraph. *Pattern Recognition Letters*, 19:255–259, 1998.
31. J.B.D. Cabrera, L. Lewis, X. Qin, W. Lee, R.K. Prasanth, B. Ravichandran, and R.K. Mehra. Proactive detection of distributed denial of service attacks using mib traffic variables - a feasibility study. In *Proceedings of the IEEE/IFIP International Symposium on Integrated Network Management*, pages 609–622, May 2001.
32. J. Case, M. Fedor, M. Schoffstall, and J. Davin. RFC 1157: A simple network management protocol (SNMP), May 1990.
33. J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Coexistence between version 1 and version 2 of the internet-standard network management framework. RFC 1908 (Draft Standard), January 1996. Obsoleted by RFC 2576, Available at **http://www.ietf.org/rfc/rfc1908.txt**.
34. J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Conformance statements for version 2 of the simple network management protocol (SNMPv2). RFC 1904 (Draft Standard), January 1996. Obsoleted by RFC 2580, Available at **http://www.ietf.org/rfc/rfc1904.txt**.
35. J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Introduction to community-based SNMPv2. RFC 1901 (Historic), Available at **http://www.ietf.org/rfc/rfc1901.txt**, January 1996.

36. J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Management information base for version 2 of the simple network management protocol (SNMPv2). RFC 1907 (Draft Standard), January 1996. Obsoleted by RFC 3418, Available at **http://www.ietf.org/rfc/rfc1907.txt**.
37. J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Protocol operations for version 2 of the simple network management protocol (SNMPv2). RFC 1905 (Draft Standard), January 1996. Obsoleted by RFC 3416, Available at **http://www.ietf.org/rfc/rfc1905.txt**.
38. J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Structure of management information for version 2 of the simple network management protocol (SNMPv2). RFC 1902 (Draft Standard), January 1996. Obsoleted by RFC 2578, Available at **http://www.ietf.org/rfc/rfc1902.txt**.
39. J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Textual conventions for version 2 of the simple network management protocol (SNMPv2). RFC 1903 (Draft Standard), January 1996. Obsoleted by RFC 2579, Available at **http://www.ietf.org/rfc/rfc1903.txt**.
40. J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Transport mappings for version 2 of the simple network management protocol (SNMPv2). RFC 1906 (Draft Standard), January 1996. Obsoleted by RFC 3417, Available at **http://www.ietf.org/rfc/rfc1906.txt**.
41. P. Chan, M. Mahoney, and M. Arshad. A machine learning approach to anomaly detection. Technical Report CS-2003-06, Florida Institute of Technology, Melbourne, FL, USA, March 2003.
42. M. Chapman, F. Dupuy, and G. Nilsson. An overview of the telecommunications information networking architecture. *Electronics and Communication Engineering Journal*, 8(3):135–141, June 1996.
43. M. Cheikhrouhou and J. Labetoulle. An efficient polling layer for snmp. In *Proceedings of the IFIP/IEEE Network Operations and Management Symposium*, pages 477–490, September 2000.
44. T. Chen and L. Hu. Internet performance monitoring. In *Proceedings of the IEEE*, volume 90, pages 1592–1603, September 2002.
45. W.J. Christmas, J. Kittler, and M. Petrou. Structural matching in computer vision using probabilistic relaxation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8:749–764, 1995.
46. F.R.K. Chung. *Spectral Graph Theory*. CBMS, Regional Conference Series in Mathematics. American Mathematical Society, 1997.
47. F.R.K. Chung and L. Lu. Connected components in random graphs with given expected degree sequences. *Annals of Combinatorics*, 6:125–145, 2002.
48. Cisco Systems. *NetFlow Services Solutions Guide*, 2005. Available at **http://www.cisco.com/univercd/cc/td/doc/cisintwk/intsolns/netflsol/nfwhite.htm**.
49. K. Claffy and T. Monk. What's next for internet data analysis? Status and challenges facing the community. In *Proceedings of the IEEE; Special Issue on Communications in the 21st Century*, volume 85, pages 1563–1571, October 1997.
