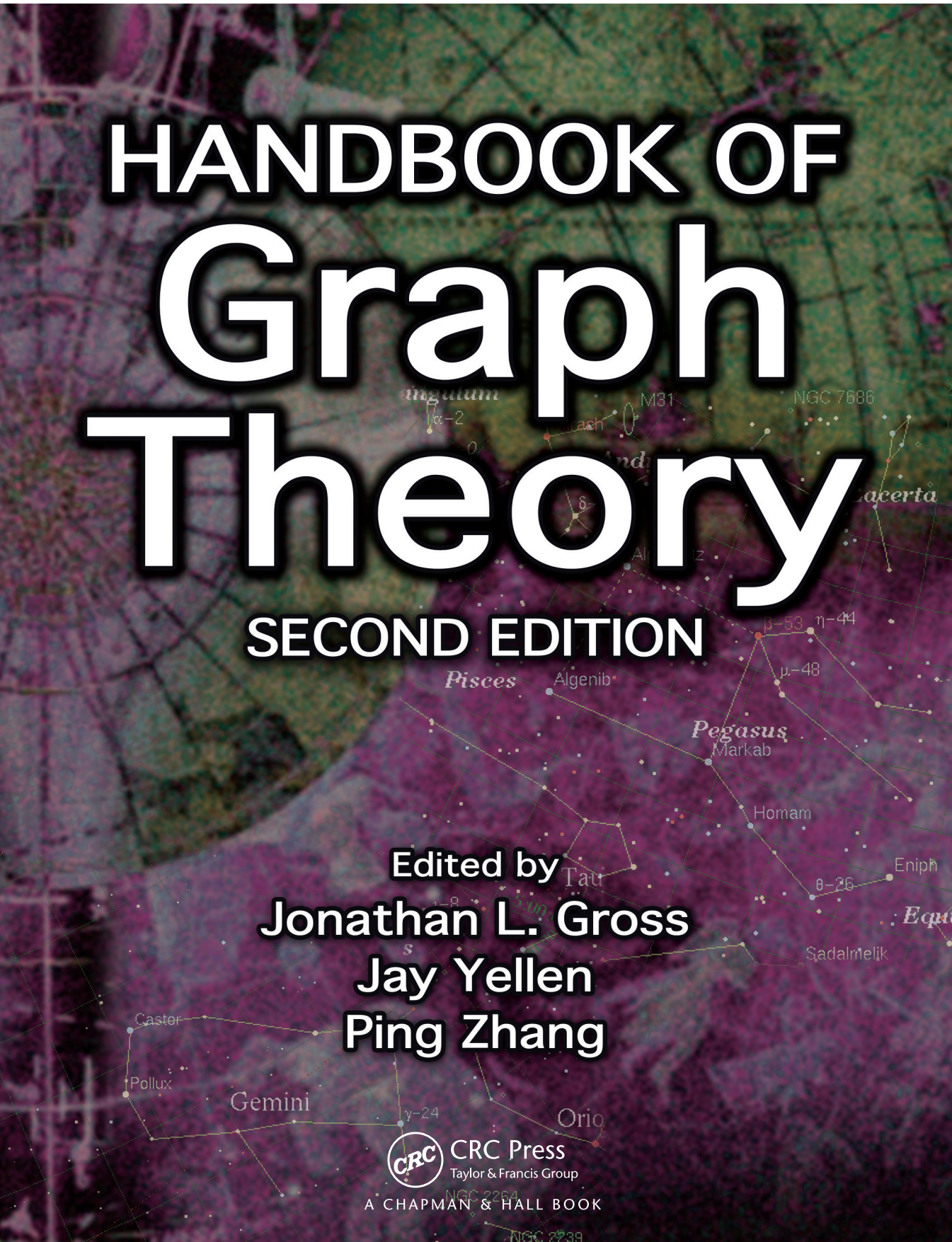
DISCRETE MATHEMATICS AND ITS APPLICATIONS
Series Editor KENNETH H. ROSEN

HANDBOOK OF Graph Theory

SECOND EDITION



Edited by
Jonathan L. Gross
Jay Yellen
Ping Zhang

CRC Press
Taylor & Francis Group
A CHAPMAN & HALL BOOK

HANDBOOK OF Graph Theory

SECOND EDITION

DISCRETE MATHEMATICS AND ITS APPLICATIONS

R. B. J. T. Allenby and Alan Slomson, How to Count: An Introduction to Combinatorics, Third Edition

Craig P. Bauer, Secret History: The Story of Cryptology

Juergen Bierbrauer, Introduction to Coding Theory

Katalin Bimbó, Combinatory Logic: Pure, Applied and Typed

Donald Bindner and Martin Erickson, A Student's Guide to the Study, Practice, and Tools of Modern Mathematics

Francine Blanchet-Sadri, Algorithmic Combinatorics on Partial Words

Miklós Bóna, Combinatorics of Permutations, Second Edition

Jason I. Brown, Discrete Structures and Their Interactions

Richard A. Brualdi and Dragoš Cvetković, A Combinatorial Approach to Matrix Theory and Its Applications

Kun-Mao Chao and Bang Ye Wu, Spanning Trees and Optimization Problems

Charalambos A. Charalambides, Enumerative Combinatorics

Gary Chartrand and Ping Zhang, Chromatic Graph Theory

Henri Cohen, Gerhard Frey, et al., Handbook of Elliptic and Hyperelliptic Curve Cryptography

Charles J. Colbourn and Jeffrey H. Dinitz, Handbook of Combinatorial Designs, Second Edition

Abhijit Das, Computational Number Theory

Martin Erickson, Pearls of Discrete Mathematics

Martin Erickson and Anthony Vazzana, Introduction to Number Theory

Steven Furino, Ying Miao, and Jianxing Yin, Frames and Resolvable Designs: Uses, Constructions, and Existence

Mark S. Gockenbach, Finite-Dimensional Linear Algebra

Randy Goldberg and Lance Riek, A Practical Handbook of Speech Coders

Titles (continued)

Jacob E. Goodman and Joseph O'Rourke, Handbook of Discrete and Computational Geometry, Second Edition

Jonathan L. Gross, Combinatorial Methods with Computer Applications

Jonathan L. Gross and Jay Yellen, Graph Theory and Its Applications, Second Edition

Jonathan L. Gross, Jay Yellen, and Ping Zhang Handbook of Graph Theory, Second Edition

David S. Gunderson, Handbook of Mathematical Induction: Theory and Applications

Richard Hammack, Wilfried Imrich, and Sandi Klavžar, Handbook of Product Graphs, Second Edition

Darrel R. Hankerson, Greg A. Harris, and Peter D. Johnson, Introduction to Information Theory and Data Compression, Second Edition

Darel W. Hardy, Fred Richman, and Carol L. Walker, Applied Algebra: Codes, Ciphers, and Discrete Algorithms, Second Edition

Daryl D. Harms, Miroslav Kraetzl, Charles J. Colbourn, and John S. Devitt, Network Reliability: Experiments with a Symbolic Algebra Environment

Silvia Heubach and Toufik Mansour, Combinatorics of Compositions and Words

Leslie Hogben, Handbook of Linear Algebra, Second Edition

Derek F. Holt with Bettina Eick and Eamonn A. O'Brien, Handbook of Computational Group Theory

David M. Jackson and Terry I. Visentin, An Atlas of Smaller Maps in Orientable and Nonorientable Surfaces

Richard E. Klima, Neil P. Sigmon, and Ernest L. Stitzinger, Applications of Abstract Algebra with Maple™ and MATLAB®, Second Edition

Richard E. Klima and Neil P. Sigmon, Cryptology: Classical and Modern with Maplets

Patrick Knupp and Kambiz Salari, Verification of Computer Codes in Computational Science and Engineering

William Kocay and Donald L. Kreher, Graphs, Algorithms, and Optimization

Donald L. Kreher and Douglas R. Stinson, Combinatorial Algorithms: Generation Enumeration and Search

Hang T. Lau, A Java Library of Graph Algorithms and Optimization

C. C. Lindner and C. A. Rodger, Design Theory, Second Edition

San Ling, Huaxiong Wang, and Chaoping Xing, Algebraic Curves in Cryptography

Nicholas A. Loehr, Bijective Combinatorics

Toufik Mansour, Combinatorics of Set Partitions

Alasdair McAndrew, Introduction to Cryptography with Open-Source Software

Elliott Mendelson, Introduction to Mathematical Logic, Fifth Edition

Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone, Handbook of Applied Cryptography

Titles (continued)

Stig F. Mjølsnes, A Multidisciplinary Introduction to Information Security

Jason J. Molitierno, Applications of Combinatorial Matrix Theory to Laplacian Matrices of Graphs

Richard A. Mollin, Advanced Number Theory with Applications

Richard A. Mollin, Algebraic Number Theory, Second Edition

Richard A. Mollin, Codes: The Guide to Secrecy from Ancient to Modern Times

Richard A. Mollin, Fundamental Number Theory with Applications, Second Edition

Richard A. Mollin, An Introduction to Cryptography, Second Edition

Richard A. Mollin, Quadratics

Richard A. Mollin, RSA and Public-Key Cryptography

Carlos J. Moreno and Samuel S. Wagstaff, Jr., Sums of Squares of Integers

Gary L. Mullen and Daniel Panario, Handbook of Finite Fields

Goutam Paul and Subhamoy Maitra, RC4 Stream Cipher and Its Variants

Dingyi Pei, Authentication Codes and Combinatorial Designs

Kenneth H. Rosen, Handbook of Discrete and Combinatorial Mathematics

Douglas R. Shier and K.T. Wallenius, Applied Mathematical Modeling: A Multidisciplinary Approach

Alexander Stanoyevitch, Introduction to Cryptography with Mathematical Foundations and Computer Implementations

Jörn Steuding, Diophantine Analysis

Douglas R. Stinson, Cryptography: Theory and Practice, Third Edition

Roberto Tamassia, Handbook of Graph Drawing and Visualization

Roberto Togneri and Christopher J. deSilva, Fundamentals of Information Theory and Coding Design

W. D. Wallis, Introduction to Combinatorial Designs, Second Edition

W. D. Wallis and J. C. George, Introduction to Combinatorics

Jiacun Wang, Handbook of Finite State Based Models and Applications

Lawrence C. Washington, Elliptic Curves: Number Theory and Cryptography, Second Edition

DISCRETE MATHEMATICS AND ITS APPLICATIONS
Series Editor KENNETH H. ROSEN

HANDBOOK OF Graph Theory

SECOND EDITION

Edited by

Jonathan L. Gross
Columbia University
New York, USA

Jay Yellen
Rollins College
Winter Park, Florida, USA

Ping Zhang
Western Michigan University
Kalamazoo, USA

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business
A CHAPMAN & HALL BOOK

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

Version Date: 20130923

International Standard Book Number-13: 978-1-4398-8019-7 (eBook - PDF)

Jonathan dedicates this book to Hadas, Noa, Nili, Tirzah, Eli Chaim, Bezi, Benjamin, Naomi, Rebecca, Abigail, Shayna, Alice, and Ruth.

Jay dedicates this book to Betsey and Tara.

Ping dedicates this book to Gary Chartrand.

CONTENTS

PREFACE

Over the past fifty years, graph theory has been one of the most rapidly growing areas of mathematics. Since 1960, more than 10,000 different authors have published papers classified as graph theory by *Math Reviews*, and for the past decade, more than 1000 graph theory papers have been published each year. Not surprisingly, this Second Edition is about 450 pages longer than the First Edition, which appeared in 2004.

This *Handbook* is intended to provide as comprehensive a view of graph theory as is feasible in a single volume. Many of our chapters survey areas that have large research communities, with hundreds of active mathematicians, and which could be developed into independent handbooks. The 89 contributors to this volume, 31 of whom are new to this edition, collectively represent perhaps as much as 90% or more of the main topics in pure and applied graph theory. Thirteen of the sections in the Second Edition cover newer topics that did not appear in the First Edition.

Format

In order to achieve this kind of comprehensiveness, we challenged our contributors to restrict their expository prose to a bare minimum, by adhering to the ready-reference style of the CRC Handbook series, which emphasizes quick accessibility for the non-expert. We thank the contributors for responding so well to this challenge.

The 13 chapters of the *Handbook* are organized into 65 sections. Within each section, several major topics are presented. For each topic, there are lists of the essential definitions and facts, accompanied by examples, tables, remarks, and in some cases, conjectures and open problems. Each section ends with a bibliography of references tied directly to that section. In many cases, these bibliographies are several pages long, providing extensive guides to the research literature and pointers to monographs.

To ensure that each section be reasonably self-contained, we encouraged contributors to include some definitions that may have appeared in earlier sections. Each contributor was also asked to include a glossary with his or her section. These section glossaries were then merged by the editors into 13 chapter glossaries.

Terminology and Notation

Graph theory has attracted mathematicians and scientists from diverse disciplines and, accordingly, is blessed (and cursed) with a proliferation of terminology and notations. Since the *Handbook* objective is to survey topics for persons whose expertise may be elsewhere, either on other topics, or outside of graph theory, we asked our contributors to tilt toward the general usage in the mathematical community, rather than staying strictly within the idioms of their specialties. But to understand graph theory literature, it helps to accept the legacy of history. As editors, we tried to strike a balance between preserving the notation and terminology that evolved from each area's rich history and our desire create a cohesive, uniform body of material.

Some uniformity of usage came easily. In general, the word *graph* is used inclusively to refer to graphs with directed edges and/or to graphs with multi-edges and self-loops. In most sections, G denotes a graph and V and E denote its vertex- and edge-sets, respectively.

However, some words are used differently by different graph theory communities. For instance, to an algebraic graph theorist, a *Cayley graph* is simple, connected, and undirected, but to a topological graph theorist, it may be non-connected, possibly directed, and have multi-edges and/or self-loops. To some graph theorists, a *clique* is a complete subgraph, maximal under set inclusion, and to others maximality is not required.

Consistency in notation was also problematic. In the literature of graph coloring, the Greek letter χ is used as the chromatic number, but to an algebraic topologist, it means the Euler characteristic.

Notes regarding terminology and notation were added to make these variations explicit, thereby improving cross-chapter compatibility.

Acknowledgments

We would like to thank Bob Stern of CRC Press for his continued enthusiasm and patience during the gestation period and Bob Ross at CRC for providing the final support in bringing the *Handbook* to publication.

Jonathan Gross, Jay Yellen, and Ping Zhang

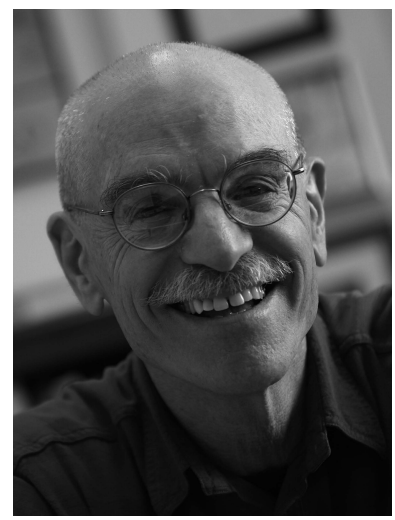
About the Editors

Jonathan Gross is professor of computer science at Columbia University. His research in topology, graph theory, and cultural sociometry has earned him an Alfred P. Sloan Fellowship, an IBM Postdoctoral Fellowship, and various research grants from the Office of Naval Research, the National Science Foundation, and the Russell Sage Foundation.

Professor Gross is the inventor of the voltage graph, a construct widely used in topological graph theory and in other branches as well. His main current research interest is the genus distribution of graphs. His other recent areas of research publication include computer graphics and knot theory. He has received several awards for outstanding teaching at Columbia University, including the career Great Teacher Award from the Society of Columbia Graduates. He appears on the Columbia Video Network and on the video network of the National Technological University.

Prior to Columbia University, Professor Gross was in the mathematics department at Princeton University. His undergraduate work was at M.I.T., and he wrote his Ph.D. thesis on 3-dimensional topology at Dartmouth College.

His previous books include *Topological Graph Theory*, coauthored with Thomas W. Tucker, *Graph Theory and Its Applications*, coauthored with Jay Yellen, and *Combinatorial Methods with Computer Applications*. Another previous book, *Measuring Culture*, coauthored with Steve Rayner, constructs network-theoretic tools for measuring sociological phenomena.



Jay Yellen is Archibald Granville Bush Professor of Mathematics at Rollins College. He received his B.S. and M.S. in mathematics at Polytechnic University of New York and did his doctoral work in finite group theory at Colorado State University. Dr. Yellen has had regular faculty appointments at Allegheny College, the State University of New York at Fredonia, and the Florida Institute of Technology, where he was chair of Operations Research from 1995 to 1999. He has had visiting appointments at Emory University, Georgia Institute of Technology, Columbia University, and the University of Nottingham, UK.

In addition to his book *Graph Theory and Its Applications*, coauthored with Professor Gross, Professor Yellen has written manuscripts used at IBM for two

courses in discrete mathematics within the Principles of Computer Science Series and has contributed two sections to the *Handbook of Discrete and Combinatorial Mathematics*. He also has designed and conducted several summer workshops on creative problem solving for secondary-school mathematics teachers, which were funded by the National Science Foundation and New York State. At Rollins, he has received the Hugh F. McKean Award for Outstanding Teaching, the Student's Choice Professor Award, and the Hugh F. McKean Research Grant Award.

Dr. Yellen has published research articles in character theory of finite groups, graph theory, power-system scheduling, and timetabling. His current research interests include graph theory, discrete optimization, and graph algorithms for software testing and course timetabling.

Ping Zhang is professor of mathematics at Western Michigan University. She wrote her Ph.D. thesis on algebraic combinatorics at Michigan State University. Her previous books, coauthored with Gary Chartrand, include *Graphs & Digraphs* (5th edition), *Mathematical Proofs: A Transition to Advanced Mathematics* (3rd edition), *Chromatic Graph Theory*, *A First Course in Graph Theory* and *Discrete Mathematics*. The first of these was also coauthored with Linda Lesniak and the second coauthored with Albert D. Polimeni. Her research interests are algebraic combinatorics and colorings, distance and convexity, traversability, decompositions, and domination within graph theory.

CONTRIBUTORS

Alfred V. Aho
Columbia University

Brian Alspach
University of Newcastle, Australia

Dan Archdeacon
University of Vermont

David C. Arney
West Point

Camino Balbuena
Universitat Politècnica de Catalunya, Spain

Lowell W. Beineke
Purdue University at Fort Wayne

Jacek Blazewicz
Poznan University of Technology, Poland

Béla Bollobás
University of Memphis
Trinity College, Cambridge, UK

Anthony Bonato
Ryerson University, Canada

Danail Bonchev
Virginia Commonwealth University

Richard B. Borie
University of Alabama

Robert C. Brigham
University of Central Florida

Edmund Burke
University of Stirling, Scotland

Gary Chartrand
Western Michigan University

Jianer Chen
Texas A&M University

Maria Chudnovsky
Columbia University

Fan Chung
University of California, San Diego

Alice M. Dean
Skidmore College

Camil Demetrescu
University of Rome La Sapienza, Italy

A. K. Dewdney
University of Western Ontario, Canada

Dominique de Werra
École Polytechnique Fédérale, de Lausanne, Switzerland

Emilio Di Giacomo
University of Perugia, Italy

Michael Doob
University of Manitoba, Canada

Ernesto Estrada
University of Strathclyde, Scotland

Josep Fàbrega
Universitat Politècnica de Catalunya, Spain

Ralph J. Faudree
University of Memphis

Irene Finocchi
University of Rome La Sapienza, Italy

Miquel Àngel Fiol
Universitat Politècnica de Catalunya, Spain

Lisa Fleischer
Dartmouth College

Herbert Fleischner
Technical University of Vienna, Austria

Chapter 1

Introduction to Graphs

Section 1.1
Fundamentals of Graph Theory

Jonathan L. Gross, Columbia University
Jay Yellen, Rollins College

INTRODUCTION

Configurations of nodes and connections occur in a great diversity of applications. They may represent physical networks, such as electrical circuits, roadways, or organic molecules. They are also used in representing less tangible interactions as might occur in ecosystems, sociological relationships, databases, or in the flow of control in a computer program.

1.1.1 Graphs and Digraphs

Any mathematical object involving points and connections between them may be called a *graph*. If all the connections are unidirectional, it is called a *digraph*. Our highly inclusive definition in this initial section of the *Handbook* permits fluent discussion of almost any particular modification of the basic model that has ever been called a graph.

Basic Terminology

DEFINITIONS

D1: A ***graph*** $G = (V, E)$ consists of two sets V and E.

- The elements of V are called ***vertices*** (or ***nodes***).
- The elements of E are called ***edges***.
- Each edge has a set of one or two vertices associated to it, which are called its ***endpoints***. An edge is said to ***join*** its endpoints.

NOTATION: The subscripted notations V_G and E_G (or $V(G)$ and $E(G)$) are used for the vertex- and edge-sets when G is not the only graph under consideration.

D2: If vertex v is an endpoint of edge e, then v is said to be ***incident*** on e, and e is incident on v.

D3: A vertex u is ***adjacent*** to vertex v if they are joined by an edge.

D4: Two adjacent vertices may be called ***neighbors***.

D5: ***Adjacent edges*** are two edges that have an endpoint in common.

D6: A ***proper edge*** is an edge that joins two distinct vertices.