50. M. Coates, A. Hero, R. Nowak, and B. Yu. Internet tomography. *IEEE Signal Processing Magazine*, 19:47–65, May 2002.
51. L.P. Cordella, P. Foggia, C. Sansone, and M. Vento. An improved algorithm for matching large graphs. In *Proceedings of the 3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition*, pages 149–159, 2001.
52. C. Cortes, L.D. Jackel, and W-P Chiang. Predicting failures of telecommunication paths: Limits on learning machine accuracy imposed by data quality. In *Proceeding of the International Workshop on Applications of Neural Networks to Telecommunications 2, Stockholm*, 1995.
53. T.M. Cover and J.A. Thomas. *Elements of Information Theory*. Wiley, 1991.

54. A. Cross, R. Wilson, and E. Hancock. Inexact graph matching with genetic search. *Journal of Pattern Recognition*, 30:953–970, 1997.
55. D.L. Davies and D.W Bouldin. A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1(4):224–227, 1979.
56. D. Denning. An intrusion detection model. *IEEE Transactions on Software Engineering*, SE-13(2):222–232, February 1987.
57. L.E. Diamond, M.E. Gaston, and M. Kraetzl. An observation of power law distribution in dynamic networks. In *Proceedings of the International Conference on Information, Decision and Control*, pages 101–105, Adelaide, 2002.
58. P. Dickinson. *Graph Based Techniques for Measurement of Intranet Dynamics*. PhD thesis, Institute for Telecommunications Research, University of South Australia, Adelaide, April 2006.
59. P. Dickinson, H. Bunke, A. Dadej, and M. Kraetzl. Median graphs and anomalous change detection in communications networks. In *Proceedings of the International Conference on Information, Decision and Control*, pages 59–64, Adelaide, 2002.
60. P. Dickinson, H. Bunke, A. Dadej, and M. Kraetzl. Similarity measure for hierarchical graph representation and its application to computer network monitoring. In *Proceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics*, volume IV, pages 457–462, Orlando, Florida, 2002.
61. P. Dickinson, H. Bunke, A. Dadej, and M. Kraetzl. A novel graph distance measure and its application to monitoring change in computer networks. In *Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics*, volume III, pages 333–338, Orlando, FL, 2003.
62. P. Dickinson, H. Bunke, A. Dadej, and M. Kraetzl. Matching graphs with unique node labels. *Pattern Analysis and Applications*, 7(3):243–254, 2004.
63. P.J. Dickinson, H. Bunke, A. Dadej, and M. Kraetzl. Application of median graphs in detection of anomalous change in communication networks. In *Proceedings of the 5th World Multiconference on Systemics, Cybernetics and Informatics*, volume 5, pages 194–197, Orlando, FL, 2001.
64. P.J. Dickinson, M. Kraetzl, H. Bunke, M. Neuhaus, and A. Dadej. Similarity measures for hierarchical representations of graphs with unique node labels. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(3):425–442, 2004. Special issue: Graph Matching in Pattern Recognition and Machine Vision.
65. G. A. Dirac. Some theorems on abstract graphs. *Proceedings of the London Mathematical Society*, 2(3):69–81, 1952.
66. R.C. Dubes. Cluster analysis and related issues. In C.H. Chen, L.F. Pau, and P. Wang, editors, *Handbook of Pattern Recognition and Computer Vision*, pages 3–32. World Scientific, 1993.
67. R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification*. John Wiley, 2nd edition, 2001.
68. N. Duffield and M. Grossglauser. Trajectory sampling for direct traffic observation. *IEEE/ACM Transactions on Networking*, 9(3):280–292, June 2001.
69. R. Caceresand N. Duffield, A. Feldmann, J. Friedmann, A. Greenberg, R. Greer, T. Johnson, C. Kalmanek, B. Krishnamurthy, D.Lavelle, P. Mishra, K. Ramakrishnan, J. Rexford, F. True, and J. van der Merwe. Measurement and analysis of IP network usage and behaviour. *IEEE Communications Magazine*, 38(5):144–151, May 2000.
70. J.C. Dunn. Well separated clusters and fuzzy partition. *Journal of Cybernetics*, pages 95–104, 1974.
71. R. Espinosa, M. Tripod, and S. Tomic. *Cisco Router Configuration and Troubleshooting*. New Riders, Indianapolis, Indiana, December 1998.

72. C. Fraleigh, C. Diot, S. Moon, P. Owezarski, D. Papiannaki, and F. Tobagi. Design and deployment of a passive monitoring infrastructure. In *Proceedings of the Thyrrhenian International Workshop on Digital Communications*, pages 556–575, London, UK, 2001. Springer-Verlag.
73. M. Friedman and A. Kandel. *Introduction to Pattern Recognition*. World Scientific, 1999.
74. G. Frobenius. Über Matrizen aus nicht negativen Elementen. *Sitzungsberichte der Preußischen Akademie der Wissenschaften*, 23:456–477, 1912.
75. G. Golub and C.F. van Loan. *Matrix Computations*. Johns Hopkins University Press, 1983.
76. L.A. Goodman and W.H. Kruskal. Measures of association for cross-classification. *Journal of the American Statistical Association*, pages 732–764, 1954.
77. M. Grossglauser and J. Rexford. Passive traffic measurement for IP operations. In K. Park and W. Willinger, editors, *The Internet as a Large-Scale Complex System*. Oxford University Press, 2002.
78. S. Günter and H. Bunke. Self-organizing map for clustering in the graph domain. *Pattern Recognition Letters*, 23(4):405–417, February 2002.
79. S. Günter and H. Bunke. Validation indices for graph clustering. *Pattern Recognition Letters*, 24(8):1107–1113, 2003.
80. D. Gusfield. *Algorithms on Strings, Trees and Sequences*. Cambridge University Press, 1997.
81. W.R. Hamilton. The Icosian game. Jaques and Son, 1859. Reprinted in [12].
82. G. Held. *LAN Testing and Troubleshooting: Reliability Tuning Techniques*. John Wiley, New York, June 1996.
83. J. Hellerstein and T.J. Watson. An approach to selecting metrics for detecting performance problems in information systems. *Proceedings of the 2nd IEEE International Workshop on Systems Management*, pages 30–39, 1996.
84. G.N. Higginbottom. *Performance Evaluation of Communication Networks*. Artech House, Massachusetts, 1998.
85. M. Hills. *Intranet Business Strategies*. John Wiley and Sons, 1996.
86. L.L. Ho, C.J. Macey, and R.G. Hiller. Real-time performance monitoring and anomaly detection in the internet: An adaptive, objective-driven, mix-and-match approach. *Bell Labs Technical Journal*, 4(4):23–41, October 1999.
87. C.S. Hood and C. Ji. Intelligent network monitoring. In *Proceedings of the IEEE Workshop on Neural Networks for Signal Processing*, pages 521–530, 1995.
88. C.S. Hood and C. Ji. Probabilistic network fault detection. In *Proceedings of the IEEE GLOBECOM*, volume 3, pages 1872–1876, 1996.
89. C.S. Hood and C. Ji. Proactive network-fault detection. *IEEE Transactions on Reliability*, 46(3):333–341, 1997.
90. C.S. Hood and C. Ji. Intelligent agents for proactive fault detection. *IEEE Internet Computing*, 2(2):65–72, March 1998.
91. J.E. Hopcroft and J.K. Wong. Linear time algorithm for isomorphism of planar graphs. In *Proceedings of the 6th Annual ACM Symposium on Theory of Computing*, pages 172–184, 1974.
92. R. Horn and C.R. Johnson. *Matrix Analysis*. Cambridge University Press, 1985.
93. M. Huber. Estimating the average shortest path length in a graph. Technical report, Cornell University, 1996.
94. B.A. Huberman and R.M. Lukose. Social dilemas and internet congestion. *Science*, 277(5325):535–537, July 1997.
95. B. Krishnamurthy J. Jung and M. Rabinovich. Flash crowds and denial of service attacks: Characterization and implications for CDNs and web sites. In *Proceedings of the*

International World Wide Web Conference, pages 252–262, Honolulu, Hawaii, May 2002. IEEE.