D7: A ***multi-edge*** is a collection of two or more edges having identical endpoints.

D8: A ***simple adjacency*** between vertices occurs when there is exactly one edge between them.

D9: The ***edge-multiplicity*** between a pair of vertices u and v is the number of edges between them.

D10: A ***self-loop*** is an edge that joins a single endpoint to itself.

TERMINOLOGY

- An alternative term for *self-loop* is ***loop***. This can be used in contexts in which *loop* has no other meanings.
- In computer science, the term *graph* is commonly used either to mean a graph as defined here, or to mean a computer-represented data structure whose value is a graph.

EXAMPLE

E1: A line drawing of a graph $G = (V, E)$ is shown in Figure 1.1.1. It has vertex-set $V = \{u, v, w, x\}$ and edge-set $E = \{a, b, c, d, e, f\}$. The set $\{a, b\}$ is a multi-edge with endpoints u and v, and edge c is a self-loop.

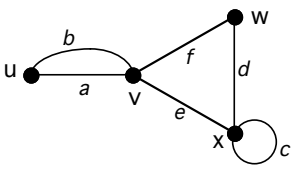


Figure 1.1.1: A graph.

REMARKS

R1: A graph is realized in a plane or in 3-space as a set of points, representing the vertices, and a set of curved or straight line segments, representing the edges. The curvature or length of such a line segment is irrelevant to the meaning. However, if a *direction* is indicated, that is significant.

R2: Occasionally, a graph is *parametrized* so that each edge is regarded as the homeomorphic image of the real interval $[0, 1]$ (except that for a self-loop, the endpoints 0 and 1 have the same image).

Simple Graphs

Most of theoretical graph theory is concerned with *simple* graphs. This is partly because many problems regarding general graphs can be reduced to problems about simple graphs.

DEFINITIONS

D11: A ***simple graph*** is a graph that has no self-loops or multi-edges.

D12: A ***trivial graph*** is a graph consisting of one vertex and no edges.

D13: A ***null graph*** is a graph whose vertex- and edge-sets are empty.

Edge Notation for Simple Adjacencies and for Multi-Edges

NOTATION: An edge joining vertices u and v of a graph may be denoted by the juxtaposition uv if it is the only such edge. Occasionally, the ordered pair (u, v) is used in this situation, instead of uv. To avoid ambiguities when multi-edges exist, or whenever else desired, the edges of a general graph may be given their own names, as in Figure 1.1.1 above.

EXAMPLE

E2: The simple graph shown in Figure 1.1.2 has edge-set $E = \{uv, vw, vx, wx\}$.

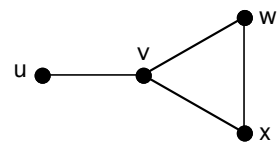


Figure 1.1.2: A simple graph.

General Graphs

Many applications require non-simple graphs as models. Moreover, some non-simple graphs serve an essential role in theoretical constructions, especially in constructing graph drawings (simple and non-simple) on surfaces (see Chapter 7).

TERMINOLOGY NOTE: Although the term "graph" means that self-loops and multi-edges are allowed, sometimes, for emphasis, the term ***general graph*** is used.

DEFINITIONS

D14: A ***loopless graph*** is a graph that has no self-loops. (It might have multi-edges.) Sometimes a loopless graph is referred to as a ***multigraph***.

D15: The ***dipole*** D_n is a loopless graph with two vertices and n edges joining them.

D16: The ***bouquet*** B_n is a graph with one vertex and n self-loops.

EXAMPLES

E3: The loopless graph in Figure 1.1.3 depicts the benzene molecule C_6H_6.

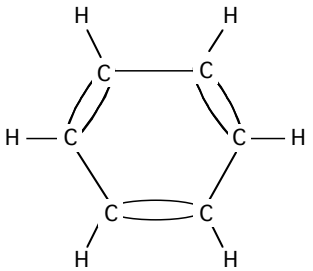


Figure 1.1.3: Graph model for a benzene ring.

E4: The dipole D_3 is shown in Figure 1.1.4.

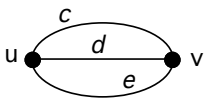


Figure 1.1.4: The loopless graph D_3.

E5: Two graphs with self-loops are shown in Figure 1.1.5.

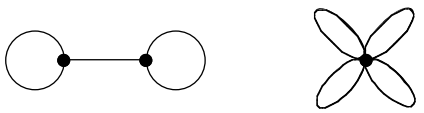


Figure 1.1.5: The dumbbell graph and the bouquet B_4.

Attributes

Allowing graphs to have additional attributes beyond vertices and edges enables them to serve as mathematical models for a wide variety of applications. Two of the most common additional edge attributes, both described in great detail later in the *Handbook*, are edge *direction* (e.g., Chapters 3 and 11) and edge *weight* (e.g., Chapters 4 and 11). Another common attribute (for edges or vertices) is *color*. Graph coloring is discussed in Chapter 5.

DEFINITIONS

D17: A ***vertex attribute*** is a function from the vertex-set to some set of possible attribute values.

D18: An ***edge attribute*** is a function from the edge-set to some set of possible attribute values.

Digraphs

An edge between two vertices creates a connection in two opposite senses at once. Assigning a direction makes one of these senses *forward* and the other *backward*. Viewing direction as an edge attribute is partly motivated by its impact on computer implementations of graph algorithms. Moreover, from a mathematical perspective, regarding *directed graphs* as augmented graphs makes it easier to view certain results that tend to be established separately for graphs and for digraphs as a single result that applies to both. The attribute of edge direction is developed extensively in Chapter 3 and elsewhere in this *Handbook*.

DEFINITIONS

D19: A ***directed edge*** (or ***arc***) is an edge e, one of whose endpoints is designated as the ***tail***, and whose other endpoint is designated as the ***head***. They are denoted $head(e)$ and $tail(e)$, respectively.

TERMINOLOGY: A directed edge is said to be ***directed from*** its tail and ***directed to*** its head. (The tail and the head of a directed self-loop are the same vertex.)

NOTATION: In a line drawing, the arrow points toward the head.

D20: A ***multi-arc*** is a set of two or more arcs having the same tail and same head.

D21: A ***digraph*** (or ***directed graph***) is a graph each of whose edges is directed.

D22: A ***simple digraph*** is a digraph with no self-loops and no multi-arcs.

D23: A ***mixed graph*** (or ***partially directed graph***) is a graph that has both undirected and directed edges. In a mixed graph, using the unmodified term *edge* avoids specifying whether the edge is directed or undirected.

D24: The ***underlying graph*** of a directed or partially directed graph G is the graph that results from removing all the designations of *head* and *tail* from the directed edges of G (i.e., deleting all the edge-directions).

Ordered-Pair Representation of Arcs

NOTATION: In a simple digraph, an arc from vertex u to vertex v is commonly denoted (u, v) (or sometimes uv). When multi-arcs are possible, using distinct names is often necessary.

COMPUTATIONAL NOTE: (*A caution to software designers*) From the perspective of object-oriented software design, the ordered-pair representation of arcs in a digraph treats digraphs as a different class of objects from graphs. This could seriously undermine *software reuse*. Large portions of computer code might have to be rewritten in order to adapt an algorithm that was originally designed for a digraph to work on an undirected graph.

The ordered-pair representation could also prove awkward in implementing algorithms for which the graphs or digraphs are *dynamic* structures (i.e., they change during the

algorithm). Whenever the direction on a particular edge must be reversed, the associated ordered pair has to be deleted and replaced by its reverse. Even worse, if a directed edge is to become undirected, then an ordered pair must be replaced with an unordered pair. Similarly, the undirected and directed edges of a partially directed graph would require two different types of objects.

EXAMPLES

E6: The digraph on the left in Figure 1.1.6 has the undirected graph on the right as its underlying graph. The digraph has two multi-arcs: $\{a, b\}$ and $\{f, h\}$.

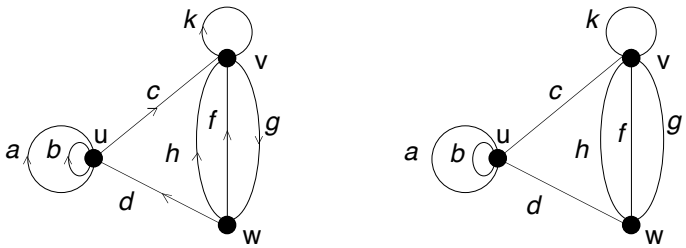


Figure 1.1.6: A digraph and its underlying graph.

E7: A simple digraph can have one arc in each direction between two vertices, as illustrated in Figure 1.1.7.

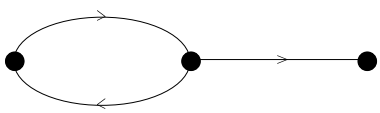


Figure 1.1.7: A simple digraph whose underlying graph is not simple.

Vertex-Coloring

When the vertex-set of a graph is partitioned, the cells of the partition are commonly assigned distinct *colors*. This is developed at length in Chapter 5.

DEFINITIONS

D25: A ***vertex-coloring*** of a graph G is a function from its vertex-set V_G vertices to a set C whose elements are called *colors*.

D26: A vertex-coloring is ***proper*** if two adjacent vertices are always assigned different colors.

D27: A graph is ***c-colorable*** if it has a proper vertex-coloring with c or fewer colors.

D28: The (***vertex***) ***chromatic number of a graph*** G, denoted $\chi(G)$, is the smallest number c of colors such that G is c-colorable.

REMARK

R3: Definitions of ***edge-coloring***, *c*-***edge-colorable***, and ***edge-chromatic number***, denoted $\chi'(G)$, are obtained by simply replacing the word "vertices" with the word "edges" in the definitions above.

EXAMPLE

E8: The graph G in Figure 1.1.8 is shown with a 3-coloring of its vertex-set. Since it is not 2-colorable, its chromatic number is 3. Also, the graph is easily seen to be 3-edge-colorable and clearly is not 2-edge-colorable; hence, $\chi'(G) = 3$.

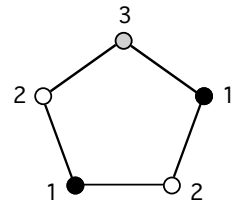


Figure 1.1.8: A graph G with $\chi(G) = \chi'(G) = 3$.

1.1.2 Degree and Distance

Two of the most fundamental notions in graph theory are those of the *degree* of a vertex and the *distance* between two vertices. Distance is developed fully in Chapter 9.

Degree

DEFINITIONS

D29: The ***degree*** (or ***valence***) of a vertex v in a graph G, denoted $deg(v)$, is the number of proper edges incident on v plus twice the number of self-loops. (For simple graphs, of course, the degree is simply the number of neighbors.)

TERMINOLOGY: Applications of graph theory to physical chemistry motivate the use of the term ***valence*** as an alternative to *degree*. Thus, a vertex of degree d is also called a d-***valent vertex***.

D30: The ***degree sequence*** of a graph is the sequence formed by arranging the vertex degrees into non-decreasing order.

D31: The ***indegree*** of a vertex v in a digraph is the number of arcs directed to v; the ***outdegree*** of vertex v is the number of arcs directed from v. Each self-loop at v counts one toward the indegree of v and one toward the outdegree.

D32: An ***isolated vertex*** in a graph is a vertex of degree 0.

EXAMPLES

E9: The graph in Figure 1.1.9 has degree sequence $< 0, 1, 1, 4, 6, 6 >$. Vertices u and v both have degree 6.

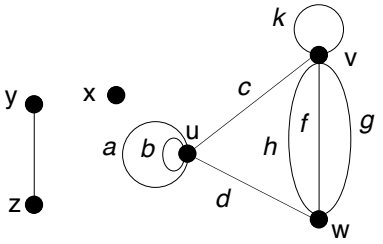
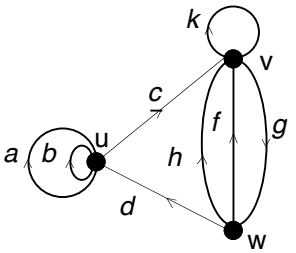


Figure 1.1.9: A graph with degree sequence $< 0, 1, 1, 4, 6, 6 >$.

E10: Figure 1.1.10 below shows the indegrees and outdegrees of a digraph.



vertex	*u*	*v*	*w*
indegree	3	4	1
outdegree	3	2	3

Figure 1.1.10: The indegrees and outdegrees of the vertices of a digraph.

FACTS

For proofs of the following elementary facts, see [GrYe06, §1.1] or other basic texts.

F1: (Euler) The sum of the degrees of the vertices of a graph is twice the number of edges.

F2: In every graph, the number of vertices having odd degree is an even number.

F3: A non-trivial simple graph G must have at least one pair of vertices whose degrees are equal.

F4: In a digraph, the sum of the indegrees and the sum of the outdegrees both equal the number of edges.

F5: The degree sequence of a graph is a finite, non-decreasing sequence of non-negative integers whose sum is even.

F6: Conversely, any non-decreasing, non-negative sequence of integers whose sum is even is the degree sequence of some graph, but not necessarily of a simple graph.

Walks, Trails, and Paths

DEFINITIONS

D33: A ***walk*** in a graph G is an alternating sequence of vertices and edges,

$$W = v_0, e_1, v_1, e_1, \ldots, e_n, v_n$$

such that for $j = 1, \ldots, n$, the vertices v_{j-1} and v_j are the endpoints of the edge e_j. If, moreover, the edge e_j is directed from v_{j-1} to v_j, then W is a ***directed walk***.

- In a simple graph, a walk may be represented simply by listing a sequence of vertices: $W = v_0, v_1, \ldots, v_n$ such that for $j = 1, \ldots, n$, the vertices v_{j-1} and v_j are adjacent.
- The ***initial vertex*** is v_0.
- The ***final vertex*** (or ***terminal vertex***) is v_n.
- An ***internal vertex*** is a vertex that is neither initial nor final.

D34: The ***length of a walk*** is the number of edges (counting repetitions).

D35: A walk is ***closed*** if the initial vertex is also the final vertex; otherwise, it is ***open***.

D36: A ***trail*** in a graph is a walk such that no edge occurs more than once.

D37: An ***eulerian trail*** in a graph G is a walk that contains each edge of G *exactly* once. (See §4.2.)

D38: A ***path*** in a graph is a trail such that no internal vertex is repeated.

D39: A ***cycle*** is a closed path of length at least 1.

D40: A ***trivial*** walk, trail, or path consists of a single vertex and no edges.

EXAMPLE

E11: In the graph shown in Figure 1.1.11, the vertex sequence $\langle u, v, x, v, z \rangle$ represents a walk that is not a trail, and the vertex sequence $\langle u, v, x, y, v, z \rangle$ represents a trail that is not a path.

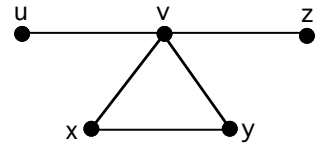


Figure 1.1.11: A graph.

Distance and Connectivity

DEFINITIONS

D41: The ***distance between two vertices*** in a graph is the length of the shortest walk between them.

D42: The ***directed distance from*** a vertex u ***to*** a vertex v in a digraph is the length of the shortest directed walk from u to v.

D43: A graph is ***connected*** if between every pair of vertices there is a walk.

D44: A digraph is ***(weakly) connected*** if its underlying graph is connected.

D45: A digraph is ***strongly connected*** if from each vertex to each other vertex there is a directed walk.

D46: The ***eccentricity*** of a vertex v in a connected graph is its distance to a vertex farthest from v.

D47: The ***radius*** of a connected graph is its minimum eccentricity.

D48: The ***diameter*** of a connected graph is its maximum eccentricity.

EXAMPLE

E12: The digraph shown on the left in Figure 1.1.12 is strongly connected; the digraph on the right is connected but not strongly connected.

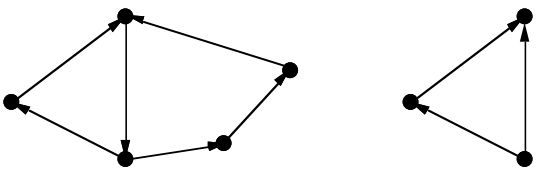


Figure 1.1.12: A strongly connected digraph and a weakly connected one.

1.1.3 Basic Structural Concepts

We are concerned with the possible equivalence of two graphs, with the symmetries of an individual graph, and with the possible appearance of one graph within another graph.

Isomorphism

In concept, two graphs are *isomorphic* if they are structurally identical, which means that they correspond in all structural details. A formal vertex-to-vertex and edge-to-edge correspondence is called an *isomorphism*.

DEFINITIONS

D49: An ***isomorphism between two simple graphs*** G and H is a vertex bijection $\phi : V_G \to V_H$ such that for $u, v \in V_G$, the vertex u is adjacent to the vertex v in graph G if and only if $\phi(u)$ is adjacent to $\phi(v)$ in graph H. Implicitly, there is also an edge bijection $E_G \to E_H$ such that $uv \mapsto \phi(u)\phi(v)$.

D50: An ***isomorphism between two general graphs*** G and H is a pair of bijections $\phi_V : V_G \to V_H$ and $\phi_E : E_G \to E_H$ such that for every pair of vertices $u, v \in V_G$, the set of edges in E_G joining u and v is mapped bijectively to the set of edges in E_H joining the vertices $\phi(u)$ and $\phi(v)$.

D51: We say that G and H are ***isomorphic graphs*** and we write $G \cong H$ if there is an isomorphism $G \to H$.

D52: An ***adjacency matrix*** for a simple graph G whose vertices are explicitly ordered $v_1, v_2, \ldots, v_n$ is the $n \times n$ matrix A_G such that

$$A_G(i,j) = \begin{cases} 1 & \text{if } v_i \text{ and } v_j \text{ are adjacent} \\ 0 & \text{otherwise} \end{cases} \tag{1.1.1}$$

D53: A property associated with all graphs is an ***isomorphism invariant*** if it has the same value (or is the same) for any two isomorphic graphs.

EXAMPLES

E13: The two graphs in Figure 1.1.13 are isomorphic under the mapping

$$u_1 \mapsto v_1 \quad u_2 \mapsto v_1 \quad u_3 \mapsto v_4 \quad u_4 \mapsto v_3$$

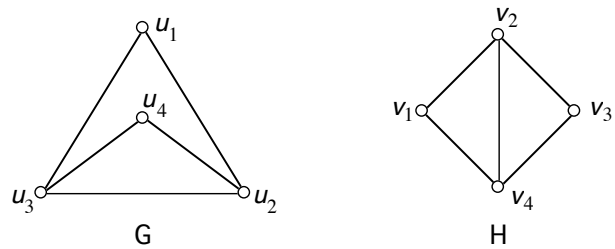


Figure 1.1.13: Two isomorphic graphs.

If one flips vertex u_4 of graph G downward to the bottom and rotates the figure a quarter-turn counterclockwise, then the resulting image of graph G "looks just like" graph H. Their adjacency matrices are:

$$A_G = \begin{array}{c} \\ u_1 \\ u_2 \\ u_3 \\ u_4 \end{array} \begin{array}{c} \begin{array}{cccc} u_1 & u_2 & u_3 & u_4 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \end{array} \qquad A_H = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

We observe that transposing rows u_3 and u_4 and also transposing columns u_3 and u_4 transforms the matrix A_G into matrix A_H.

E14: The two graphs in Figure 1.1.14 are isomorphic, even if the drawings look quite different. The vertex-labels indicate an isomorphism.

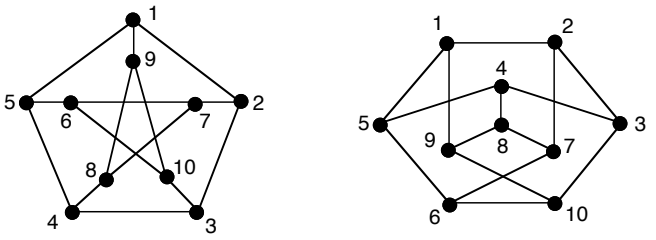


Figure 1.1.14: Two isomorphic graphs that look quite different.

E15: Figure 1.1.15 shows two non-isomorphic graphs with identical degree sequences. (It is easy to show that connectedness is an isomorphism invariant.)

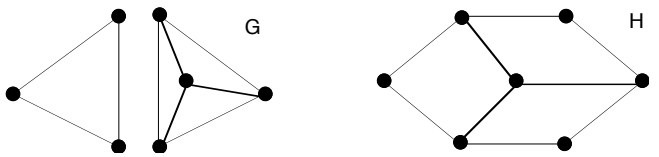


Figure 1.1.15: Two graphs whose degree sequences are both $\langle 2, 2, 2, 3, 3, 3, 3 \rangle$.

FACTS

F7: Considering all possible bijections of the vertex-sets of two n-vertex graphs requires $O(n!)$ steps.

F8: Although some fast heuristics are known (see §2.2), there is no known polynomial-time algorithm for testing graph isomorphism.

F9: The number of vertices, the number of edges, and the degree sequence are all isomorphism invariants. On the other hand, having the same values for all three of these invariants does not imply that two graphs are isomorphic, as illustrated by Example 15.

F10: Each row sum (and column sum) in an adjacency matrix equals the degree of the corresponding vertex.

Automorphisms

The notion of symmetry in a graph is formalized in terms of isomorphisms of the graph to itself.

DEFINITIONS

D54: A ***graph automorphism*** is an isomorphism of the graph to itself.

D55: The ***orbit of a vertex*** u of a graph G is the set of all vertices $v \in V_G$ such that there is an automorphism ϕ such that $\phi(u) = v$.