96. A.K. Jain, M. N. Murty, and P. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31:264–323, 1999.
97. G. Jakobson and M.D. Weissman. Alarm correlation. *IEEE Network Journal*, 7(6):52–59, 1993.
98. J.L. Jerkins and J.L. Wang. A close look at traffic measurements from packet networks. In *Proceedings of the IEEE GLOBECOM*, volume 4, pages 2405–2411, 1998.
99. X. Jiang and H. Bunke. Including geometry in graph representations: a quadratic-time graph isomorphism algorithm and its application. In P. Perner, P. Wang, and A. Rosenfeld, editors, *Advances in Structural and Syntactic Pattern Recognition*, volume 1121 of *LNCS*, pages 110–119. Springer, 1996.
100. X. Jiang and H. Bunke. On median graphs: Properties, algorithms, and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(10):1144–1151, 2001.
101. I. Katzela and M. Schwartz. Schemes for fault identification in communication networks. *IEEE/ACM Transactions on Networking*, 3(6):753–764, 1995.
102. M.G. Kendall. Further contributions to the theory of paired comparisons. *Biometrics*, 11:43–62, 1955.
103. K. Keys, D. Moore, R. Koga, E. Lagache, M. Tesch, and K. Claffy. The architecture of CoralReef: an Internet traffic monitoring software suite. In *PAM2001: Passive and Active Measurement Workshop*. CAIDA, RIPE NCC, April 2001.
104. S. Khuller and B. Raghavachari. Advanced combinatorial algorithms. In M.J. Atallah, editor, *Algorithms and Theory of Computation Handbook*, chapter 7. CRC Press, 1999.
105. T.P. Kirkman. On the representation of polyhedra. *Philosophical Transactions of the Royal Society, London*, 146:413–418, 1856.
106. T. Kohonen. *Self-Organizing Maps*. Springer-Verlag, New York, 1997.
107. J.L. Kolodner. *Case-Based Reasoning*. Morgan Kaufmann, San Francisco, California, 1993.
108. S. Kosinov and T. Caelli. Inexact multisubgraph matching using graph eigenspace and clustering models. In T. Caelli, A. Amin, R. Duin, M. Kamel, and D. de Ridder, editors, *Proceedings of the Structural, Syntactic, and Statistical Pattern Recognition workshop*, volume 2396 of *LNCS*, pages 133–142. Springer, 2002.
109. W. Kropatsch, M. Burge, S. Ben Jacoub, and N. Sehmanoui. Dual graph contraction with LEDA. *Computing Journal, Supplementum*, 12:101–110, 1998.
110. C. Kruegel and T. Toth. Using decision trees to improve signature-based intrusion detection. In *Proceedings of the 6th Symposium on Recent Advances in Intrusion Detection (RAID)*, volume 2820 of *LNCS*, pages 173–191, Pittsburgh, USA, 2003. Springer.
111. C. Kruegel and G. Vigna. Anomaly detection of web-based attacks. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, pages 251–261, October 2003.
112. M. Thottan L. Li, B. Yao, and S. Paul. Distributed network monitoring with bounded link utilization. In *Proceedings of the IEEE INFOCOM*, pages 1189–1198, March 2003.
113. A. Lakhina, M. Crovella, and C. Diot. Characterization of network-wide anomalies in traffic flows. In *Proceedings of the 4th ACM SIGCOMM conference on Internet Measurement*, pages 201–206, Taormina, Sicily, Italy, October 2004. ACM.
114. P. Lancaster and M. Tismenetsky. *The Theory of Matrices*. Academic Press, 1985.
115. J. Larrosa and G. Valiente. Constraint satisfaction algorithms for graph pattern matching. *Mathematical Structures in Computer Science*, 12(4):403–422, 2002.

116. A. Lazar, W. Wang, and R. Deng. Models and algorithms for network fault detection and identification: A review. In *Proceedings of the IEEE International Conference on Communications*, pages 999–1003, Singapore, November 1992.