D56: The ***orbit of an edge*** d of a graph G is the set of all edges $e \in E_G$ such that there is an automorphism ϕ such that $\phi(d) = e$.

D57: A graph is ***vertex-transitive*** if all the vertices are in the same orbit.

D58: A graph is ***edge-transitive*** if all the edges are in the same orbit.

FACTS

F11: The vertex orbits partition the vertex-set of a graph.

F12: The edge orbits partition the edge-set of a graph.

EXAMPLE

E16: For the graph on the left in Figure 1.1.16, the vertex orbits are $\{u_1, u_4\}$ and $\{u_2, u_3\}$, and the edge orbits are $\{u_1u_2, u_1u_3, u_2u_4, u_3u_4\}$ and $\{u_2u_3\}$. The graph on the right is vertex-transitive and edge-transitive.

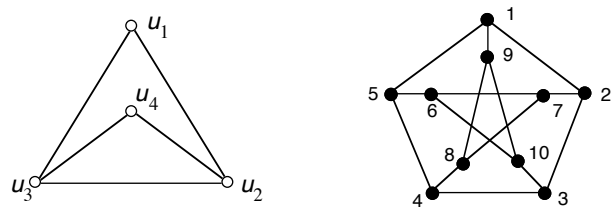


Figure 1.1.16: The graph $K_4 - e$ and the Petersen graph.

Subgraphs

DEFINITIONS

D59: A subgraph of a graph G is a graph H such that $V_H \subset V_G$ and $E_H \subset E_G$. (Usually, any graph isomorphic to a subgraph of G is also said to be a subgraph of G.)

D60: In a graph G, the ***induced subgraph*** on a set of vertices $W = \{w_1, \ldots, w_k\}$, denoted $G(W)$, has W as its vertex-set, and it contains every edge of G whose endpoints are in W. That is,

$$V(G(W)) = W \text{ and } E(G(W)) = \{e \in E(G) \mid \text{the endpoints of edge } e \text{ are in } W\}$$

D61: A subgraph H of a graph G is a ***spanning subgraph*** if $V(H) = V(G)$. (Also, if H is isomorphic to a spanning subgraph of G, we may say that H spans G.)

D62: A ***component*** of a graph G is a connected subgraph H such that no subgraph of G that properly contains H is connected. In other words, a component is a *maximal* connected subgraph.

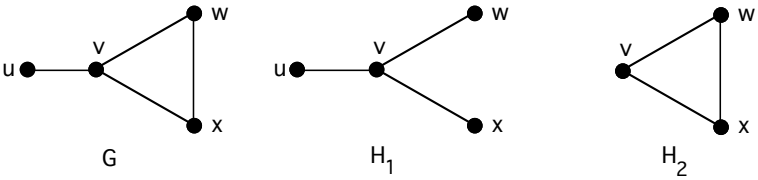


Figure 1.1.17: A spanning subgraph H_1 and an induced subgraph H_2.

EXAMPLE

E17: For the graph G in Figure 1.1.17, H_1 is a spanning subgraph but not an induced subgraph, and H_2 is an induced subgraph but not a spanning subgraph.

FACTS

F13: Let $\phi : G \to H$ be a graph isomorphism, and let J be a subgraph of G. Then the restriction of ϕ to the subgraph J is an isomorphism onto its image $\phi(J)$.

F14: If a graph J is a subgraph of a graph G but not a subgraph of a graph H, then $G \ncong H$. This is a corollary of Fact F13.

Graph Operations

The operations of adding and deleting vertices and edges of a graph are regarded as *primary operations*, because they are the foundation for other operations, which may be called *secondary operations*.

DEFINITIONS

D63: The operation of ***adding the vertex*** u to a graph $G = (V, E)$, such that $u \notin V$, yields a new graph with vertex-set $V \cup \{u\}$ and edge-set E, which is denoted $G \cup \{u\}$. (The new vertex u has no neighbors.)

D64: The operation of ***deleting the vertex*** u from a graph $G = (V, E)$ not only removes the vertex u but also removes every edge of which u is an endpoint. The resulting graph is denoted $G - u$.

D65: The operation of ***adding an edge*** d (or uv) to a graph $G = (V, E)$ joining the vertices u and v yields a new graph with vertex-set V and edge-set $E \cup \{d\}$ (or $E \cup \{uv\}$), which is denoted $G \cup \{d\}$ (or $G \cup \{uv\}$).

D66: The operation of ***deleting an edge*** d (or uv) from a graph $G = (V, E)$ removes only that edge. The resulting graph is denoted $G - d$ (or $G - uv$).

D67: A ***cut-vertex*** (or ***cutpoint***) is a vertex whose removal increases the number of components.

D68: A ***cut-edge*** is an edge whose removal increases the number of components.

D69: The ***edge-complement*** of a simple graph G is the graph $\overline{G}$ (alternatively denoted G^c) that has the same vertex-set as G, such that uv is an edge of $\overline{G}$ if and only if it is *not* an edge of G.

D70: The ***join*** (or ***suspension***) of two graphs G and H is denoted by $G + H$. It has the following vertex-set and edge-set:

$$V(G + H) = V(G) \cup V(H)$$
$$E(G + H) = E(G) \cup E(H) \cup \{uv \mid u \in V(G) \text{ and } v \in V(H)\}$$

D71: The ***cartesian product*** (or ***product***) of two graphs G and H is denoted by $G \times H$. Its vertex-set and edge-set are as follows:

$$V(G \times H) = V(G) \times V(H)$$
$$E(G \times H) = E(G) \times V(H) \cup V(G) \times E(H)$$

The endpoints of the edge $(d, v) \in E(G) \times V(H)$ are the vertices (x, v) and (y, v), where x and y are the endpoints of edge $d \in E(G)$. The endpoints of the edge $(u, e) \in V(G) \times E(H)$ are the vertices (u, s) and (u, t), where s and t are the endpoints of edge $e \in E(H)$.

D72: The ***graph union*** of two graphs G and H is the graph $G \cup H$ whose vertex-set and edge-set are the disjoint unions, respectively, of the vertex-sets and the edge-sets of G and H.

D73: The **m-*fold self-union*** mG is the iterated disjoint union $G \cup \cdots \cup G$ of m copies of the graph G.

EXAMPLES

E18: Figure 1.1.18 illustrates the operation of edge-complementation.

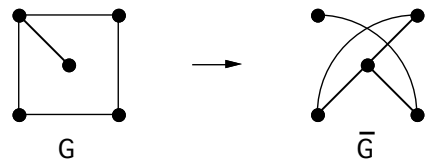


Figure 1.1.18: Edge-complementation.

E19: Figure 1.1.19 illustrates the join operation.

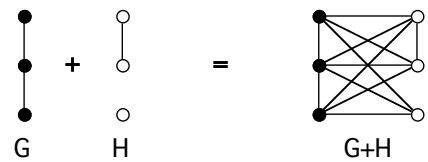


Figure 1.1.19: Join operation.

E20: In Figure 1.1.18, the vertex in the upper left corner of the drawing of the graph G is a cut-vertex, and the edge from that vertex to the center vertex is a cut-edge.

E21: Figure 1.1.20 illustrates the product operation.

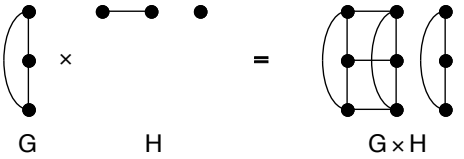


Figure 1.1.20: Cartesian product.

1.1.4 Trees

Trees are important to the structural understanding of graphs and to the algorithmics of information processing, and they play a central role in the design and analysis of connected networks. A standard characterization theorem for trees appears in Chapter 2.

Acyclic Graphs

DEFINITIONS

D74: A ***tree*** is a connected graph with no cycles (i.e., *acyclic*).

D75: A ***forest*** is a (not necessarily connected) graph with no cycles.

D76: A ***central vertex*** in a graph is a vertex whose eccentricity equals the radius of the graph.

D77: The ***center*** of a graph is the subgraph induced on its set of central vertices.

TERMINOLOGY NOTE: Classically (see Chapter 3), the words *center* and *bicenter* were used to mean the set of central vertices of a tree, when there was only one vertex or two vertices, respectively. (See Fact F15 below.)

EXAMPLE

E22: The graph on the left in Figure 1.1.21 is a tree; the other two graphs are not.

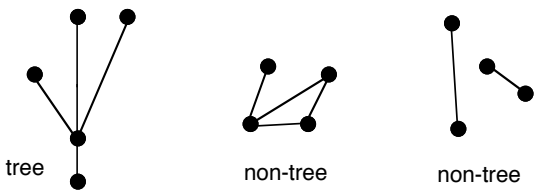


Figure 1.1.21: A tree and two non-trees.

FACT

F15: The center of a tree is isomorphic to K_1 or to K_2. (See §1.3 for information about the historical context of this fact.)

Trees as Subgraphs

Several different problem-solving algorithms involve growing a tree within a graph, one edge and one vertex at a time. All these techniques are refinements and extensions of the same basic tree-growing scheme given in this section.

DEFINITIONS

TERMINOLOGY: For a given tree T in a graph G, the edges and vertices of T are called ***tree edges*** and ***tree vertices***, and the edges and vertices of G that are not in T are called ***non-tree edges*** and ***non-tree vertices***.

D78: A ***frontier edge*** for a given tree T in a graph is a non-tree edge with one endpoint in T and one endpoint not in T.

D79: A ***spanning tree*** of a graph G is a spanning subgraph of G that is a tree.

EXAMPLE

E23: For the graph in Figure 1.1.22, the tree edges of a tree T are drawn in bold. The tree vertices are black, and the non-tree vertices are white. The frontier edges for T, appearing as dashed lines, are edges a, b, c, and d. The plain edges are the non-tree edges that are not frontier edges for T.

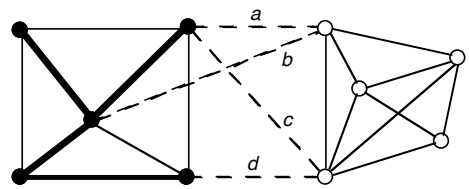


Figure 1.1.22: A tree with frontier edges a, b, c, and d.

Observe that when any one of the frontier edges in Figure 1.1.22 is added to the tree T, the resulting subgraph is still a tree. This property holds in general, and applying it iteratively forms the core of the tree-growing scheme of this section.

FACT

F16: Let T be a tree in a graph G, and let e be a frontier edge for T. Then the subgraph of G formed by adding edge e to tree T is a tree. (Formally, adding frontier edge e to a tree involves adding a new vertex to current tree T, i.e., its non-tree endpoint.)

Basic Tree-Growing Algorithm

The basic tree-growing scheme uses vertex labels to keep track of the order in which vertices are added to the tree.

TERMINOLOGY NOTE: A ***standard (0-based) vertex-labeling*** of an n-vertex graph is a one-to-one assignment of the integers $0, 1, \ldots, n-1$ to the vertices of that graph.

Algorithm 1.1.1: Basic Tree-Growing with Vertex Labels

Input: a graph G and a starting vertex $v \in V_G$.
Output: a spanning tree T of $C_G(v)$ and a standard vertex-labeling of $C_G(v)$.

```
Initialize tree T as vertex v.
Write label 0 on vertex v.
Initialize label counter i := 1
While tree T does not yet span component C_G(v)
    Choose a frontier edge e for tree T.
    Let w be the endpoint of edge e that lies outside of T.
    Add edge e and vertex w to tree T.
    Write label i on vertex w.
    i := i + 1
Return tree T and vertex-labeling of C_G(v).
```

REMARK

R4: Uniqueness of the Output Tree from Tree-Growing

Without a rule for choosing a frontier edge (including a way to break ties), the output tree from Algorithm 1.1.1 would not be unique (in which case, many computer scientists would hesitate to use the term *algorithm*). The uniqueness of the output depends on some *default priority* based on the ordering of the edges (and vertices) in the data structure chosen to implement the algorithm. The default priority is used whenever no other rule is given and as a way of breaking ties left from other rules.

FACTS

F17: If an execution of the basic tree-growing algorithm starts at vertex v of a graph G, then the subgraph consisting of the labeled vertices and tree edges is a spanning tree of the component $C_G(v)$.

F18: A graph is connected if and only if the basic tree-growing algorithm labels all its vertices.

Prioritizing the Edge Selection

The edge-prioritized tree-growing algorithm, Algorithm 1.1.2, is a refinement of basic tree-growing.

Algorithm 1.1.2: Edge-Prioritized Tree-Growing

Input: a connected graph G, a starting vertex $v \in V_G$,
and a rule for prioritizing frontier edges.
Output: a spanning tree T and a standard vertex-labeling of V_G.

Initialize tree T as vertex v.
Initialize the set of frontier edges for tree T as empty.
Write label 0 on vertex v.
Initialize label counter $i := 1$
While tree T does not yet span G
    Update the set of frontier edges for T.
    Let e be the frontier edge for T of highest priority.
    Let w be the unlabeled endpoint of edge e.
    Add edge e (and vertex w) to tree T.
    Write label i on vertex w.
    $i := i + 1$
Return tree T with its vertex-labeling.

FACT

F19: Different rules for prioritizing the frontier edges give rise to different spanning trees: the *depth-first search* tree (*last-in-first-out* priority), the *breadth-first search* tree (*first-in-first-out* priority), the *Prim tree* (*least-cost* priority), and the *Dijkstra tree* (*closest-to-root* priority). See §10.1

References

[Be85] C. Berge, *Graphs*, North-Holland, 1985.

[Bo98] B. Bollobás, *Modern Graph Theory*, Springer, 1998.

[ChLeZh10] G. Chartrand, L. Lesniak, and P. Zhang, *Graphs and Digraphs*, Fifth Edition, CRC Press, 2010.

[GrYe06] J. L. Gross and J. Yellen, *Graph Theory and Its Applications*, Second Edition, CRC Press, 2006.

[Ha94] F. Harary, *Graph Theory*, Perseus reprint, 1994. (First Edition, Addison-Wesley, 1969.)

[ThSw92] K. Thulasiraman and M. N. S. Swamy, *Graphs: Theory and Algorithms*, John Wiley & Sons, 1992.

[Tu00] W. T. Tutte, *Graph Theory*, Cambridge University Press, 2000.

[We01] D. B. West, *Introduction to Graph Theory*, Second Edition, Prentice-Hall, 2001. (First Edition, 1996.)

Section 1.2
Families of Graphs and Digraphs

Lowell W. Beineke, Purdue University at Fort Wayne

INTRODUCTION

Whenever a property of graphs is defined, a family of graphs — those with that property — results. Consequently, we focus on basic families. Along with the definitions of families, we include characterizations where appropriate. [ReWi98] offers a detailed catalog of the members of various graph and digraph families.

1.2.1 Building Blocks

Some simple graphs have as few edges or as many as possible for a given number of vertices. Some multigraphs and general graphs have as few vertices as possible for a given number of edges.

DEFINITIONS

D1: A simple graph is a ***complete graph*** if every pair of vertices is joined by an edge. The complete graph with n vertices is denoted K_n.

D2: The ***empty graph*** $\overline{K_n}$ is defined to be the graph with n vertices and no edges.

D3: The ***null graph*** K_0 is the graph with no vertices or edges.

D4: The ***trivial graph*** K_1 is the graph with one vertex and no edges.

D5: The ***bouquet*** B_n is the general graph with one vertex and n self-loops.

D6: The ***dipole*** D_n is the multigraph with two vertices and n edges.

D7: A simple digraph is a ***complete digraph*** if between every pair of vertices there is an arc in each direction. The complete digraph with n vertices is denoted $\overleftrightarrow{K}_n$.

D8: The ***path graph*** P_n is the n-vertex graph with $n-1$ edges, all on a single open path. (Quite commonly elsewhere, the subscript of the notation P_n denotes the number of edges.)

D9: The ***cycle graph*** C_n is the n-vertex graph with n edges, all on a single cycle.

REMARKS

R1: Although the empty graph may seem to some a "pointless" concept, it is the default initial value in computer representations of graph-valued variables.

R2: Whereas a "path" and a "cycle" are alternating sequences of vertices and edges, a "path graph" and a "cycle graph" are kinds of graphs.

EXAMPLES

E1: Figure 1.2.1 shows the complete graph K_4 and the complete digraph $\overleftrightarrow{K}_4$.

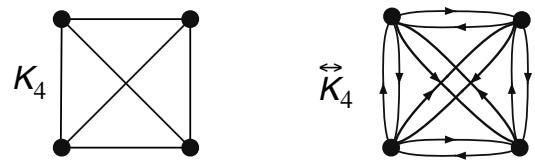


Figure 1.2.1: A complete graph and a complete digraph.

E2: Figure 1.2.2 shows a path graph and a cycle graph.

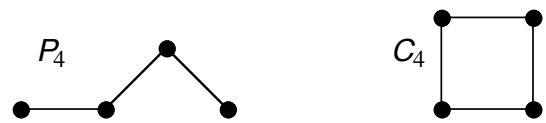


Figure 1.2.2: A path graph and a cycle graph.

1.2.2 Symmetry

Graphs with various kinds of symmetry are of particular interest.

Local Symmetry: Regularity

Regularity of a graph is an elementary form of local symmetry.

DEFINITIONS

D10: A graph is ***regular*** if every vertex is of the same degree.

- It is **k-*regular*** if every vertex is of degree k.

D11: A **k-*factor*** of a graph G is a k-regular spanning subgraph.

FACT

F1: All *vertex-transitive* graphs (see §1.1) are regular.

EXAMPLES

E3: For $k = 0, 1, 2, 3$, there is exactly one k-regular simple graph with 4 vertices.

E4: The only regular simple graphs with 5 vertices are the empty graph $\overline{K_5}$ (degree 0), the cycle graph C_5 (degree 2), and the complete graph K_5 (degree 4).

E5: [ReWi98] There are exactly two 3-regular simple graphs with 6 vertices.

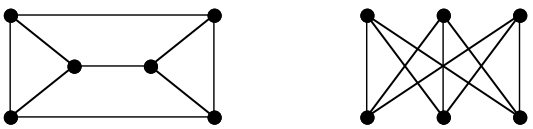


Figure 1.2.3: The two 3-regular simple graphs with 6 vertices.

E6: The disjoint union of the complete graphs K_3 and K_4 is a 2-regular simple 7-vertex graph that is not vertex-transitive. Its edge-complement is a 4-regular connected simple 7-vertex graph that is not vertex-transitive.

E7: Of the five 3-regular connected simple graphs with 8 vertices, two are vertex-transitive.

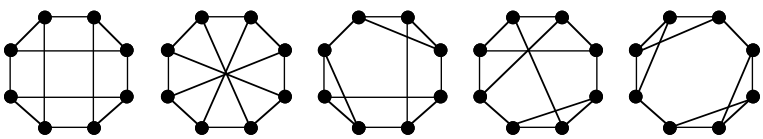


Figure 1.2.4: The five 3-regular connected simple graphs with 8 vertices.

Global Symmetry: Vertex-Transitivity

Often vertex-transitivity arises from algebra or geometry. See §6.2 for further discussion of Cayley graphs and circulant graphs.

DEFINITIONS

D12: The ***Cayley graph*** $C(\mathcal{A}, X)$ for a group $\mathcal{A}$ with generating set X has the elements of $\mathcal{A}$ as vertices and has an edge directed from a to ax for every $a \in \mathcal{A}$ and $x \in X$. We assume that vertices are labeled by elements of $\mathcal{A}$ and that edges are labeled by elements of X.

- We note that an involution x gives rise to a pair of oppositely directed edges between a and ax, for each $a \in \mathcal{A}$; sometimes we identify each such pair of directed edges to a single undirected edge labeled x.

D13: A ***circulant graph*** $\mathrm{Circ}(n; X)$ is defined for a positive integer n and a subset X of the integers $1, 2, \ldots, \lfloor \frac{n}{2} \rfloor$, called the ***connections***.

- The vertex set is $\mathbb{Z}_n$, the integers modulo n.
- There is an edge joining two vertices j and k if and only if the difference $|j - k|$ is in the set X. A circulant graph is a special case of a Cayley graph; an involution in the connection set gives rise to a single edge.

D14: The 1-***skeleton*** (often in graph theory, the ***skeleton***) of a k-complex K is the graph consisting of the vertices and the edges of K.

D15: The **d-*hypercube graph*** Q_d (or **d-*cube graph***) is the 1-skeleton of the d-dimensional hypercube $\{(x_1, \ldots, x_n) \mid 0 \leq x_j \leq 1\}$. This graph has 2^d vertices and is regular of degree d.

D16: The **d-*octahedral graph*** $\mathcal{O}_d$ is defined recursively:

$$\mathcal{O}_d = \begin{cases} \overline{K_2} & \text{if } n = 1 \\ \mathcal{O}_{d-1} + \overline{K_2} & \text{if } n \geq 2 \end{cases}$$

D17: The ***Petersen graph*** is the 10-vertex 3-regular graph depicted in Figure 1.2.5.

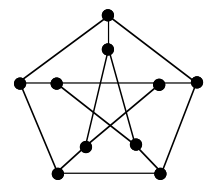


Figure 1.2.5: The Petersen graph.

EXAMPLES

E8: The n-***simplex*** is the convex hull of $n + 1$ affinely independent points in n-dimensional space. Its 1-skeleton is isomorphic to the complete graph K_n.