117. A. Lazarevic, L. Ertoz, V. Kumar, A. Ozgur, and J. Srivastava. A comparative study of anomaly detection schemes in network intrusion detection. In *Proceedings of the 3rd SIAM Conference on Data Mining*, May 2003.
118. G. Levi. A note on the derivation of maximal common subgraphs of two directed or undirected graphs. *Calcolo*, 9:341 – 354, 1972.
119. L. Lewis. A case based reasoning approach to the managment of faults in communications networks. In *Proceedings of the IEEE INFOCOM*, volume 3, pages 1422–1429, San Francisco, CA, March 1993.
120. L. Lovász and M.D. Plummer. *Matching Theory*. North-Holland, Amsterdam, 1986.
121. E.M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of Computer and Systems Sciences*, 25:42–65, 1982.
122. B. Luo and E. Hancock. Structural graph matching using the em algorithm and singular value decomposition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23:1120–1136, 2001.
123. B. Luo, R. Wilson, and E. Hancock. Spectral feature vectors for graph clustering. In T. Caelli, A. Amin, R. Duin, M. Kamel, and D. de Ridder, editors, *Structural, Syntactic, and Statistical Pattern Recognition*, volume 2396 of *LNCS*, pages 83–93. Springer, 2002.
124. E. Madruga and L. Tarouco. Fault management tools for a cooperative and decentralised network operations environment. *IEEE Journal on Selected Areas in Communications*, 12(6):1121–1130, August 1994.
125. A. Magnaghi, T. Hamada, and T. Katsuyama. A wavelet-based framework for proactive detection of network misconfigurations. In *SIGCOMM 2004*, pages 253–258, August 2004.
126. M. Mahoney and P. Chan. Learning rules for anomaly detection of hostile network traffic. In *Proceedings of the 3rd IEEE International Conference on Data Mining*, pages 601–604, Washington, DC, USA, 2003. IEEE Computer Society.
127. R. Maxion and M. Feather. A case study of ethernet anomalies in a distributed computing environment. *IEEE Transactions on Reliability*, 39:433–443, October 1990.
128. K. McCloghrie and M. T. Rose. RFC 1213: Management information base for network management of TCP/IP-based internets:MIB-II, March 1991.
129. J. McGregor. Backtrack search algorithms and the maximal common subgraph problem. *Software – Practice and Experience*, 12(1):23–34, 1982.
130. B. McKay. Practical graph isomorphism. *Congressus Numerantium*, 30:45–87, 1981.
131. B. Messmer and H. Bunke. A new algorithm for error-tolerant sub-graph isomorphism detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(5):493–504, May 1998.
132. S. Milgram. The small world problem. *Psychology Today*, 2:60–67, 1967.
133. M. Miller. *Troubleshooting Internetworks*. Macmillan Technical, Indianapolis, Indiana, 1991.
134. T.M. Mitchell. *Machine Learning*. Mc Graw-Hill, 1997.
135. M.E.J. Newman. The structure and function of complex networks. *SIAM Review*, 45:167–256, 2003.
136. H. Niemann. *Pattern Analysis and Understanding*. Springer, 2nd edition, 1990.
137. C. Noble and D. Cook. Graph-based anomaly detection. In *SIGKDD*, pages 631–636, Washington, DC, USA, August 2003. ACM.
138. O. Ore. Note on Hamilton circuits. *American Mathematical Monthly*, 67:55, 1960.

139. D.D. Parkes and W.D. Wallis. *Graph Theory and the Study of Activity Structure*. Timing Space and Spacing Time, vol. 2: Human Activity and Time Geography. Edward Arnold, London, 1978.
140. M. Pelillo. Matching free trees, maximal cliques and monotone game dynamics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(11):1535–1541, 2002.
141. M. Pelillo and A. Jagota. Feasible and infeasible maxima in a quadratic program for maximum clique. *Journal of Artificial Neural Networks 2*, 2:411–420, 1995.