E9: A ***Platonic graph*** is the 1-skeleton of one of the five Platonic solids: the tetrahedron, the cube, the octahedron, the dodecahedron, and the icosahedron.

E10: The Petersen graph is vertex-transitive, since there is an automorphism that swaps the pentagram (i.e., the star) with the pentagon. It is not a Cayley graph of either of the two groups of order 10, i.e., of the cyclic group $\mathbb{Z}_{10}$ or of the dihedral group $\mathbb{D}_5$, and thus, not a Cayley graph.

E11: The octahedral graph $\mathcal{O}_d$ is isomorphic to $\overline{dK_2}$.

FACTS

F2: [Hypercube Characterization Theorem] The graph whose vertices are the binary sequences of length d in which two vertices are adjacent if their sequences differ in exactly one place is isomorphic to Q_d.

F3: We can construct the d-dimensional hypercube Q_d recursively, using the cartesian product operation:

$$Q_d = \begin{cases} K_1 & \text{if } d = 0 \\ Q_{d-1} \times K_2 & \text{if } d \geq 1 \end{cases}$$

1.2.3 Integer-Valued Invariants

Some of the most useful graph properties are provided by integer-valued invariants of isomorphism type. Such invariants partition all graphs into an infinite list of subclasses. Often the subclasses with low invariant values are of special interest.

Cycle Rank

The connected graphs of *cycle rank* 0 are of great special interest, since they are the *trees* (see §1.1).

DEFINITION

D18: The ***cycle rank*** of a connected graph $G = (V, E)$ is the number $|E|-|V|+1$. (See §6.4 for an interpretation of cycle rank as the rank of a vector space.) More generally, for a graph G with $c(G)$ components, the cycle rank is the number $|E(G)| - |V(G)| + c(G)$.

EXAMPLE

E12: The connected graphs of cycle rank 0 are the trees. The smallest trees are shown in Figure 1.2.6.

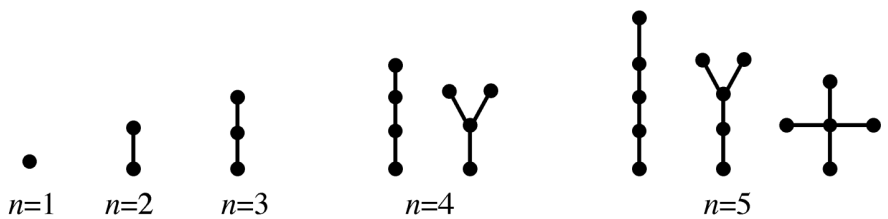


Figure 1.2.6: The trees with up to five vertices.

FACTS

F4: [Tree Characterization Theorem] The following statements are equivalent for a graph T with n vertices (e.g., see [GrYe06, Theorem 3.1.8]):

- T is a tree (that is, G is connected and has no cycles).
- T is connected and has $n-1$ edges.
- T has no cycles and has $n-1$ edges.
- Any two vertices of T are connected by exactly one path.

F5: [Inductive (Recursive) Definition of Trees] Let $\mathcal{T}$ be the family of graphs defined as follows:

(i) $K_1 \in \mathcal{T}$.

(ii) If $T \in \mathcal{T}$ and T' can be obtained by adding a new vertex and joining it to a vertex of T, then $T' \in \mathcal{T}$.

Then $\mathcal{T}$ is the family of all trees.
(Several more classes of recursively defined graphs are presented in §2.4.)

F6: The cycle rank of a graph is the sum of the cycle ranks of its components.

F7: A forest is a graph such that every component is a tree.

Chromatic Number and k-Partite Graphs

In a proper coloring of a graph, no two vertices with the same color are adjacent, and thus, every edges joins vertices in different color classes. The graphs with a proper 2-coloring are of special interest. Graph coloring is covered extensively in §5.1 and §5.2.

DEFINITIONS

D19: A simple graph or multigraph is ***bipartite*** if its vertices can be partitioned into two sets (called ***partite sets***) in such a way that no edge joins two vertices in the same set. (For technical reasons, this includes the graph K_1 in this definition.) If r and s are the orders of the partite sets, then the graph is said to be an ***r-by-s bipartite graph***.

D20: A ***complete bipartite graph*** is a simple bipartite graph in which each vertex in one partite set is adjacent to all the vertices in the other partite set. If the two partite sets have cardinalities r and s, then this graph is denoted $K_{r,s}$.

D21: A graph is ***k-partite*** if its vertices can be partitioned into k sets (called ***partite sets***) in such a way that no edge joins two vertices in the same set.

D22: A ***complete k-partite graph*** is a simple k-partite graph in which two vertices are adjacent if and only if they are in different partite sets. All such graphs are called ***complete multipartite graphs***. If the k partite sets have orders $n_1, n_2, \ldots, n_k$, then the graph is denoted $K_{n_1,n_2,\ldots,n_k}$, and if each partite set has order r, then $K_{k(r)}$.

EXAMPLES

E13: Every tree is bipartite.

E14: Every cycle with an even number of vertices is bipartite, and no cycle with an odd number is bipartite.

E15: The complete d-partite graph $K_{d(2)}$ is isomorphic to the d-octahedral graph $\mathcal{O}_d$. The first four complete d-partite graphs are shown in Figure 1.2.7.

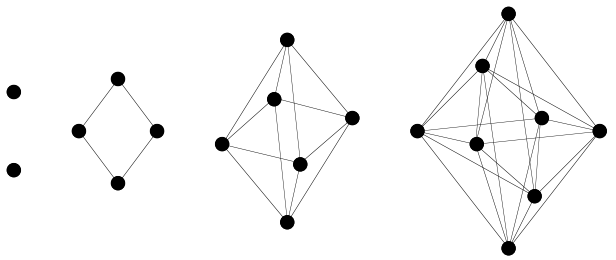


Figure 1.2.7: The complete d-partite graphs $K_{d(2)}$, for $d = 1, \ldots, 4$.

FACTS

F8: [Bipartite Graph Characterization Theorem] A graph is bipartite if and only if the length of each of its cycles is even (e.g., see [GrYe06, Theorem 1.5.4]).

F9: A graph is k-colorable if and only if it is k-partite.

F10: For $k \geq 3$, the problem of deciding whether a graph is k-partite is NP-complete.

K-Connectivity and K-Edge-Connectivity

Graphs can be categorized according to their *connectivity* and their *edge-connectivity*. There are analogues for strong connectedness in digraphs. See §4.1 and §4.7 for extensive coverage of connectivity.

DEFINITIONS

D23: The ***(vertex-)connectivity*** of a graph G, denoted $\kappa_v(G)$, is the minimum number of vertices whose removal from G leaves a non-connected or trivial graph.

D24: The ***edge-connectivity*** of a nontrivial graph G, denoted $\kappa_e(G)$ is the minimum number of edges whose removal from G results on a non-connected graph.

NOTATION: The subscripted "G" is often suppressed when the graph G is understood. Elsewhere, the notation κ and λ are used instead of κ_v and κ_e, respectively.

D25: A graph G with connectivity $\kappa_v \geq k \geq 1$ is called ***k-connected***. Equivalently, G is k-connected if the removal of $k-1$ or fewer vertices leaves neither a non-connected graph nor a trivial one.

D26: A graph G with edge-connectivity $\kappa_e \geq k \geq 1$ is called ***k-edge-connected***. That is, the removal of $k-1$ or fewer edges from a k-edge-connected graph results in a connected graph.

D27: A digraph is ***strongly k-connected*** (or ***k-strong***) if the result of removing any set of fewer than k vertices is strongly connected and non-trivial.

D28: A digraph is ***strongly k-arc-connected*** (or ***k-arc-strong***) if the result of removing any set of fewer than k arcs is strongly connected and non-trivial.

Minimum Genus

Graphs can be categorized according to their topological properties.

DEFINITIONS

D29: The ***minimum genus*** (or simply the ***genus***) of a connected graph G is the smallest number g such that G can be drawn on the orientable surface S_g (see §7.1) without any edge-crossings.

D30: A graph of genus 0 is ***planar***.

1.2.4 Criterion Qualification

A graph family is also specified as the set of all graphs or digraphs that match a stated criterion, e.g., traversibility and various forms of minimality and maximality.

DEFINITIONS

D31: A graph is ***eulerian*** if it has a closed walk that contains every edge exactly once. (See §1.3 for the history of eulerian graphs and §4.2 for an extensive discussion.)

D32: A graph is ***hamiltonian*** if it has a spanning cycle. (See §1.3 for the history of hamiltonian graphs and §4.5 for an extensive discussion.)

D33: A k-chromatic graph is ***critically k-chromatic*** if its chromatic number would decrease if any edge were removed. (See §5.1.)

D34: A k-connected graph is ***critically k-connected*** if its connectivity would decrease if any vertex were removed. (See §4.1.)

D35: A k-edge-connected graph is ***critically k-edge-connected*** if its edge-connectivity would decrease if any edge were removed. (See §4.1.)

D36: A ***tournament*** is a digraph in which there is exactly one arc between each pair of vertices. (See §3.3.)

D37: The ***line graph*** $L(G)$ of a graph G has the edges of G as its vertices; two vertices of $L(G)$ are adjacent if the edges in G to which they correspond have a common vertex. A graph and its line graph are illustrated in Figure 1.2.8. Also, a graph H is said to be a ***line graph*** if there exists a graph G such that H is isomorphic to $L(G)$.

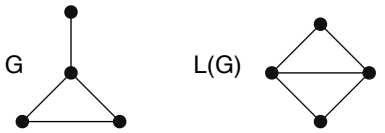


Figure 1.2.8: A graph and its line graph.

FACTS

F11: [Line Graph Characterization] The following statements are equivalent:

- G is a line graph.
- [Kr43] The edges of G can be partitioned into complete subgraphs in such a way that no vertex is in more than two.
- [Be70] None of the nine graphs in Figure 1.2.9 is an induced subgraph of G.

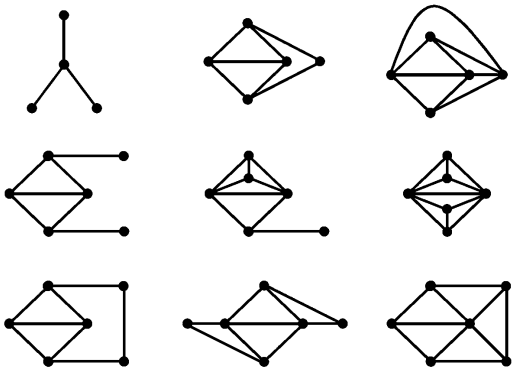


Figure 1.2.9: The nine forbidden induced subgraphs.

F12: A strongly connected tournament contains a directed spanning cycle.

EXAMPLE

E16: The eight tournaments with one to four vertices are shown in Figure 1.2.10.

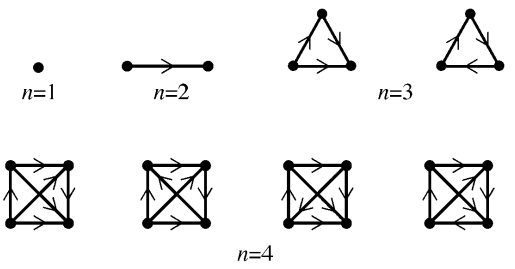


Figure 1.2.10: All tournaments with one to four vertices.

References

[Be70] L. W. Beineke, Characterizations of derived graphs, *J. Combin. Theory* 9 (1970), 129–135.

[GrYe06] J. L. Gross and J. Yellen, *Graph Theory and Its Applications*, Second Edition, CRC Press, 2006.

[Kr43] J. Krausz, Dèmonstation nouvelle d'une théorème de Whitney sur les réseaux, *Mat. Fiz. Lapok* 50 (1943), 75–89.

[ReWi98] R. C. Read and R. J. Wilson, *An Atlas of Graphs*, Oxford University Press, 1998.

Section 1.3
History of Graph Theory

Robin J. Wilson, Pembroke College, Oxford University, UK

INTRODUCTION

Although the first mention of a graph was not until 1878, graph-theoretical ideas can be traced back to 1735 when Leonhard Euler (1707–83) presented his solution of the Königsberg bridges problem. This chapter summarizes some important strands in the development of graph theory since that time. Further information can be found in [BiLlWi98] or [Wi99].

1.3.1 Traversability

The origins of graph theory can be traced back to Euler's work on the Königsberg bridges problem (1735), which subsequently led to the concept of an *eulerian graph*. The study of cycles on polyhedra by the Revd. Thomas Penyngton Kirkman (1806–95) and Sir William Rowan Hamilton (1805–65) led to the concept of a *Hamiltonian graph*.

The Königsberg Bridges Problem

The *Königsberg bridges problem*, pictured in Figure 1.3.1, asks whether there is a continuous walk that crosses each of the seven bridges of Königsberg exactly once — and if so, whether a closed walk can be found. See §4.2 for more extensive discussion of issues concerning *eulerian graphs*.

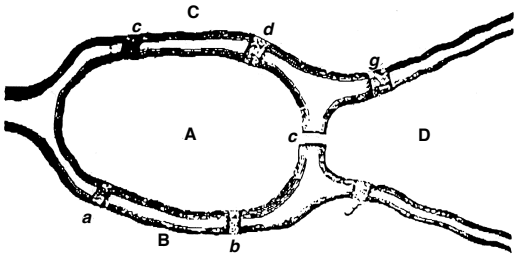


Figure 1.3.1: The seven bridges of Königsberg.

FACTS [BiLlWi98, Chapter 1]

F1: On 26 August 1735 Leonhard Euler gave a lecture on "The solution of a problem relating to the geometry of position" to the Academy of Sciences of St. Petersburg, Russia, proving that there is no such continuous walk across the seven bridges.

F2: In 1736, Euler communicated his solution to several other mathematicians, outlining his views on the nature of the problem and on its situation in the geometry of position [HoWi04].

F3: Euler [Eu:1736] sent his solution of the problem to the *Commentarii Academii Scientiarum Imperialis Petropolitanae* under the title "Solutio problematis ad geometriam ad geometriam situs pertinentis". Although dated 1736, it did not appear until 1741, and was later republished in the new edition of the *Commentarii* in 1752.

F4: Euler's paper is divided into 21 sections, of which 9 are on the Königsberg bridges problem, and the remainder are concerned with general arrangements of bridges and land areas.

F5: Euler did not draw a graph in order to solve the problem, but he reformulated the problem as one of trying to find a sequence of eight letters A, B, C, or D (the land areas) such that the pairs AB and AC are adjacent twice (corresponding to the two bridges between A and B and between A and C), and the pairs AD, BD, and CD are adjacent just once (corresponding to the remaining bridges). He showed by a counting argument that no such sequence exists, thereby proving that the Königsberg bridges problem has no solution.

F6: In discussing the general problem, Euler first observed that the number of bridges written next to the letters A, B, C, etc. together add up to twice the number of bridges. This is the first appearance of what some graph-theorists now call the "handshaking lemma", that the sum of the vertex-degrees in a graph is equal to twice the number of edges.

F7: Euler's main conclusions for the general situation were as follows:

- If there are more than two areas to which an odd number of bridges lead, then such a journey is impossible.
- If the number of bridges is odd for exactly two areas, then the journey is possible if it starts in either of these two areas.
- If, finally, there are no areas to which an odd number of bridges lead, then the required journey can be accomplished starting from any area.

These results correspond to the conditions under which a graph has an eulerian, or semi-eulerian, trail.

F8: Euler noted the converse result, that if the above conditions are satisfied, then a route is possible, and gave a heuristic reason why this should be so, but did not prove it. A valid demonstration did not appear until a related result was proved by C. Hierholzer [Hi:1873] in 1873.

Diagram-Tracing Puzzles

A related area of study was that of *diagram-tracing puzzles*, where one is required to draw a given diagram with the fewest possible number of connected strokes. Such puzzles can be traced back many hundreds of years – for example, there are some early African examples.

FACTS [BiLlWi98, Chapter 1]

F9: In 1809 L. Poinsot [Po:1809] wrote a memoir on polygons and polyhedra in which he posed the following problem:

> Given some points situated at random in space, it is required to arrange a single flexible thread uniting them two by two in all possible ways, so that the two ends of the thread join up and the total length is equal to the sum of all the mutual distances.

Poinsot noted that a solution is possible only when the number of points is odd, and gave a method for finding such an arrangement for each possible value. In modern terminology, the question is concerned with eulerian trails in complete graphs of odd order.

F10: Other diagram-tracing puzzles were posed and solved by T. Clausen [Cl:1844] and J. B. Listing [Li:1847]. The latter appeared in the book *Vorstudien zur Topologie*, the first place that the word "topology" appeared in print.

F11: In 1849, O. Terquem asked for the number of ways of laying out a complete ring of dominoes. This is essentially the problem of determining the number of eulerian tours in the complete graph K_7, and was solved by M. Reiss [Re:1871–3] and later by G. Tarry.

F12: The connection between the Königsberg bridges problem and diagram-tracing puzzles was not recognized until the end of the 19th century. It was pointed out by W. W. Rouse Ball [Ro:1892] in *Mathematical Recreations and Problems*. Rouse Ball seems to have been the first to use the graph in Figure 1.3.2 to solve the problem.

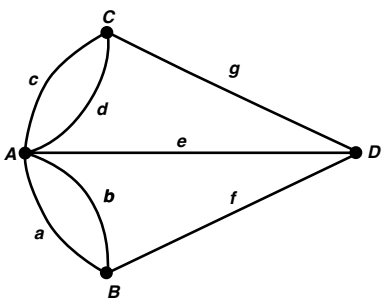


Figure 1.3.2: Rouse Ball's graph of the Königsberg bridges problem.

Hamiltonian Graphs

A type of graph problem that superficially resembles the eulerian problem is that of finding a cycle that passes just once through each vertex of a given graph. Because of Hamilton's influence, such graphs are now called *hamiltonian graphs* (see §4.5), instead of more justly being named after Kirkman, who, prior to Hamilton's consideration of the dodecahedron, as discussed below, considered the more general problem.

FACTS [BiLlWi98, Chapter 2]

F13: An early example of such a problem is the *knight's tour problem*, of finding a succession of knight's moves on a chessboard, visiting each of the 64 squares just once and returning to the starting point. This problem can be dated back many hundreds of years, and systematic solutions were given by Euler [Eu:1759], A.-T. Vandermonde [Va:1771], and others.

F14: In 1856 Kirkman [Ki:1856] wrote a paper investigating those polyhedra for which one can find a cycle passing through all the vertices just once. He proved that every polyhedron with even-sided faces and an odd number of vertices has no such cycle, and gave as an example the polyhedron obtained by "cutting in two the cell of a bee" (see Figure 1.3.3).

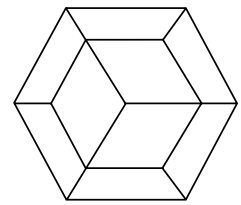


Figure 1.3.3: Kirkman's "cell of a bee" example.

F15: Arising from his work on non-commutative algebra, Hamilton considered cycles passing through all the vertices of a dodecahedron. He subsequently invented a game, called the icosian game (see Figure 1.3.4), in which the player was challenged to find such cycles on a solid dodecahedron, satisfying certain extra conditions.

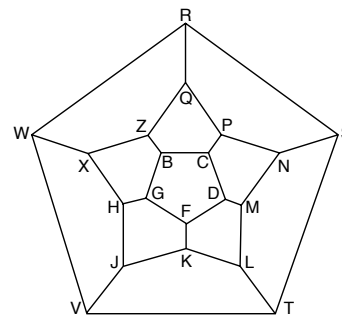


Figure 1.3.4: Hamilton's icosian game.

F16: In 1884, P. G. Tait asserted that every 3-valent polyhedron has a hamiltonian cycle. This assertion was subsequently disproved by W. T. Tutte [Tu46] in 1946 (see Figure 1.3.5).

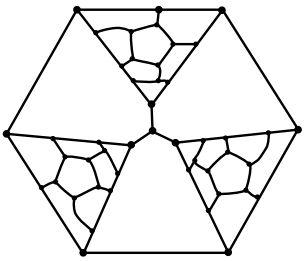


Figure 1.3.5: Tutte's 3-valent non-hamiltonian polyhedron.

F17: Sufficient conditions for a graph to be hamiltonian were later obtained by G. A. Dirac [Di52], O. Ore [Or60], J. A. Bondy and V. Chvátal [BoCh76], and others.

F18: Hamiltonian digraphs have also been investigated, by A. Ghouila-Houri (1960), H. Meyniel (1973), and others.

1.3.2 Trees

The concept of a tree, a connected graph without cycles, appeared implicitly in the work of Gustav Kirchhoff (1824–87), who employed graph-theoretical ideas in the calculation of currents in electrical networks. Later, trees were used by Arthur Cayley (1821–95), James Joseph Sylvester (1806–97), Georg Pólya (1887–1985), and others, in connection with the enumeration of certain chemical molecules.

Counting Trees

Enumeration techniques involving trees first arose in connection with a problem in the differential calculus, but they soon came to be fundamental tools in the counting of chemical molecules, as well as providing a fascinating topic of interest in their own right. Enumeration of various kinds of graphs is discussed in §6.3.

FACTS [BiLlWi98, Chapter 3] [PóRe87]

F19: While working on a problem inspired by some work of Sylvester on "differential transformation and the reversion of serieses", Cayley [Ca:1857] was led to the enumeration of rooted trees.

F20: Cayley's method was to take a rooted tree and remove its root, thereby obtaining a number of smaller rooted trees (see Figure 1.3.6).

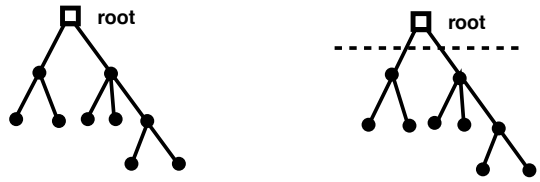


Figure 1.3.6: Splitting a rooted tree.

Letting A_n be the number of rooted trees with n branches, Cayley proved that the generating function

$$1 + A_1x + A_2x^2 + A_3x^3 + \dots$$

is equal to the product

$$(1-x)^{-1} \cdot (1-x^2)^{-A_1} \cdot (1-x^3)^{-A_2} \cdot \dots$$

Using this equality, he was able to calculate the first few numbers A_n, one at a time.

F21: Around 1870, Sylvester and C. Jordan independently defined the *center/bicenter* and the *centroid/bicentroid* of a tree.

F22: In 1874, Cayley [Ca:1874] found a method for solving the more difficult problem of counting unrooted trees. This method, which he applied to chemical molecules, consisted essentially of starting at the center or centroid of the tree or molecule and working outwards.

F23: In 1889, Cayley [Ca:1889] presented his n^{n-2} formula for the number of labeled trees with n vertices. He explained why the formula holds when $n = 6$, but he did not give a proof in general. The first accepted proof was given by H. Prüfer [Pr18]: his method was to establish a one-to-one correspondence between such labeled trees and sequences of length $n-2$ formed from the numbers $1, 2, \dots, n$.

F24: In a fundamental paper of 1937, Pólya [Pó37] combined the classical idea of a generating function with that of a permutation to obtain a powerful theorem that enabled him to enumerate certain types of configuration under the action of a group of symmetries. Some of Pólya's work was anticipated by J. H. Redfield [Re27], but Redfield's paper was obscurely written and had no influence on the development of the subject.

F25: Later results on the enumeration of trees were derived by R. Otter [Ot48] and others. The field of graphical enumeration (see [HaPa73]) was subsequently further developed by F. Harary [Ha55], R. C. Read [Re63], and others.

Chemical Trees

By 1850 it was already known that chemical elements combine in fixed proportions. Chemical formulas such as CH_4 (methane) and C_2H_5OH (ethanol) were known, but it was not understood how the elements combine to form such substances. Around this time, chemical ideas of valency began to be established, particularly when Alexander Crum Brown presented his graphic formulae for representing molecules. Figure 1.3.7 presents his representation of ethanol, the usual drawing, and the corresponding tree graph.

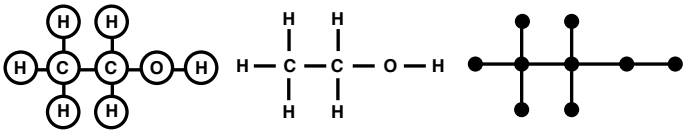


Figure 1.3.7: Representations of ethanol.

FACTS [BiLlWi98, Chapter 4]

F26: Crum Brown's graphic notation explained for the first time the phenomenon of isomerism, whereby there exist pairs of molecules (isomers) with the same chemical formula but different chemical properties. Figure 1.3.8 shows isomers with chemical formula C_4H_{10}.

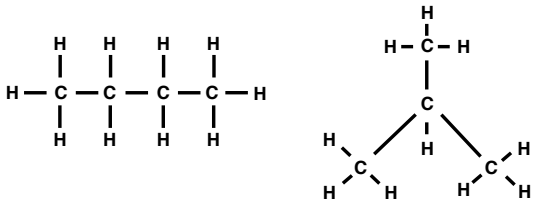


Figure 1.3.8: Two isomers: butane and isobutane.

F27: Cayley [Ca:1874] used tree-counting methods to enumerate paraffins (alkanes) with up to 11 carbon atoms, as well as various other families of molecules; the followiing table gives the number of isomers of alkanes for $n = 1, \ldots, 8$.

Formula	CH_4	C_2H_6	C_3H_8	C_4H_{10}	C_5H_{12}	C_6H_{14}	C_7H_{16}	C_8H_{18}
Number	1	1	1	2	3	5	9	18

F28: W. K. Clifford and Sylvester believed that a connection could be made between chemical atoms and binary quantics in invariant theory, a topic to which Cayley and Sylvester had made significant contributions. In 1878, Sylvester [Sy:1877–8] wrote a short note in *Nature* about this supposed connection, remarking that:

> Every invariant and covariant thus becomes expressible by a *graph* precisely identical with a Kekuléan diagram or chemicograph.

This was the first appearance of the word *graph* in the graph-theoretic sense.

F29: In 1878, Sylvester [Sy:1878] wrote a lengthy article on the graphic approach to chemical molecules and invariant theory in the first volume of the *American Journal of Mathematics*, which he had recently founded.

F30: Little progress was made on the enumeration of isomers until the 1920s and 1930s. A. C. Lunn and J. K. Senior [LuSe29] recognized the importance of permutation groups for this area, and Pólya's above-mentioned paper solved the counting problem for several families of molecules.

1.3.3 Topological Graphs

Euler's polyhedron formula [Eu:1750] was the foundation for topological graph theory, since it holds also for planar graphs. It was later extended to surfaces other than the sphere. In 1930, a fundamental characterization of graphs imbeddable in the sphere was given by Kazimierz Kuratowski (1896–1980), and recent work – notably by Neil Robertson, Paul Seymour, and others – has extended these results to the higher order surfaces.

Euler's Polyhedron Formula

The Greeks were familiar with the five regular solids, but there is no evidence that they knew the simple connection between the numbers V of vertices, E of edges, and F of faces of a polyhedron:

$$V - E + F = 2$$

In the 17th century, René Descartes studied polyhedra, and he obtained results from which Euler's formula could later be derived. However, since Descartes had no concept of an edge, he was unable to make such a deduction.

FACTS [BiLlWi98, Chapter 5] [Cr99] [BeWi09]

F31: The first appearance of the polyhedron formula appeared in a letter, dated 14 November 1750, from Euler to C. Goldbach. Denoting the number of faces, solid angles (vertices) and joints (edges) by $\mathcal{H}$, $\mathcal{S}$, and $\mathcal{A}$, he wrote:

- In every solid enclosed by plane faces the aggregate of the number of faces and the number of solid angles exceeds by two the number of edges, or $\mathcal{H} + \mathcal{S} = \mathcal{A} + 2$.

F32: Euler was unable to prove his formula. In 1752 he attempted a proof by dissection, but it was deficient. The first valid proof was given by A.-M. Legendre [Le:1794] in 1794, using metrical properties of spherical polygons.

F33: In 1813, A.-L. Cauchy [Ca:1813] obtained a proof of Euler's formula by stereographically projecting the polyhedron onto a plane and considering a triangulation of the resulting planar graph.

F34: Around the same time, S.-A.-J. Lhuilier [Lh:1811] gave a topological proof that there are only five regular convex polyhedra, and he anticipated the idea of duality by noting that four of them occur in reciprocal pairs. He also found three types of polyhedra for which Euler's formula fails – those with indentations in their faces, those

with an interior cavity, and ring-shaped polyhedra drawn on a torus (that is, polyhedra with a 'tunnel' through them). For such ring-shaped polyhedra, Lhuilier derived the formula

$$V - E + F = 0$$

and extended his discussion to prove that, if g is the number of tunnels in a surface on which a polyhedral map is drawn, then

$$V - E + F = 2 - 2g$$

The number g is now called the ***genus of the surface***, and the value of the quantity $2 - 2g$ is called the ***Euler characteristic***. (See §7.1.)

F35: In 1861–2, Listing [Li:1861–2] wrote *Der Census räumliche Complexe*, an extensive investigation into complexes, and studied how their topological properties affect the generalization above of Euler's formula. This work proved to be influential in the subsequent development of topology. In particular, H. Poincaré took up Listing's ideas in his papers of 1895–1904 that laid the foundations for algebraic topology.

F36: Poincaré's work was instantly successful, and it appeared in an article by M. Dehn and P. Heegaard [DeHe07] on analysis situs (topology) in the ten-volume *Encyklopädie der Mathematischen Wissenschaften*. His ideas were further developed by O. Veblen [Ve22] in a series of colloquium lectures on analysis situs for the American Mathematical Society in 1916.

Planar Graphs

The study of planar graphs originated in two recreational problems involving the complete graph K_5 and the complete bipartite graph $K_{3,3}$. These graphs (shown in Figure 1.3.9) are the main obstructions to planarity, as was subsequently demonstrated by Kuratowski.

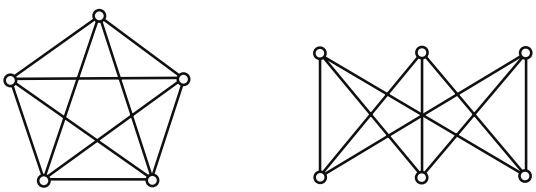


Figure 1.3.9: The Kuratowski graphs K_5 and $K_{3,3}$.

FACTS [BiLlWi98, Chapter 8]

F37: Around the year 1840, A. F. Möbius presented the following puzzle to his students:

> There was once a king with five sons. In his will he stated that after his death the sons should divide the kingdom into five regions so that the boundary of each region should have a frontier line in common with each of the other four regions. Can the terms of the will be satisfied?

This question asks whether one can draw five mutually neighboring regions in the plane. The connection with graph theory can be seen from its dual version, later formulated by H. Tietze:

> The king further stated that the sons should join the five capital cities of his kingdom by roads so that no two roads intersect. Can this be done?

In this dual formulation, the problem is that of deciding whether the graph K_5 is planar.

F38: An old problem, whose origins are obscure, is the *utilities problem*, or *gas–water–electricity problem*, mentioned by H. Dudeney [Du13] in the *Strand Magazine* of 1913:

> The puzzle is to lay on water, gas, and electricity, from W, G, and E, to each of the three houses, A, B, and C, without any pipe crossing another (see Figure 1.3.10).

This problem is that of deciding whether $K_{3,3}$ is planar.

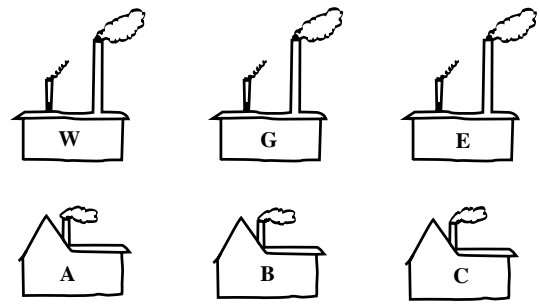


Figure 1.3.10: The gas–water–electricity problem.

F39: In 1930 Kuratowski [Ku30] published a celebrated paper proving that every nonplanar graph has a subgraph homeomorphic to K_5 or $K_{3,3}$; this result was obtained independently by O. Frink and P. A. Smith.

F40: In 1931 H. Whitney [Wh31] discovered an abstract definition of duality that is purely combinatorial and agrees with the geometrical definition of duality for planar graphs. He proved that, with this general definition of duality, a graph is planar if and only if it has an abstract dual. Related results were obtained by S. MacLane and others.

F41: In 1935 Whitney [Wh35] generalized the idea of independence in graphs and vector spaces to the concept of a matroid. The dual of a matroid extends and clarifies the duality of planar graphs, and Tutte [Tu59] used these ideas in the late 1950s to obtain a Kuratowski-type criterion for a matroid to arise from a graph (see §6.6).

Graphs on Higher Surfaces

A graph drawn without crossings on a plane corresponds (by stereographic projection) to a graph similarly drawn on the surface of a sphere. This leads to the idea of graphs drawn on surfaces other than the sphere. The initial work in this area was carried out, in

the context of coloring maps, by Percy Heawood (1861–1955) and Lothar Heffter (1862–1962) for orientable surfaces, and by Heinrich Tietze (1880–1964) for non-orientable surfaces, but the basic problems in the area were not solved until Gerhard Ringel and Ted Youngs solved the Heawood conjecture in the 1960s and Neil Robertson and Paul Seymour generalized Kuratowski's theorem to other surfaces in the 1980s.

FACTS [BiLlWi98, Chapter 7; Ri74]

F42: In 1890, Heawood [He:1890] presented an imbedding of the complete graph K_7 on a torus. He also derived a formula for the genus of a surface on which a given complete graph can be imbedded, but his attempted proof of this formula was deficient.

F43: In 1891, L. Heffter [He:1891] investigated the imbedding of complete graphs on orientable surfaces other than the sphere and the torus, and he proved that Heawood's formula is correct for orientable surfaces of low genus and certain other surfaces.

F44: In 1910, H. Tietze [Ti10] extended Heffter's considerations to certain non-orientable surfaces, such as the Möbius band and the projective plane, and stated a corresponding Heawood formula. He was unable to prove it for the Klein bottle, but this case was settled in 1934 by P. Franklin [Fr34], who found that it was an exception to the formula. In 1935, I. N. Kagno [Ka35] proved the formula for surfaces of non-orientable genus 3, 4, and 6.

F45: The Heawood formula for general non-orientable surfaces was proved in 1952 by Ringel. The proof for orientable surfaces proved to be much more difficult, involving 300 pages of consideration of 12 separate cases. Most of these were settled in the mid-1960s, and the proof was completed in 1968 by Ringel and Youngs [RiYo68], using W. Gustin's [Gu63] combinatorial inspiration in 1963 of a *current graph*. Since then, the transformation by J. L. Gross [Gr74] of numerous types of specialized combinatorial current graphs into a unified topological object, with its dualization to a *voltage graph* (see §7.4), has led to simpler solutions (see Gross and T. W. Tucker [GrTu74]).

F46: In a sequence of papers in the 1980s of great mathematical depth, Robertson and Seymour [RoSe85] proved that, for each orientable genus g, the set of "forbidden subgraphs" is finite (see §7.7). However, apart from the sphere, the number of forbidden subgraphs runs into hundreds, even for the torus. For non-orientable surfaces, there is a similar result, and in 1979 H. H. Glover, J. P. Huneke, and C. S. Wang [GlHuWa79] obtained a set of 103 forbidden subgraphs for the projective plane.

1.3.4 Graph Colorings

Early work on colorings concerned the coloring of the countries of a map and, in particular, the celebrated four-color problem. This was first posed by Francis Guthrie in 1852, and a celebrated (incorrect) "proof" by Alfred Bray Kempe appeared in 1879. The four-color theorem was eventually proved by Kenneth Appel and Wolfgang Haken in 1976, building on the earlier work of Kempe, George Birkhoff, Heinrich Heesch, and others, and a simpler proof was subsequently produced by Neil Robertson, Daniel Sanders, Paul Seymour, and Robin Thomas [1997]. Meanwhile, attention had turned to the dual problem of coloring the vertices of a planar graph and of graphs in general.

There was also a parallel development in the coloring of the edges of a graph, starting with a result of Tait [1880], and leading to a fundamental theorem of V. G. Vizing in 1964. As mentioned earlier, the corresponding problem of coloring maps on other surfaces was settled by Ringel and Youngs in 1968.

The Four-Color Problem

Many developments in graph theory can be traced back to attempts to solve the celebrated four-color problem on the coloring of maps.

FACTS [BiLlWi98, Chapter 6] [Wi02]

F47: The earliest known mention of the four-color problem occurs in a letter from A. De Morgan to Hamilton, dated 23 October 1852. De Morgan described how a student had asked him whether every map can be colored with just four colors in such a way that neighbouring countries are colored differently. The student later identified himself as Frederick Guthrie, giving credit for the problem to his brother Francis, who formulated it while coloring the counties of a map of England. Hamilton was not interested in the problem.

F48: De Morgan wrote to various friends, outlining the problem and trying to describe where the difficulty lies. On 10 April 1860, the problem appeared in print, in an unsigned book review in the *Athenaeum*, written by De Morgan. This review was read in the U.S. by C. S. Peirce, who developed a life-long interest in the problem. An earlier printed reference, signed by "F.G.", appeared in the *Athenaeum* in 1854 [McK12].

F49: On 13 June 1878, at a meeting of the London Mathematical Society, Cayley asked whether the problem had been solved. Shortly after, he published a short note describing where the difficulty might lie, and he showed that it is sufficient to restrict one's attention to trivalent maps.

F50: In 1879, Kempe [Ke:1879], a former Cambridge student of Cayley, published a purported proof of the four-color theorem in the *American Journal of Mathematics*, which had recently been founded by Sylvester. Kempe showed that every map must contain a country with at most five neighbours, and he showed how any coloring of the rest of the map can be extended to include such a country. His solution included a new technique, now known as a ***Kempe-chain*** argument, in which the colors in a two-colored section of the map are interchanged. Kempe's proof for a map that contains a digon, triangle, or quadrilateral was correct, but his argument for the pentagon (where he used two simultaneous color-interchanges) was fallacious.

F51: In 1880, Tait [Ta:1878–80] presented "improved proofs" of the four-color theorem, all of them fallacious. Other people interested in the four-color problem at this time were C. L. Dodgson (Lewis Carroll), F. Temple (Bishop of London), and the Victorian educator J. M. Wilson.

F52: In 1890, Heawood [He:1890] published a paper in the *Quarterly Journal of Pure and Applied Mathematics*, pointing out the error in Kempe's proof, salvaging enough to deduce the five-color theorem, and generalizing the problem in various ways, such as for other surfaces (see §1.1.3). Heawood subsequently published another six papers on the problem, the last while he was in his 90th year. Kempe admitted his error, but he was unable to put it right.

F53: During the first half of the 20th century two ideas emerged, each of which finds its origin in Kempe's paper. The first is that of an ***unavoidable set*** — a set of configurations, at least one of which must appear in any map. Unavoidable sets were produced by P. Wernicke [We:1904] (see Figure 1.3.11), by P. Franklin, and by H. Lebesgue.

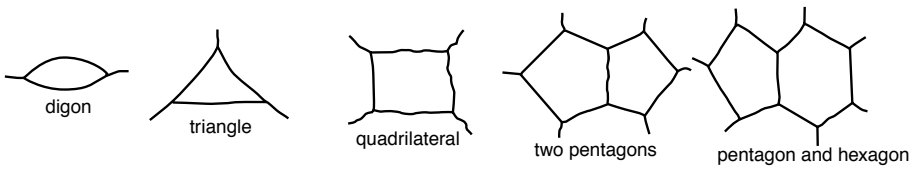


Figure 1.3.11: Wernicke's unavoidable set.

The second is that of a ***reducible configuration*** — a configuration of countries with the property that any coloring of the rest of the map can be extended to the configuration: no such configuration can appear in any counter-example to the four-color theorem. Birkhoff [Bi13] showed that the arrangement of four pentagons in Figure 1.3.12 (known as the ***Birkhoff diamond***) is a reducible configuration.

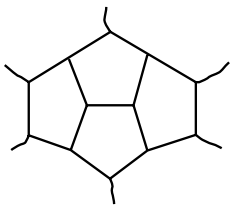


Figure 1.3.12: The Birkhoff diamond.

F54: In 1912, Birkhoff [Bi12] investigated the number of ways of coloring a given map with k colors, and he showed that this is always a polynomial in k, now called the *chromatic polynomial* of the map.

F55: In 1922, Franklin [Fr22] presented further unavoidable sets and reducible configurations, and he deduced that the four-color theorem is true for all maps with up to 25 countries. This number was later increased several times by other authors.

F56: Around 1950 Heesch started to search for an unavoidable set of reducible configurations. Over the next few years, Heesch [He69] produced thousands of reducible configurations.

F57: In 1976, Appel and Haken [ApHa77, ApHaKo77], with the assistance of J. Koch, obtained an unavoidable set of 1482 reducible configurations, thereby proving the four-color theorem. Their solution required substantial use of a computer to test the configurations for reducibility.

F58: Around 1994, Robertson, Sanders, Seymour, and Thomas [RoSaSeTh97] produced a more systematic proof. Using a computer to assist with both the unavoidable set and the reducible configuration parts of the solution, they systematized the Appel–Haken approach, and they obtained an unavoidable set of 633 reducible configurations.

Other Graph Coloring Problems

Arising from work on the four-color problem, progress was being made on other graph problems involving the coloring of edges or vertices.

FACTS [BiLlWi98, Chapter 6] [FiWi77] [JeTo95]

F59: In his 1879 paper on the coloring of maps, Kempe [Ke:1879] outlined the dual problem of coloring the vertices of a planar graph in such a way that adjacent vertices are colored differently. This dual approach to map-coloring was later taken up by H. Whitney in a fundamental paper of 1932 and by most subsequent workers on the four-color problem.

F60: In 1880, Tait [Ta:1878–80] proved that the four-color theorem is equivalent to the statement that the edges of every trivalent map can be colored with three colors in such a way that each color appears at every vertex.

F61: In 1916, D. König [Kö16] proved that the edges of any bipartite graph with maximum degree d can be colored with d colors. (See §11.3.)

F62: The idea of coloring the vertices of a graph so that adjacent vertices are colored differently developed a life of its own in the 1930s, mainly through the work of Whitney, who wrote his Ph.D. thesis on the coloring of graphs.

F63: In 1941, L. Brooks [Br41] proved that the chromatic number of any simple graph with maximum degree d is at most $d + 1$, with equality only for odd cycles and odd complete graphs. (See §5.1.)