142. O. Perron. Zur Theorie der Matrizen. *Mathematische Annalen*, 64:248–263, 1907.
143. L. Pósa. A theorem concerning Hamilton lines. *Magyar Tudományos Akadémia Matematikai Kutató*, 7:225–226, 1962.
144. R.C. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36:1389–1401, 1957.
145. N. Pullman. *Matrix Theory and its Applications*. Dekker, 1976.
146. R. Quinland. *C4.5: Programs for Machine Learning*. Morgen Kaufmann Publ., 1993.
147. W.M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66:846–850, 1971.
148. B.D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, 1996.
149. S. Rizzi. Genetic operators for hierarchical graph clustering. *Pattern Recognition Letters*, 19:1293–1300, 1998.
150. I. Rouvellou. *Graph identification techniques applied to Network Management problems*. PhD thesis, Columbia University, 1993.
151. A. Sanfeliu and K.S. Fu. A distance measure between attributed relational graphs for pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, 13(3):353–362, 1983.
152. D. Sankoff and J. Kruskal, editors. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*. Addison Wesley, 1983.
153. S. Sarkar and K.L. Boyer. Quantitative measures of change based on feature organisation: Eigenvalues and eigenvectors. *Computer Vision and Image Understanding*, 71:110–136, 1998.
154. A. Schenker, M. Last, H. Bunke, and A. Kandel. *Graph Theoretic Techniques for Web Content Mining*. World Scientific, 2005.
155. D.S. Seong, H.S. Kuri, and K.H. Park. Incremental clustering of attributed graphs. *IEEE Transactions on System, Man and Cybernetics*, 23:1399–1411, 1993.
156. A. Shokonfandeh and S. Dickinson. A unified framework for indexing and matching hierarchical shape structures. In C. Arcelli, L. Cordella, and G. Sanniti di Baja, editors, *Visual Form 2001*, pages 67–84. Springer Verlag, LNCS 2059, 2001.
157. P. Shoubridge, M. Kraetzl, and D. Ray. Detection of abnormal change in dynamic networks. In *Proceedings of the International Conference on Information, Decision and Control*, pages 557–562, Adelaide, 1999.
158. P. Shoubridge, M. Kraetzl, W.D. Wallis, and H. Bunke. Detection of abnormal change in time series of graphs. *Journal of Interconnection Networks*, 3(1&2):85–101, 2002.
159. W. Stallings. *SNMP, SNMPv1, and CMIP: The Practical Guide to Network Management Standards*. Addison Wesley Publishing Company, 1997.
160. W. Stallings. SNMPv3: A security enhancement to snmp. *IEEE Communications Surveys*, 1(1):2–17, 1998.
161. W. Stallings. *Cryptography and Network Security: Principles and Practice*. Prentice-Hall, 1999.
162. W. Stallings. *SNMP, SNMPv1, SNMPv3 and RMON 1 and 2, Third Edition*. Addison-Wesley Publishing Company, September 1999.
163. G.A. Stephen. *String Searching Algorithms*. World Scientific, 1994.

164. A. Strehl, J. Ghosh, and R. Mooney. Impact of similarity measures on web-page clustering. In *Proceedings of the 17th AAAI Workshop of Artificial Intelligence for Web Search*, pages 58–64, 2000.
165. M. Subramanian. *Network Management: Principles and Practice*. Addison-Wesley, Reading, MA, 2000.
166. A. Tanenbaum. *Computer Networks*. Prentice Hall, third edition edition, 1996.
167. H. Tangmunarunkit, R. Govindan, S. Jamin, S. Shenker, and W. Willinger. Network topology generators: Degree-based vs structural. In *ACM Special Interest Group on Data Communications, SIGCOMM'02, Pittburgh, Pennyslvania, USA*, August 2002.
168. K. Terplan. *Intranet Performance Management*. CRC Press, Boca Raton, Florida, 2000.
169. S. Thomas. *IPng and the TCP/IP Protocols: implementing the next generation internet*, chapter 3, pages 43–92. John Wiley and Sons, New York, 1996.