F64: In the 1950s, substantial progress on vertex-colorings was made by G. A. Dirac, who introduced the idea of a *critical graph*.

F65: In 1964, V. G. Vizing [Vi64] proved that the edges of any simple graph with maximum degree d can always be colored with $d + 1$ colors. In the following year, Vizing produced many further results on edge-colorings.

F66: The concepts of the chromatic number and edge-chromatic number of a graph have been generalized by a number of writers — for example, M. Behzad and others introduced total colorings in the 1960s, and P. Erdős and others introduced list colorings.

Factorization

A graph is *k-regular* if each of its vertices has degree k. Such graphs can sometimes be split into regular subgraphs, each with the same vertex-set as the original graph. A *k-factor* in a graph is a k-regular subgraph that contains all the vertices of the original graph. Fundamental work on factors in graphs was carried out by Julius Petersen [1839–1910] and W. T. Tutte [1914–2002]. (See §5.4.)

FACTS [BiLlWi98, Chapter 10]

F67: In 1891, Petersen [Pe:1891] wrote a fundamental paper on the factorization of regular graphs, arising from a problem in the theory of invariants. In this paper he proved that if k is even, then any k-regular graph can be split into 2-factors. He also proved that any 3-regular graph possesses a 1-factor, provided that it has not more than two "leaves"; a leaf is a subgraph joined to the rest of the graph by a single edge.

F68: In 1898, Petersen [Pe:1898] produced a trivalent graph with no leaves, now called the ***Petersen graph*** (see Figure 1.3.13), which cannot be split into three 1-factors; it can, however, be split into a 1-factor (the spokes) and a 2-factor (the pentagon and pentagram).

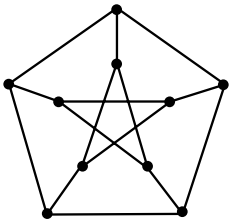


Figure 1.3.13: The Petersen graph.

F69: In 1947, Tutte [Tu47] produced a characterization of graphs that contain a 1-factor. Five years later he extended his result to a characterization of graphs that contain a k-factor, for any k.

1.3.5 Graph Algorithms

Graph theory algorithms can be traced back to the 19th century, when Fleury gave a systematic method for tracing an eulerian graph and G. Tarry showed how to escape from a maze (see §4.2). The 20th century saw algorithmic solutions to such problems as the minimum connector problem, the shortest and longest path problems, and the *Chinese Postman Problem* (see §4.3), as well as to a number of problems arising in operational research. In each of these problems we are given a network, or weighted graph, to each edge (and/or vertex) of which has been assigned a number, such as its length or the time taken to traverse it.

FACTS [Da82] [LLRS85] [LoPl86]

F70: The *Traveling Salesman Problem*, in which a salesman wishes to make a cyclic tour of a number of cities in minimum time or distance, appeared in rudimentary form in 1831. It reappeared in mathematical circles in the early 1930s, at Princeton, and was later popularized at the RAND Corporation. This led to a fundamental paper of G. B. Dantzig, D. R. Fulkerson, and S. M. Johnson [DaFuJo54] that included the solution of a traveling salesman problem with 49 cities. In the 1980s a problem with 2392 cities was settled by Padberg and Rinaldi [PaRi87]. (See §4.6.)

F71: The greedy algorithm for the *minimum connector problem*, in which one seeks a minimum-length spanning tree in a weighted graph, can be traced back to O. Boruvka [Bo26] and was later rediscovered by J. B. Kruskal [Kr56]. A related algorithm, due to V. Jarník (1931), was rediscovered by R. C. Prim (1957). (See §10.1.)

F72: Graph algorithms were developed by D. R. Fulkerson and G. B. Dantzig [FuDa55] for finding the maximum flow of a commodity between two nodes in a capacitated network, and by R. E. Gomory and T. C. Hu [GoHu61] for determining *maximum flows* in multi-terminal networks.

F73: Finding a longest path, or critical path, in an activity network dates from the 1940s and 1950s, with *PERT* (Program Evaluation and Review Technique) used by the U.S. Navy for problems involving the building of submarines and CPM (*Critical Path Method*) developed by the Du Pont de Nemours Company to minimize the total cost of a project. (See §3.2.)

F74: There are several efficient algorithms for finding the shortest path in a given network, of which the best known is due to E. W. Dijkstra [Di59]. (See §10.1.)

F75: The Chinese postman problem, for finding the shortest route that covers each edge of a given weighted graph, was originated by Meigu Guan (Mei-Ku Kwan) [Gu60] in 1960. (See §4.3.)

F76: In matching and assignment problems one wishes to assign people as appropriately as possible to jobs for which they are qualified. This work developed from work of König and from a celebrated result on matching due to Philip Hall [Ha35], later known as the "marriage theorem" [HaVa50]. These investigations led to the subject of polyhedral combinatorics and were combined with the newly emerging study of linear programming. (See §11.3.)

F77: By the late 1960s it became clear that some problems seemed to be more difficult than others, and Edmonds [Ed65] discussed problems for which a polynomial-time algorithm exists. Cook [Co71], Karp [Ka72], and others later developed the concept of NP-completeness. The assignment, transportation, and minimum spanning-tree problems are all in the *polynomial-time class* P, while the traveling salesman and Hamiltonian cycle problems are NP-hard. It is not known whether P = NP. Further information can be found in [GaJo79].

References

[ApHa77] K. Appel and W. Haken, Every planar map is 4-colorable: Part 1, Discharging, *Illinois J. Math.* 21 (1977), 429–490.

[ApHaKo77] K. Appel, W. Haken, and J. Koch, Every planar map is 4-colorable: Part 2, Reducibility, *Illinois J. Math.* 21 (1977), 429–490.

[BeWi09] L. W. Beineke and R. J. Wilson, *Topics in Topological Graph Theory 1736-1936*, Cambridge University Press, 2009.

[BiLlWi98] N. L. Biggs, E. K. Lloyd, and R. J. Wilson (eds.), *Graph Theory 1736-1936*, Oxford University Press, 1998.

[Bi12] G. D. Birkhoff, A determinantal formula for the number of ways of coloring a map, *Ann. of Math.* 14 (1912), 42–46.

[Bi13] G. D. Birkhoff, The reducibility of maps, *Amer. J. Math.* 35 (1913), 115–128.

[BiLe46] G. D. Birkhoff and D. C. Lewis, Chromatic polynomials, *Trans. Amer. Math. Soc.* 60 (1946), 355–451.

[BoCh76] J. A. Bondy and V. Chvátal, A method in graph theory, *Discrete Math.* 15 (1976), 111–136.

[Bo26] O. Boruvka, O jistém problému minimálním, *Acta Soc. Sci. Natur. Moravicae* 3 (1926), 37–58.

[Br41] R. L. Brooks, On colouring the nodes of a network, *Proc. Cambridge Philos. Soc.* 37 (1941), 194–197.

[Ca:1813] A.-L. Cauchy, Recherches sur les polyèdres-premier mémoire, *J. Ecole Polytech.* 9 (Cah. 16) (1813), 68–86.

[Ca:1857] A. Cayley, On the theory of the analytical forms called trees, *Phil. Mag.* (4) 13 (1857), 172–176.

[Ca:1874] A. Cayley, On the mathematical theory of isomers, *Phil. Mag.* (4) 47 (1874), 444–446.

[Ca:1879] A. Cayley, On the colouring of maps, *Proc. Roy. Geog. Soc.* (new Ser.) 1 (1879), 259–261.

[Ca:1889] A. Cayley, A theorem on trees, *Quart. J. Pure Appl. Math.* 23 (1889), 376–378.

[Co71] S. A. Cook, The complexity of theorem-proving procedures, *Proc. 3rd Annual ACM Symp. Theory of Computing*, pp151–158, ACM, New York, 1971.

[Cl:1844] T. Clausen, [Second postscript to] De linearum tertii ordinis proprietatibus, *Astron. Nachr.* 21 (1844), col. 209–216.

[Cr99] P. R. Cromwell, *Polyhedra*, Cambridge University Press, 1999.

[Da82] G. B. Dantzig, Reminiscences about the origins of linear programming, *Oper. Res. Lett.* 1 (1982), 43–48.

[DaFuJo54] G. B. Dantzig, D. R. Fulkerson, and S. M. Johnson, Solution of a large-scale traveling-salesman problem, *Oper. Res.* 2 (1954), 393–410.

[DeHe07] M. Dehn and P. Heegaard, Analysis situs, *Encyklopädie der Mathematischen Wissenschaften* (1907), 153–120.

[DeM:1860] A. De Morgan, A review of the philosophy of discovery, chapters historical and critical, by W. Whewell, D. D., *Athenaeum* No. 1694 (1860), 501–503.

[Di59] E. W. Dijkstra, A note on two problems in connexion with graphs, *Numer. Math.* 1 (1959), 269–271.

[Di52] G. A. Dirac, Some theorems on abstract graphs, *Proc. London Math. Soc.* (3) 2 (1952), 69–81.

[Du13] H. E. Dudeney, Perplexities, *Strand Mag.* 46 (July 1913), 110 and (August 1913), 221.

[Ed65] J. R. Edmonds, Paths, trees and flowers, *Canad. J. Math.* 17 (1965), 449–467.

[Eu:1736] L. Euler, (1736) Solutio problematis ad geometriam situs pertinentis, *Commentarii Academiae Scientiarum Imperialis Petropolitanae* 8 (1752), 128–140.

[Eu:1759] L. Euler, Solution d'une question curieuse qui ne paroit soumise à aucune analyse, *Mem. Acad. Sci. Berlin* 15 (1759), 310–337.

[FiWi77] S. Fiorini and R. J. Wilson, *Edge-Colourings of Graphs*, Pitman, 1977.

[FoFu56] L. R. Ford and D. R. Fulkerson, Maximal flow through a network, *Canad. J. Math.* 8 (1956), 399–404.

[Fr22] P. Franklin, The four color problem, *Amer. J. Math.* 44 (1922), 225–236.

[Fr34] P. Franklin, A six color problem, *J. Math. Phys.* 13 (1934), 363–369.

[FuDa55] D. R. Fulkerson and G. B. Dantzig, Computation of maximum flow in networks, *Naval Research Logistics Quarterly* 2 (1955), 277–283.

[GaJo79] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Co., 1979.

[GlHuWa79] H. H. Glover, J. P. Huneke and C. S. Wang, 103 graphs that are irreducible for the projective plane, *J. Combin. Theory* (B) 27 (1979), 332–370.

[GoHu61] R. E. Gomory and T. C. Hu, Multi-terminal network flows, *SIAM J. Appl. Math.* 9 (1961), 551–556.

[Gr74] J. L. Gross, Voltage graphs, *Discrete Math.* 9 (1974), 239–246.

[GrTu74] J. L. Gross and T. W. Tucker, Quotients of complete graphs: revisiting the Heawood problem, *Pacific J. Math.* 55 (1974), 391–402.

[Gu60] Guan Meigu, Graphic programming using odd or even points, *Acta Math. Sinica* 10 (1962), 263–266; *Chinese Math.* 1 (1962), 273–277.

[Gu63] W. Gustin, Orientable embedding of Cayley graphs, *Bull Amer. Math. Soc.* 69 (1963), 272–275.

[Ha35] P. Hall, On representatives of subsets, *J. London Math. Soc.* 10 (1935), 26–30.

[HaVa50] P. R. Halmos and H. E. Vaughan, The marriage problem, *Amer. J. Math.* 72 (1950), 214–215.

[Ha:1856] W. R. Hamilton, Memorandum respecting a new system of roots of unity, *Phil. Mag.* (4) 12 (1856), 446.

[Ha55] F. Harary, The number of linear, directed, rooted, and connected graphs, *Trans. Amer. Math. Soc.* 78 (1955), 445–463.

[HaPa73] F. Harary and E. M. Palmer, *Graphical Enumeration*, Academic Press, 1973.

[He:1890] P. J. Heawood, Map-colour theorem, *Quart. J. Pure Appl. Math.* 24 (1890), 332–338.

[He:1891] L. Heffter, Über das Problem der Nachbargebiete, *Math. Ann.* 38 (1891), 477–580.

[He69] H. Heesch, Untersuchungen zum Vierfarbenproblem, *B. I. Hochschulscripten*, 810/810a/810b, Bibliographisches Institut, Mannheim-Vienna-Zürich, 1969.

[Hi:1873] C. Hierholzer, Über die Möglichkeit, einen Lineanzug ohne Wiederholung und ohne Unterbrechnung zu umfahren, *Math. Ann.* 6 (1873), 30–32.

[HoWi04] B. Hopkins and R. Wilson, The truth about Konigsberg?, *College Math. J.* 35 (2004), 198-207.

[JeTo95] T. R. Jensen and B. Toft, *Graph Coloring Problems*, Wiley–Interscience, 1995.

[Ka35] I. N. Kagno, A note on the Heawood color formula, *J. Math. Phys.* 14 (1935), 228–231.

[Ka72] R. M. Karp, Reducibility among combinatorial problems, 85–103 in *Complexity of Computer Computations* (ed. R. E. Miller and J. W. Thatcher), Plenum Press, 1972.

[Ke:1879] A. B. Kempe, On the geographical problem of four colours, *Amer. J. Math.* 2 (1879), 193–200.

[Ki:1856] T. P. Kirkman, On the representation of polyedra, *Phil. Trans. Roy. Soc. London* 146 (1856), 413–418.

[Kö16] D. König, Über Graphen und ihre Anwendung auf Determinantentheorie und Mengenlehre, *Math. Ann.* 77 (1916), 453–465.

[Kr56] J. B. Kruskal, On the shortest spanning subtree of a graph and the traveling salesman problem, *Proc. Amer. Math. Soc.* 7 (1956), 48–50.

[Ku30] K. Kuratowski, Sur le problème des courbes gauches en topologie, *Fund. Math.* 15 (1930), 271–283.

[LLRS85] E. L. Lawler, J. K. Lenstra, A. H. G. Rinooy Kan, and D. B. Schmoys (eds.), *The Traveling Salesman Problem: A Guided Tour through Combinatorial Optimization*, Wiley, 1985.

[Le:1794] A.-M. Legendre, *Eléments de Géométrie* (1st ed.), Firmin Didot, Paris, 1794.

[Lh:1811] S.-A.-J. Lhuilier, Démonstration immédiate d'un théorème fondamental d'Euler sur les polyhèdres, et exceptions dont ce théorème est susceptible, *Mém. Acad. Imp. Sci. St. Pétersb.* 4 (1811), 271–301.

[Li:1847] J. B. Listing, Vorstudien zur Topologie, *Göttingen Studien* (Abt. 1) Math. Naturwiss. Abh. 1 (1847), 811–875.

[Li:1861–2] J. B. Listing, Der Census räumliche Complexe, *Abh. K. Ges. Wiss. Göttingen Math. Cl.* 10 (1861–2), 97–182.

[LoPl86] L. Lovász and M. D. Plummer, Matching Theory, *Annals of Discrete Mathematics* 29, North-Holland, 1986.

[Lu:1882] E. Lucas, *Récréations Mathématiques*, Vol. 1, Gauthier-Villars, Paris (1882).

[LuSe29] A. C. Lunn and J. K. Senior, Isomerism and configuration, *J. Phys. Chem.* 33 (1929), 1027–1079.

[Ma69] J. Mayer, Le problème des régions voisines sur les surfaces closes orientables, *J. Combin. Theory* 6 (1969), 177–195.

[McK12] B. McKay, A note on the history of the four-colour conjecture, *J. Graph Theory* 72 (2013), 361–363.

[Or60] O. Ore, Note on Hamiltonian circuits, *Amer. Math. Monthly* 67 (1960), 55.

[Ot48] R. Otter, The number of trees, *Ann. of Math.* 49 (1948), 583–599.

[PaRi87] M. W. Padberg and G. Rinaldi, Optimization of a 532-city symmetric traveling salesman problem by branch and cut, *Oper. Res. Lett.* 6 (1987), 1–7.

[Pe:1891] J. Petersen, Die Theorie der regulären Graphs, *Acta Math.* 15 (1891), 193–220.

[Pe:1898] J. Petersen, Sur le théorème de Tait, *Interméd. Math.* 5 (1898), 225–227.

[Po:1809–10] L. Poinsot, Sur les polygones et les polyèdres, *J. Ecole Polytech.* 4 (1809–10) (Cah. 10), 16–48.

[Pó37] G. Pólya, Kombinatorische Anzahlbestimmungen für Gruppen, Graphen und chemische Verbindungen, *Acta Math.* 68 (1937), 145–254.

[PóRe87] G. Pólya and R. C. Read, *Combinatorial Enumeration of Groups, Graphs and Chemical Compounds*, Springer, 1987.

[Pr18] H. Prüfer, Neuer Beweis eines Satzes über Permutationen, *Arch. Math. Phys.* (3) 27 (1918), 142–144.

[Re63] R. C. Read, On the number of self-complementary graphs and digraphs, *J. London Math. Soc.* 38 (1963), 99–104.

[Re27] J. H. Redfield, The theory of group-reduced distributions, *Amer. J. Math.* 49 (1927), 433–455.

[Re:1871–3] M. Reiss, Evaluation du nombre de combinaisons desquelles les 28 dés d'un jeu du domino sont susceptibles d'après la règle de ce jeu, *Ann. Mat. Pura. Appl.* (2) 5 (1871–3), 63–120.

[Ri74] G. Ringel, *Map Color Theorem*, Springer, 1974.

[RiYo68] G. Ringel and J. W. T. Youngs, Solution of the Heawood map-coloring problem, *Proc. Nat. Acad. Sci. U.S.A.* 60 (1968), 438–445.

[RoSe85] N. Robertson and P. D. Seymour, Graph minors — a survey, in *Surveys in Combinatorics 1985* (ed. I. Anderson), London Math. Soc. Lecture Notes Series 103 (1985), Cambridge University Press, 153–171.

[RoSaSeTh97] N. Robertson, D. Sanders, P. Seymour, and R. Thomas, The four-colour theorem, *J. Combin. Theory, Ser. B* 70 (1997), 2–44.

[Ro:1892] W. W. Rouse Ball, *Mathematical Recreations and Problems of Past and Present Times* (later entitled *Mathematical Recreations and Essays*), Macmillan, London, 1892.

[SaStWi88] H. Sachs, M. Stiebitz and R. J. Wilson, An historical note: Euler's Königsberg letters, *J. Graph Theory* 12 (1988), 133–139.

[Sh49] C. E. Shannon, A theorem on coloring the lines of a network, *J. Math. Phys.* 28 (1949), 148–151.

[Sy:1877–8] J. J. Sylvester, Chemistry and algebra, *Nature* 17 (1877–8), 284.

[Sy:1878] J. J. Sylvester, On an application of the new atomic theory to the graphical representation of the invariants and covariants of binary quantics, *Amer. J. Math.* 1 (1878), 64–125.

[Ta:1878–80] P. G. Tait, Remarks on the colouring of maps, *Proc. Roy. Soc. Edinburgh* 10 (1878–80), 729.

[Ti10] H. Tietze, Einige Bemerkungen über das Problem des Kartenfärbens auf einseitigen Flächen, *Jahresber. Deut. Math.-Ver.* 19 (1910), 155–179.

[Tu46] W. T. Tutte, On hamiltonian circuits, *J. London Math. Soc.* 21 (1946), 98–101.

[Tu47] W. T. Tutte, The factorizations of linear graphs, *J. London Math. Soc.* 22 (1947), 107–111.

[Tu59] W. T. Tutte, Matroids and graphs, *Trans. Amer. Math. Soc.* 90 (1959), 527–552.

[Tu70] W. T. Tutte, On chromatic polynomials and the golden ratio, *J. Combin. Theory* 9 (1970), 289–296.

[Va:1771] A.-T. Vandermonde, Remarques sur les problèmes de situation, *Mém. Acad. Sci. (Paris)* (1771), 556–574.

[Ve22] O. Veblen, *Analysis Situs*, Amer. Math. Soc. Colloq. Lect. 1916, New York, 1922.

[Vi64] V. G. Vizing, On an estimate of the chromatic class of a p-graph, *Diskret. Analiz* 3 (1964), 25–30.

[Vi65] V. G. Vizing, The chromatic class of a multigraph, *Diskret. Analiz* 5 (1965), 9–17.

[We:1904] P. Wernicke, Über den kartographischen Vierfarbensatz, *Math. Ann.* 58 (1904), 413–426.

[Wh31] H. Whitney, Non-separable and planar graphs, *Proc. Nat. Acad. Sci. U.S.A.* 17 (1931), 125–127.

[Wh35] H. Whitney, On the abstract properties of linear dependence, *Amer. J. Math.* 57 (1935), 509–533.

[Wi99] R. J. Wilson, Graph Theory, Chapter 17 in *History of Topology* (editor, I. M. James), Elsevier Science, 1999.

[Wi02] R. Wilson, *Four Colors Suffice*, Allen Lane, 2002; Princeton University Press, 2002.

Glossary for Chapter 1

bipartite graph: a graph whose vertices can be partitioned into two sets (called the *partite sets*) in such a way, that no edge joins two vertices in the same set. (For technical reasons, this includes the graph K_1 in this definition.)

bouquet B_n: the general graph with one vertex and n self-loops.

Cayley graph $C(\mathcal{A}, X)$ – for a group $\mathcal{A}$ with generating set X: the digraph whose vertex-set is $\mathcal{A}$ with an edge directed from a to ax for every $a \in \mathcal{A}$ and every $x \in X$. Sometimes two oppositely directed edges corresponding to an involution x are merged into a single undirected edge. (The underlying undirected graph of a Cayley graph is also commonly called a Cayley graph.)

circulant graph $\mathrm{Circ}(n; X)$: a Cayley graph for a cyclic group $\mathbb{Z}_n$.

complete k-partite graph $K_{n_1,n_2,\ldots,n_k}$: a simple k-partite graph such that two vertices are adjacent if and only if they are in different partite sets. All such graphs are called *complete multipartite graphs*.

complete bipartite graph: a simple bipartite graph such that each vertex in one partite set is adjacent to all the vertices in the other partite set. If the two partite sets have cardinalities r and s, then this graph is denoted $K_{r,s}$.

complete digraph $\overleftrightarrow{K}_n$: the simple digraph on n vertices such that between every pair of vertices, there is an arc in both directions.

complete graph K_n: the simple graph with n vertices in which every pair of vertices is joined by an edge.

k-connected graph: a graph such that the result of removing fewer than k vertices is connected and nontrivial, for all possible choices of the vertices.

connectivity of a graph G: the largest number k such that G is k-connected. It is denoted $\kappa(G)$ or $\kappa_V(G)$.

critically k-chromatic graph: a graph of chromatic number k whose chromatic number would decrease if any edge were removed. (See §5.1.)

critically k-connected graph: a graph of connectivity k whose connectivity would decrease if any vertex were removed. (See §4.1.)

critically k-edge-connected graph: a graph of edge-connectivity k whose edge-connectivity would decrease if any edge were removed. (See §4.1.)

cube: see *hypercube*.

cube graph: see *hypercube graph*.

cycle graph C_n: the n-vertex graph with n edges, such that every edge lies on a single cycle.

cycle rank – for a graph $G = (V, E)$ with $c(G)$ components: the number $|E(G)| - |V(G)| + c(G)$.

dipole D_n: the multigraph with two vertices and n edges.

edge-connectivity of a graph G: the largest number k such that G is k-edge-connected. It is denoted $\kappa'(G)$ or $\kappa_E(G)$.

k-edge-connected graph: a graph such that the result of removing fewer than k edges is connected and nontrivial, for all possible choices of the edges.

empty graph $\overline{K_n}$: the graph with n vertices and no edges.

eulerian graph: a graph with a closed walk that contains every edge exactly once. (See §1.3 for the history of eulerian graphs and §4.2 for an extensive discussion.)

k-factor of a graph G: a k-regular subgraph.

genus: see *minimum genus*.

hamiltonian graph: a graph that has a spanning cycle. (See Section 1.3 for the history of hamiltonian graphs and Section 4.6 for an extensive discussion.)

hypercube of dimension d: $\{(x_1, \ldots, x_d) \mid 0 \geq x_j)\}$.

hypercube graph Q_d: the 1-skeleton of a d-dimensional hypercube.

line graph of a graph G: the simple graph $L(G)$ whose vertex-set is the edge-set of G, and in which two vertices are adjacent if the edges in G to which they correspond have a common vertex. Also, a graph H is said *to be a line graph* if there exists a graph G such that H is isomorphic to $L(G)$.

minimum genus (or **genus**) of a connected graph G: the smallest number g such that G can be drawn on the orientable surface S_g (see Section 7.1) without any edge-crossings. Notation: $\gamma_{min}(G)$ or $\gamma(G)$.

null graph K_0: the graph with no vertices and no edges.

octahedral graph $\mathcal{O}_d$: the edge-complement of a 1-factor in K_{2d}.

partite sets: see *k-partite* graph.

p-partite graph: a graph whose vertex-set can be partitioned into p subsets (called the *partite sets*) in such a way that no edge joins two vertices in the same subset.

Petersen graph: a 10-vertex 3-regular graph, commonly depicted as a 5-pointed star inside a pentagon, with a 1-factor joining the vertices of the pentagon to the points of the star.

path graph P_n: the n-vertex graph with $n-1$ edges, such that every edge lies on a single open path. (Quite commonly elsewhere, the subscript of the notation P_n denotes the number of edges.)

planar graph: a graph of minimum genus 0, i.e., a graph that can be drawn in the sphere or plane with no edge crossings.

platonic graph: the skeleton of any of the five platonic solids.

platonic solid: any of the five regular 3-dimensional polyhedra — tetrahedron, cube, octahedron, dodecaheron, icosahedron.

regular graph: a graph in which every vertex is of the same degree. It is *k-regular* if every vertex is of degree k.

simplex: the convex hull of a set S of affinely independent points in Euclidean space. It is a *k-simplex* if $|S| = k+1$.

skeleton (or 1-*skeleton*) of a k-complex K: the graph consisting of the vertices and the edges of K.

trivial graph K_1: the graph with one vertex and no edges.

Chapter 2

Graph Representation

Section 2.1
Computer Representations of Graphs

Alfred V. Aho, Columbia University

INTRODUCTION

Many problems in science and engineering can be modeled in terms of directed and undirected graphs. The data structures and algorithms used to represent graphs can have a significant impact on the size of problems that can be implemented on a computer and the speed with which they can be solved. This section presents the fundamental representations used in computer programs for graphs and illustrates the tradeoffs among the representations using key algorithms for some of the most common graph problems.

Throughout this section we use the notation $|X|$ to denote the number of elements in a set X. The graphs and digraphs in this section are assumed to be simple.

2.1.1 Basic Representations for Graphs

The two most basic representations for a graph are the adjacency matrix and the adjacency list.

DEFINITIONS

D1: A ***directed graph*** or ***digraph*** $G = (V, E)$ consists of a finite, nonempty set of *vertices* V and a set of *edges* E. Each edge is an ordered pair (v, w) of vertices.

D2: An ***undirected graph*** $G = (V, E)$ consists of a finite, nonempty set of *vertices* V and a set of *edges* E. Each edge is a set $\{v, w\}$ of vertices.

D3: In a directed graph $G = (V, E)$, vertex w is ***adjacent*** to vertex v if (v, w) is an edge in E. The number of vertices adjacent to v is called the ***out-degree*** of v.

D4: In an undirected graph $G = (V, E)$, vertex w is ***adjacent*** to vertex v if $\{v, w\}$ is an edge in E. The number of vertices adjacent to v is called the ***degree*** of v.

D5: A ***path*** in a directed or undirected graph is a seqence of edges (v_1, v_2), (v_2, v_3),..., (v_{n-1}, v_n). This path is from vertex v_1 to vertex v_n and has length $n-1$.

D6: A graph $G = (V, E)$ is ***dense*** when the number of edges is close to $|V|^2$.

D7: A graph $G = (V, E)$ is ***sparse*** when the number of edges is much less than $|V|^2$.

D8: An ***adjacency matrix*** representation for a simple graph or digraph $G = (V, E)$ is a $|V| \times |V|$ matrix A, where $A[i, j] = 1$ if there is an edge from vertex i to vertex j; $A[i, j] = 0$ otherwise.

D9: An ***adjacency list*** representation for a graph or digraph $G = (V, E)$ is an array L of $|V|$ lists, one for each vertex in V. For each vertex i, there is a pointer L_i to a linked list containing all vertices j adjacent to i. A linked list is terminated by a ***nil pointer***.

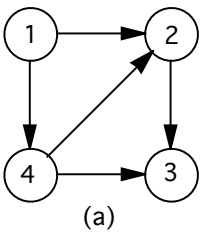
D10: An ***incidence matrix*** representation for a simple digraph $G = (V, E)$ is a $|V| \times |E|$ matrix I, where

$$I[v, e] = \begin{cases} -1 & \text{if edge } e \text{ is directed } \textit{to} \text{ vertex } v \\ 1 & \text{if edge } e \text{ is directed } \textit{from} \text{ vertex } v \\ 0 & \text{otherwise} \end{cases}$$

For an undirected graph, $I[v, e] = 1$ if e is incident on v and 0 otherwise.

EXAMPLES

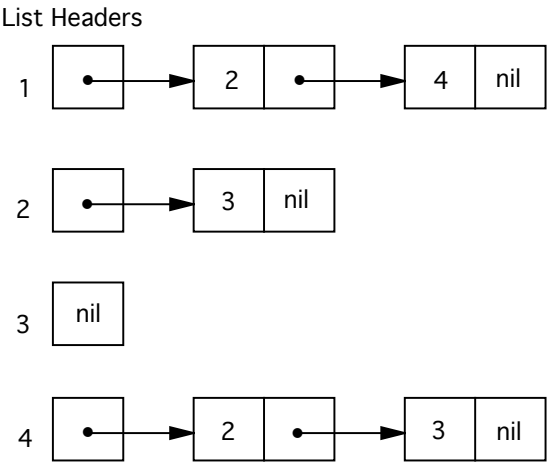
E1: Figure 2.1.1 shows the adjacency matrix and adjacency list representations of a directed graph.



(a)

	1	2	3	4
1	0	1	0	1
2	0	0	1	0
3	0	0	0	0
4	0	1	1	0

(b)



(c)

Figure 2.1.1: (a) A directed graph G. (b) Adjacency matrix for G. (c) Adjacency list representation for G.

E2: An incidence matrix for the digraph of Figure 2.1.1 is shown below.

$$I_G = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccccc} (1,2) & (1,4) & (2,3) & (4,2) & (4,3) \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & -1 & 0 \\ 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & 0 & 1 & 1 \end{pmatrix} \end{array}$$

FACTS

F1: An adjacency matrix representation for a graph $G = (V, E)$ always takes $O(|V|^2)$ space.

F2: An adjacency list representation for a graph $G = (V, E)$ takes $O(|V| + |E|)$ space.

REMARKS

R1: For a more detailed discussion of graph representations, see [AhHoUl74, AhHoUl83, CoLeRiSt09, Ev79, Ta83].

R2: As a general rule, an adjacency list representation is preferred when a graph is sparse, because it takes space that is linearly proportional to the number of vertices and edges.

R3: When a graph $G = (V, E)$ is dense, both an adjacency matrix and an adjacency list representation require $O(|V|^2)$ space. However, with the adjacency matrix, we can determine whether an edge exists in constant time, whereas with the adjacency list we may need $O(|V|)$ time. For this reason, adjacency matrix representations are often used with dense graphs.

R4: Note that in an adjacency list representation of an undirected graph, an edge i, j appears on two adjacency lists: the list for vertex i and the list for vertex j.

2.1.2 Graph Traversal Algorithms

One of the most fundamental tasks in algorithms involving graphs is visiting the vertices and edges of a graph in a systematic order. Depth-first and breadth-first search are frequently used traversal techniques for both directed and undirected graphs. For both these techniques, the adjacency list representation of a graph works well.

Depth-First Search

ALGORITHM

Depth-first search systematically visits all the vertices of a graph. Initially, all vertices are marked "new". When a vertex is visited, it is marked "old". Depth-first search works by selecting a new vertex v, marking it old, and then calling itself recursively on each of the vertices adjacent to v. The algorithm below is called "depth-first search" because it searches along a path in the forward (deeper) direction looking for new vertices as long as it can.

Algorithm 2.1.1 Depth-First Search

Input: A graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$
and $L[v]$ is a pointer to the list of vertices adjacent to vertex v.
Output: Traversal of all vertices in V in a depth-first order.

```
procedure DepthFirstSearch(G){
  for v := 1 to n do
    mark[v] := new;
  for v := 1 to n do
    if mark[v] = new; then
      dfs(v);
}
procedure dfs(v){
  mark[v] := old;
  for each vertex w on L[v] do
    if mark[w] = new then
      dfs(w);
}
```

DEFINITIONS

During the course of its traversal, depth-first search partitions the graph into a collection of ***depth-first trees*** that make up a ***depth-first forest***. The forest and its trees are determined by the edges, which are partitioned by the search into four sets:

D11: ***Tree edges*** are those edges (v, w) where w is first encountered by exploring edge (v, w).

D12: ***Back edges*** are those edges (v, w) that connect a vertex v to an ancestor w in a depth-first tree.

D13: ***Forward edges*** are those nontree edges (v, w) that connect a vertex v to a proper descendant in a depth-first tree.

D14: ***Cross edges*** are the remaining edges. They connect vertices that are neither ancestors nor descendants of one another.

FACTS

F3: Depth-first search takes $O(|V| + |E|)$ time on a graph $G = (V, E)$.

F4: If we represent the first visit of a vertex v with a left parenthesis "(v" and its last visit by a right parenthesis "v)", then the sequence of first and last visits forms an expression in which the parentheses are properly nested.

F5: In a depth-first search of an undirected graph, every edge is either a tree edge or a back edge.

REMARKS

R5: Depth-first search is a fundamental graph algorithm that has been in use since the 1950s. [Ta72, HoTa73] developed several efficient graph algorithms using depth-first search.

R6: Depth-first search forms the basis of many important graph algorithms such as determining the biconnected components of an undirected graph and finding the strongly connected components of a directed graph.

Breadth-First Search

Breadth-first search is another fundamental technique for exploring a graph G. It starts from a specified *source* vertex s from which it constructs a ***breadth-first tree*** consisting of all vertices of G reachable from s. In the process it computes a breadth-first tree rooted at s such that if a vertex v is reachable from s in G, there is a path in the tree from the root to s. The path in the tree is a shortest path from s to v in G.

ALGORITHM

Breadth-first search uses the abstract data type *queue* to hold vertices as they are being processed. The operation $enqueue(s, Q)$ places vertex s on the back of the queue Q. The operation $dequeue(Q)$ removes the element at the front of the queue Q.

Breadth-first search (Algorithm 2.1.2) visits the vertices of a graph G uniformly across the breadth of the frontier of its search, visiting all vertices at distance d from s, before looking for vertices at distance $d + 1$. In contrast, depth-first search plunges as deeply into the graph along a path as it can before backtracking to visit nodes closer to s.

DEFINITION

D15: Let BFT be the tree with root s, vertices v such that $parent[v]$ is not **nil**, and edges $\{(parent[v], v) | parent[v] \text{ is not } \textbf{nil}\}$. BFT is the ***breadth-first tree*** constructed by $BreadthFirstSearch(G, s)$.

FACTS

F6: Breadth-first search takes $O(|V| + |E|)$ time on a graph $G = (V, E)$.

F7: $BreadthFirstSearch(G, s)$ computes the length of the shortest path from s to v in $distance[v]$.

Algorithm 2.1.2 Breadth-First Search

Input: A graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$, $L[v]$ is a pointer to the list of vertices adjacent to vertex v, and where s is a specified source vertex.
Output: A breadth-first tree consisting of root s and all vertices in V that are reachable from s.

```
procedure BreadthFirstSearch(G, s) {
  for v := 1 to n do {
    mark[v] := new;
    distance[v] := ∞;
    parent[v] := nil;
  }
  mark[s] := visited;
  distance[s] := 0;
  initialize queue Q;
  enqueue(s, Q);
  while Q is not empty do {
    v := dequeue(Q);
    for each vertex w on L[v] do
      if mark[w] = visited then {
        mark[w] := visited;
        distance[w] := distance[w] + 1;
        parent[w] := v;
        enqueue(w, Q);
      }
  }
}
```

REMARKS

R7: Like depth-first search, breadth-first search has been used since the 1950s. Early applications of breadth-first search included maze searching and routing wires on printed circuit boards.

R8: The ideas found in breadth-first search are the building blocks of many other graph algorithms such as Dijkstra's single-source shortest-paths algorithm and Prim's algorithm for finding minimal spanning trees.

2.1.3 All-Pairs Problems

This section considers two algorithms: one for computing the shortest paths between all pairs of vertices in a directed graph and the other for computing the transitive closure of a directed graph. For both algorithms the adjacency matrix is a natural representation for the graph.

All-Pairs Shortest-Paths Algorithm

Suppose that we have a schedule that tells us the driving time between n cities at a given time of day and that we wish to compute the shortest driving time between all pairs of cities. This is an instance of the *all-pairs shortest-paths* problem. We could iterate through every pair of cities and compute the shortest path between each using a single-source shortest-path algorithm such as Dijkstra's algorithm.

ALGORITHM

An easier way is to use the Floyd–Warshall algorithm below. The natural representation for a graph in the Floyd–Warshall algorithm is an adjacency matrix. Assume that we are given a directed graph $G = (V, E)$ and that the vertices in V are numbered $1, 2, \ldots, n$. Further assume that we are given a matrix $C[i, j]$ that tells us the cost of edge (i, j). If there is no edge $C[i, j]$, then we assume $C[i, j]$ is set to infinity. We assume all other costs are non-negative.

Algorithm 2.1.3 Floyd–Warshall

Input: A directed graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$;
and a cost matrix $C[i, j]$.
Output: Cost matrix $A[1..n, 1..n]$ where $A[i, j]$ is the cost of
the cheapest path from i to j.

```
procedure FloydWarshall(G) {
  for i := 1 to n do
    for j := 1 to n do
      A[i,j] := C[i,j];
  for i := 1 to n do
    A[i,i] := 0;
  for k := 1 to n do
    for i := 1 to n do
      for j := 1 to n do
        if A[i,k] + A[k,j] < A[i,j] then
          A[i,j] := A[i,k] + A[k,j];
}
```

The Floyd–Warshall algorithm computes a cheapest-cost array A, where $A[i, j]$ gives the cheapest cost of any path from vertex i to vertex j. For the algorithm to work correctly, it is important that there are no negative cost cycles in the graph.

FACT

F8: The Floyd–Warshall algorithm computes the cost matrix of the cheapest paths between all pairs of vertices of a directed graph $G = (V, E)$ in $O(|V|^3)$ time and $O(|V|^2)$ space.

REMARKS

R9: For additional discussion of the Floyd–Warshall algorithm and its variants see [AhHoUl74] and [CoLeRiSt09].

R10: Let $A^k[i,j]$ be the cost of the cheapest path from vertex i to vertex j that does not pass through a vertex numbered higher than k, except possibly for the endpoints. We can prove by induction on k that $A^k[i,j] = min(A^{k-1}, A^{k-1}[i,k] + A^{k-1}[k,j])$. In the next section we see that the Floyd–Warshall algorithm is a special case of Kleene's algorithm.

Transitive Closure

In some problems we may just want to know whether there exists a path from vertex i to vertex j of length one or more in a graph $G = (V, E)$. We call this the problem of computing the ***transitive closure*** of G. Given a directed graph $G = (V, E)$ with adjacency matrix A, we want to compute a Boolean matrix T such that $T[i,j]$ is 1 if there is a path from i to j of length 1 or more, and 0 otherwise. We call T the *transitive closure* of the adjacency matrix.

The transitive-closure algorithm below is similar to the Floyd–Warshall algorithm except that it uses the Boolean operation **and** to conclude that if there is a path from i to k and one from k to j, then there is a path from i to j.

Algorithm 2.1.4 Transitive Closure

Input: A directed graph $G = (V, E)$, with $V = \{1, 2, \ldots, n\}$
and adjacency matrix $A[i,j]$.
Output: Boolean transitive-closure matrix $T[1..n, 1..n]$ where $T[i,j]$ is
1 if there is a path from i to j of length 1 or more, and 0 otherwise.

```
procedure TransitiveClosure(G) {
  for i := 1 to n do
    for j := 1 to n do
      T[i,j] := A[i,j];
  for k := 1 to n do
    for i := 1 to n do
      for j := 1 to n do
        if A[i,j] = false then
          A[i,j] := A[i,k] or A[k,j];
}
```

FACT

F9: The algorithm $TransitiveClosure(G)$ computes the transitive closure of G in $O(|V|^3)$ time and $O(|V|^2)$ space.

REMARKS

R11: The transitive-closure algorithm is due to S. Warshall [Wa62].

R12: Let $T^k[i,j] = 1$ if there is a path of length one or more from vertex i to vertex j that does not pass through an intermediate vertex numbered higher than k, except for the endpoints. We can prove by induction on k that

$$C^k[i,j] = C^{k-1}[i,j] \textbf{ or } C^{k-1}[i,k] \textbf{ and } C^{k-1}[k,j]$$

where **and** and **or** are the Boolean *and* and *or* operators. In the next subsection we will see the transitive-closure algorithm is a special case of Kleene's algorithm.

2.1.4 Applications to Pattern Matching

Graphs play a major role in problems arising in the specification and translation of programming languages. A special kind of graph called a ***finite automaton*** is used in language theory to specify and recognize sets of strings called ***regular expressions***. Regular expressions are used to specify the lexical structure of many programming language constructs. They are also widely used in many string-pattern-matching applications.

This section presents an algorithm due to S. C. Kleene to construct representations called ***regular expressions*** for all paths between the vertices of a directed graph.

DEFINITIONS

D16: A ***nondeterministic finite automaton*** (NFA) is a labeled, directed graph $G = (V, E)$ in which

1. one vertex is distinguished as the *start* vertex
2. a set of vertices are distinguished as *final* vertices
3. each edge is labeled by a symbol from a set $\Sigma \cup \{\epsilon\}$ where
   Σ is a finite set of *alphabet symbols*, and
   ϵ is a special symbol denoting the empty string

D17: An NFA G ***accepts*** a string x if there is a path in G from the start vertex to a final vertex whose edge labels spell out x.

D18: The set of strings accepted by an NFA G is called the ***language*** defined by G.