170. G.L. Thompson. *Lectures on Game Theory Markov Chains and Related Topics*, volume SCR-11. Sandia Corporation Monograph, 1958.
171. M. Thottan and C. Ji. Proactive anomaly detection using distributed intelligent agents. *IEEE Network*, 12(5):21–27, September 1998.
172. M. Thottan and C. Ji. Anomaly detection in IP networks. *IEEE Transactions on Signal Processing*, 51(8):2191–2204, August 2003.
173. J.R. Ullman. An algorithm for subgraph isomorphism. *Journal of the ACM*, 23(1):31–42, 1976.
174. S. Umeyama. An eigendecomposition approach to weighted graph matching problems. *IEEE Transactions on Pattern Recognition and Machine Intelligence*, 10(5):695–703, September 1988.
175. T.J. Velte. Simulating your NT network. *Windows NT Magazine*, January 1999.
176. R.A. Wagner and M.J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21:168–173, 1974.
177. S. Waldbusser. RFC 1757: Remote network monitoring management information base, February 1995.
178. S. Waldbusser. RFC 2021: Remote network monitoring management information base version 2 using SMIv2, January 1997.
179. W. D. Wallis. One-factorizations of complete graphs. In *Contemporary Design Theory*, pages 593–631, New York, 1992. Wiley.
180. W. D. Wallis. *One-factorizations*. Kluwer, Dortrecht, Netherlands, 1997.
181. W.D. Wallis. *A Beginner's Guide to Graph Theory*. Birkhauser, 2000.
182. W.D. Wallis, P.J. Shoubridge, M. Kraetzl, and D. Ray. Graph distances using graph union. *Pattern Recognition Letters*, 22:701–704, 2001.
183. I. Wang, K-C. Fan, and J-T. Horng. Genetic-based search for error-correcting graph isomorphism. *IEEE Transactions on SMC*, 27:588–597, 1997.
184. D.J. Watts. *Small Worlds*. Princeton University Press, 1999.
185. D.J. Watts and S.H. Strogatz. Collective dynamics of "small-world" networks. *Nature*, 393:440–442, 1998.
186. A. Webb. *Statistical Pattern Recognition*. Oxford University Press, 1999.
187. T.H. Wei. *The Algebraic Foundations of Ranking Theory*. PhD thesis, Cambridge, 1952.
188. D.B. West. *Introduction to Graph Theory*. Prentice Hall, New Jersey, 1996.
189. C.C. White, E.A. Sykes, and J.A. Morrow. An analytical approach to the dynamic topology problem. *Telecommunication Systems*, 3:397–413, 1995.
190. J. De Wiele and S. Rabie. Meeting network management challenges: Customization, integration and scalability. In *Proceedings of the IEEE International Conference on Communications*, volume 2, pages 1197–1204, May 1993.

191. R.C. Wilson and E. Hancock. Structural matching by discrete relaxation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19:634–648, 1997.
192. Y. Yemini. The OSI network management model. *IEEE Communications Magazine*, pages 20–29, May 1993.
193. Y. Yemini. A critical survey of network management protocol standards. In *Telecommunications Network Management into the 21st Century (S.Aidarous and T Plevyak, eds)*. IEEE Press, 1994.
194. S.M.S. Zabir, A. Ashir, and N. Shiratori. Estimation of network performance: An approach based on network experience. In *Proceedings of the The 15th International Conference on Information Networking*, pages 657–662, Washington, DC, USA, 2001. IEEE Computer Society.
195. L.A. Zadek. Fuzzy sets. *Information and Control*, 8:338–353, 1965.
196. C.T. Zahn. Graph-theoretical methods for detecting and describing gestalt structures. *IEEE Transactions on Computers*, C-20:68–86, 1971.
197. F. Zhang and J.L. Hellerstein. An approach to on-line predictive detection. In *Proceedings of the 8th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*, pages 549–556, 2000.
198. T. Zseby, M. Molina, N. Duffield, S. Niccolini, and F. Raspall. Techniques for IP packet selection. Internet draft, IETF, February 2005.

Index