D19: If R and S are sets of strings, then their ***concatenation*** $R \cdot S$ is the set of strings $\{xy | x \text{ is in } r \text{ and } y \text{ is in } S\}$.

D20: Let S be a set of strings. Define $S^0 = \{\epsilon\}$ and $S^i = S \cdot S^{i-1}$ for $i \geq 1$. The ***Kleene closure*** of S, denoted S^*, is defined to be $\cup_{i=0}^{\infty} S^i$.

D21: Let Σ be a finite set of alphabet symbols. The ***regular expressions*** over Σ and the languages they denote are defined recursively as follows:

1. ϕ is a regular expression that denotes the empty set.
2. ϵ is a regular expression that denotes $\{\epsilon\}$.
3. For each a in Σ, a is a regular expression that denotes $\{a\}$.
4. If r and s are regular expressions denoting the languages R and S, then
   $(r + s)$ is a regular expression denoting the language $R \cup S$,
   rs is a regular expression denoting $R \cdot S$, and
   (r^*) is a regular expression denoting R^*.

We can avoid writing many parentheses in a regular expression by adopting the convention that the Kleene closure operator * has higher precedence than concatenation or $+$, and that concatenation has higher precedence than $+$. For example, $((a(b^*)) + c)$ may be written $ab^* + c$. This regular expression denotes the set of strings $\{ab^i | i \geq 0\} \cup \{c\}$.

Kleene's Algorithm

S. C. Kleene presented an algorithm for constructing a regular expression from a nondeterministic finite automaton. This algorithm, shown below, includes the Floyd–Warshall algorithm and the transitive-closure algorithm as special cases.

ALGORITHM

Let $G = (V, E)$ be an NFA in which the vertices are numbered $1, 2, \ldots, n$. Kleene's algorithm (Algorithm 2.1.5) works by constructing a sequence of matrices C^k in which the entry $C^k[i, j]$ is a regular expression for all paths from vertex i to vertex j with no intermediate vertex on the path (except possibly for the endpoints) that is numbered higher than k.

Algorithm 2.1.5 Kleene's Algorithm

Input: A directed graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$,
and a label matrix $L[i, j]$.
Output: Matrix $C[1..n, 1..n]$ where $C[i, j]$ is a regular expression describing all paths from i to j.

```
procedure Kleene(G) {
  for i := 1 to n do
    for j := 1 to n do
      C^0[i,j] := L[i,j];
  for i := 1 to n do
    C^0[i,i] := ε + C^0[i,i];
  for k := 1 to n do
    for i := 1 to n do
      for j := 1 to n do
        C^k[i,j] := C^{k-1}[i,j] + C^{k-1}[i,k] · (C^{k-1}[k,k])* · C^{k-1}[k,j];
  for i := 1 to n do
    for j := 1 to n do
      C[i,j] := C^n[i,j];
}
```

REMARKS

R13: Kleene's algorithm appeared in [Kl56].

R14: To prove the correctness of Kleene's algorithm, we can prove by induction on k that $C^k[i, j]$ is the set of path labels of all paths from vertex i to vertex j with no intermediate vertex numbered higher than k, excluding the endpoints. The term $C^{k-1}[i, k]$ in the inner loop represents the labels of all paths from vertex i to vertex k that

do not have an intermediate vertex numbered higher than $k-1$. The term $(C^{k-1}[k,k])^*$ represents the labels of all paths that go from vertex k to vertex k zero or more times without passing through an intermediate vertex numbered higher than $k-1$. The term $C^{k-1}[k,j]$ represents the labels of all paths from vertex k to vertex j that do not have an intermediate vertex numbered higher than $k-1$. Thus, the term $C^{k-1}[i,j] \cdot (C^{k-1}[k,k])^* \cdot C^{k-1}[k,j])$ represents the path labels of all paths with the segments: from i to k, from k to k zero or more times, and from k to j with no intermediate vertex numbered higher than $k-1$ on any of the segments.

R15: The Floyd–Warshall algorithm is a special case of Kleene's algorithm with the inner loop replaced by $C^k[i,j] := min(C^{k-1}[i,j], C^{k-1}[i,k] + C^{k-1}[k,j]$. In the Floyd–Warshall algorithm we don't need to consider paths from k to k since we assume the edge costs are non-negative. Also, in the Floyd–Warshall algorithm the operator representing concatenation $(\cdot)$ is arithmetic addition.

R16: The transitive-closure algorithm is a special case of Kleene's algorithm with the inner loop replaced by $C^k[i,j] := C^{k-1}[i,j] + C^{k-1}[i,k] \cdot C^{k-1}[k,j]$ where $+$ represents Boolean **or** and $\cdot$ represents Boolean **and**.

R17: Aho, Hopcroft, and Ullman present Kleene's algorithm in the general setting of a closed semiring [AhHoUl74].

R18: One of the key results of formal language theory is that the set of languages defined by NFAs is exactly the same as the set of languages defined by regular expressions. These languages are called ***regular sets***.

R19: For applications of finite automata and regular expressions to string pattern matching and compiling see [Ah90, AhSeUl86].

References

[Ah90] A. V. Aho, Algorithms for finding patterns in strings, pp. 255–300 in *Handbook of Theoretical Computer Science A, Algorithms and Complexity*, Ed. J. Van Leeuwen, MIT Press, 1990.

[AhHoUl74] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms,* Addison-Wesley, 1974.

[AhHoUl83] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *Data Structures and Algorithms,* Addison-Wesley, 1983.

[AhSeUl86] A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools,* Addison-Wesley, 1986.

[CoLeRiSt09] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition,* MIT Press, 2009.

[Ev79] S. Even, *Graph Algorithms,* Computer Science Press, 1979.

[Fl62] R. W. Floyd, Algorithm 97 (Shortest Path), *Comm. ACM* 5(6) (1962), 345.

[HoTa73] J. E. Hopcroft and R. E. Tarjan, Efficient algorithms for graph manipulation, *Comm. ACM* 16(6) (1973), 372–378.

[Jo77] D. B. Johnson, Efficient algorithms for shortest paths in sparse networks, *J. ACM* 24(1) (1977), 1–13.

[Kl56] S. C. Kleene, Representation of events in nerve nets and finite automata, pp. 3–40 in *Automata Studies*, Eds. C. E. Shannon and J. McCarthy, Princeton Univ. Press, 1985.

[Ta72] R. E. Tarjan, Depth first search and linear graph algorithms, *SIAM J. Comput.* 1(2) (1972), 146–160.

[Ta83] R. E. Tarjan, *Data Structures and Network Algorithms,* Soc. Industrial and Applied Mathematics, 1983.

[Wa62] S. Warshall, A theorem on Boolean matrices, *J. ACM* 9(1) (1962), 11–12.

Section 2.2
Graph Isomorphism

Brendan D. McKay, Australian National University

INTRODUCTION

Isomorphism between graphs and related objects is a fundamental concept in graph theory and its applications to other parts of mathematics. The problem also occupies a central position in complexity theory as a proposed occupant of the region that must exist between the polynomial-time and NP-complete problems if P≠NP. Due to its many practical applications a considerable number of algorithms for graph isomorphism have been proposed.

2.2.1 Isomorphisms and Automorphisms

Informally, two graphs are isomorphic if they are the same except for the names of their vertices and edges. Formally, this relationship is defined by means of bijections between them.

Basic Terminology

DEFINITIONS

D1: Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be simple graphs. An ***isomorphism*** from G_1 to G_2 is a bijection $\phi : V_1 \to V_2$ such that $vw \in E_1$ if and only if $\phi(v)\phi(w) \in E_2$.

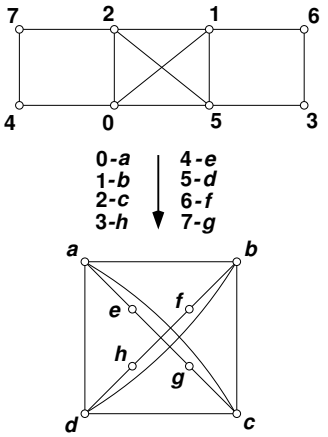
D2: A second way to define an isomorphism is that there are two bijections $\phi : V_1 \to V_2$ and $\phi' : E_1 \to E_2$, such that the incidence relation between vertices and edges is preserved. That is, $v \in V_1$ is incident to $e \in E_1$ if and only if $\phi(v)$ is incident to $\phi(e)$. This method is preferred if edges have additional attributes that should be preserved by the mapping. However, we will use the previous definition where it applies.

D3: An isomorphism from a graph to itself is called an ***automorphism*** or ***symmetry***.

D4: The set of automorphisms of a graph G form a group under the operation of composition, called the ***automorphism group*** Aut(G). The automorphism group of a simple graph is a subgroup of the symmetric group acting on the vertex set of the graph.

EXAMPLE

E1: Figure 2.2.1 shows an isomorphism between two graphs and gives the automorphism group of the first graph.



(1)
(0 2)(4 7)
(1 5)(3 6)
(0 2)(4 7)(1 5)(3 6)
(0 5)(1 2)(3 4)(6 7)
(0 1 2 5)(3 4 6 7)
(0 5 2 1)(3 7 6 4)
(0 1)(2 5)(3 7)(4 6)

Figure 2.2.1: An isomorphism between two graphs and the automorphism group of the first graph.

DEFINITION

D5: Closely related to isomorphism is the concept of canonical labeling. Arbitrarily choose one member of each isomorphism class of graphs, and call it the ***canonical form*** of that isomorphism class. Replacing a graph by the canonical form of its isomorphism class is called ***canonical labeling*** or ***canonizing the graph***. Two graphs are isomorphic if and only if their canonical forms are identical, as shown in Figure 2.2.2.

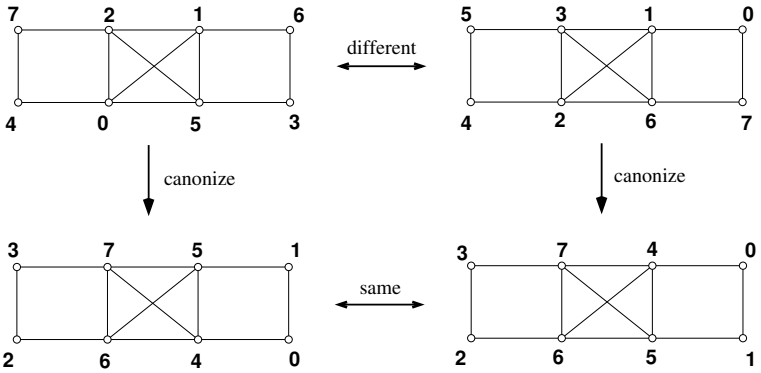


Figure 2.2.2: Isomorphism between graphs becomes equality when they are canonized.

REMARKS

R1: The labeled graph which gives the lexicographically greatest adjacency matrix is an example of an explicitly defined canonical form. In practice more complex definitions are used to assist efficient computation.

R2: Canonical labeling has central importance to practical applications. One task is to determine whether a graph is isomorphic to any graph in a database of graphs. This is best achieved by storing the canonical forms of graphs in the database and comparing them to the canonical form of the new graph. Another task is to remove isomorphs from a large collection of graphs. This is best achieved by applying a sorting algorithm to the canonical forms of the graphs. Both tasks are very expensive if only pair-wise isomorphism testing is available.

Related Isomorphism Problems

Many types of isomorphism problem can be modeled as isomorphism between simple graphs or digraphs.

FACTS

F1: *Vertex colors* that must be preserved by isomorphisms can be modeled by attaching gadgets to the vertices, a different gadget for each color. However, this is such an important generalization that most software can handle vertex colors directly.

F2: *Edge colors* can be modeled using layers, once vertex colors are available. Figure 2.2.3 illustrates one approach. The edge colors are assigned numbers according to the table in the center. The vertices of the original graph are assigned to vertical paths, with the first layer identified by vertex color. Then the original edges with each color c are represented by horizontal edges within those layers where the binary expansion of c has ones.

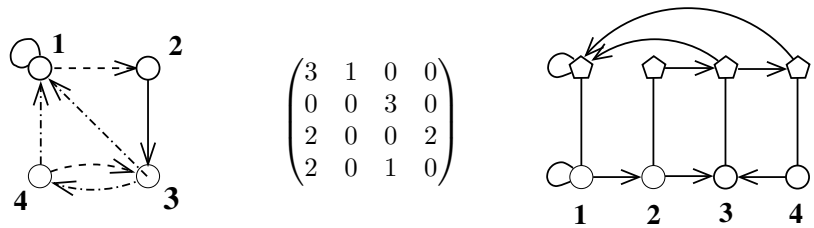


Figure 2.2.3: Modeling of graphs with colored edges.

F3: *Hypergraphs* and other types of incidence structures like *block designs* and *finite geometries* can be represented by bipartite graphs. One color class consists of the vertices of the hypergraph, while the other has a vertex for each of the hyperedges of the hypergraph. An edge of the bipartite graph represents a vertex of the hypergraph and a hyperedge it lies in.

F4: Other types of isomorphism easily modeled by graph isomorphism include equivalence of matrices defined by permutation of rows and columns, Hadamard equivalence, and isotopy (such as for Latin squares) [McPi12b].

2.2.2 Complexity Theory

The problem of determining whether two graphs are isomorphic, called ***GI*** or ***ISO***, has received a great deal of interest from theorists due to its unsolved nature.

FACTS

F5: GI is not known to have a polynomial-time algorithm, nor to be NP-complete. While obviously in NP, its presence in co-NP is also undecided. Indeed, it is considered a prime candidate for the intermediate territory between P and NPC that must exist if P≠NP. One reason for this is that the NP-completeness of GI would imply the collapse of the polynomial-time hierarchy [GoMiWi91].

F6: The fastest proven running time for GI has stood for three decades at $e^{O(\sqrt{n \log n})}$ [BaKaLu83].

F7: On the other hand, many special classes of graph are known to have polynomial-time isomorphism tests. The most general such classes are those defined by a forbidden minor [Po88, Gr10] or by a forbidden topological minor [Gr12]. These classes include many earlier classes, including graphs of bounded degree [Lu82], bounded genus [FiMa80, Mi80], and bounded tree-width [Bo90]. However, very few of these polynomial algorithms are practical.

DEFINITION

D6: A decision problem is called ***isomorphism-complete*** if it is polynomial-time equivalent to GI.

FACTS

F8: All of the isomorphism problems noted in the previous subsection are isomorphism-complete. Many other examples are known, including isomorphism of semigroups and finite automata [Bo78], homeomorphism of 2-complexes [STPi94], and polytope isomorphism [KaSc03].

F9: Isomorphism of linear codes (vector spaces over finite fields) defined by permutation of the coordinate positions, where the codes are presented as generator matrices, is at least as hard as graph isomorphism but might be harder [PeRo97].

F10: Isomorphism of groups given as multiplication tables is at least as easy as GI but might be easier [Bo78]. The best algorithm is very elementary and takes time $n^{O(\log n)}$ [Bo78, Mi78]).

F11: Some problems similar to GI are NP-complete. The best known is the subgraph isomorphism problem: given two graphs, is the first isomorphic to a subgraph of the second? Another is the presence of an automorphism without fixed points, or of such an automorphism of order 2 [Lu81].

DEFINITION

D7: A ***graph invariant*** is a property of graphs that is equal for isomorphic graphs. A ***complete graph invariant***, also called a ***certificate***, is an invariant that always distinguishes between non-isomorphic graphs.

FACTS

F12: Examples of invariants include the degree sequence and the eigenvalue set of the adjacency matrix. However, neither of those invariants is complete. Nevertheless, even incomplete invariants can sometimes be used as a short proof of non-isomorphism.

F13: An example of a complete invariant is a canonical form. However, it is not known if there is a complete invariant computable in polynomial time. In fact, it is not even known if there is a complete invariant checkable in polynomial time (which would place GI in co-NP).

2.2.3 Algorithms

The development of computer programs for graph isomorphism has been such a popular pursuit that already in 1976 it was called a "disease" [ReCo77]. Literally hundreds of algorithms have been published (many wrong).

FACTS

F14: The earliest software appeared in the 1960s. The approach which has been the most successful is the "individualization-refinement" paradigm introduced by Parris and Read [PaRe69] and further developed by Corneil and Gotlieb [CoGo70] and Arlazarov et al. [ArZuUsFa74]. This genre is now represented by the author's `nauty` and other software mentioned below.

REMARK

R3: We will focus our attention on canonical labeling, which is the method used by the most useful modern algorithms.

DEFINITIONS

D8: A key routine is that of ***partition refinement***, which is any process of making a partition finer (i.e., breaking its cells into smaller cells) by detecting combinatorial differences between the vertices. For isomorphism purposes, only properties independent of the numbering of the vertices may be used. This implies that vertices equivalent under the action of an automorphism fixing the input partition cannot be separated.

D9: An ***equitable partition*** is a partition of the vertices of a graph into cells such that, for any two vertices v, w in the same cell, and any cell C, we have that v and w are adjacent to the same number of vertices in C.

EXAMPLE

E2: Figure 2.2.4 shows a graph with an equitable partition of two cells. Each black vertex is adjacent to no black and two white vertices, while each white vertex is adjacent to one white and two black vertices. As this example shows, vertices in the same cell of an equitable partition do not need to be equivalent under the automorphism group of the graph.

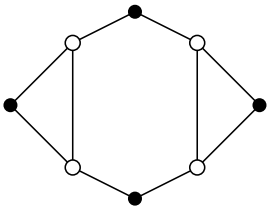


Figure 2.2.4: An equitable partition with two cells.

FACTS

F15: The most well-known partition refinement method splits cells until the partition becomes equitable. Given two cells C_1, C_2, the vertices in C_1 are separated into subcells according to their number of neighbors in C_2. This is repeated for different pairs of cells until no more splitting is possible. Algorithms differ according to the method used to choose the pairs. The fastest algorithm is in [Mc80].

F16: A generalization of this type of refinement, called the ***k-th order Weissfeiler–Lehman refinement***, uses partitions of the set of k-tuples of vertices rather than just a partition of the vertices. For some classes of graphs, it is known that there is a fixed k for which this refinement provides the automorphism partition of the graph, which can be used to build a polynomial-time isomorphism algorithm. However, Cai, Füredi, and Immerman showed that no such k is sufficient for all graphs [CaFuIm92].

Search Tree

A partition refinement method is used to define a search tree that is used to find a canonical labeling and the automorphism group.

DEFINITION

D10: Consider a graph and an initial partition (perhaps trivial). Define a ***search tree*** whose nodes are refined partitions. The root of the tree is the refined initial partition. Any node which is a discrete partition (one with only singleton cells) is a leaf of the tree. Consider any node ν which is not discrete. Choose a non-singleton cell C. Then ν has one child for each $v \in C$, obtained by splitting C into two cells $\{v\}$ and $C - \{v\}$, then refining.

FACTS

F17: The leaves of the search tree are discrete partitions, and thus, lists of the vertices in a definite order. The orders define a set of numberings of the vertices of the graph.The maximum labeled graph, according to lexicographic or other convenient ordering, is a canonical form. Moreover, two numberings that define the same labeled graph yield an automorphism, and all automorphisms can be found in that way.

F18: In practice the search tree may be much too large, so various means are employed to reduce it. One way is to employ automorphisms as they are discovered to prune branches of the tree thus shown to be equivalent to other branches. Another way is to employ invariants computed at the nodes of the tree to perform a type of branch-and-bound. A third method is to use a more powerful refinement procedure.

Software

The first program that could process structurally highly regular graphs and graphs with hundreds of vertices was that of the author, which became known as `nauty` [Mc78, Mc80] and dominated the field from the 1970s until recent years. Now there are several strong competitors.

EXAMPLES

E3: `nauty`, by this author, can find automorphisms groups and canonical forms, of graphs and digraphs. It comes in two forms, with either dense or sparse data structures [McPi12b, McPi13].

E4: `Saucy`, by Darga, Liffiton, Sakallah and Markov, computes automorphism groups and is especially efficient for large sparse graphs having many automorphisms that move few vertices [DaLiSaMa04, DaSaMa04].

E5: `Bliss`, by Junttila and Kaski, can also perform canonical labeling and has very dependable performance for highly regular graphs [JuKa07, JuKa11].

E6: `Traces`, by Piperno, introduced an entirely new way of scanning the search tree, using a combination of breadth-first and depth-first search [Pi08, McPi13]. At the time of writing, `Traces` is the most efficient program for processing many classes of very difficult graphs, as well as being highly competitive for easy graphs [McPi13]. Since January 2013, `Traces` has been distributed with `nauty` [McPi12].

E7: Other worthy programs are `conauto` by López-Presa, Fernándes Anta, and Núñez Chiroque [LPFe09, JPFe11], `VSEP` by Stoichev, and `VF` by Cordella, Foggia, Sansone, and Vento.

E8: Packages which contain high-quality graph isomorphism facilities (usually via `nauty`) include Magma, GRAPE, LINK, Sage-combinat, and Macaulay2.

REMARKS

R4: An experimental comparison of `nauty`, `Traces`, `saucy`, `Bliss`, and `conauto` can be found in [McPi13].

R5: All the named programs have exponential running time in the worst case. However, the worst-case graphs are rather difficult to find and most users will see only scaling according to a polynomial of low degree. The state of the art is that the easiest graphs can be handled if they fit into main memory (tens of millions of vertices). The most difficult graphs cause difficulty in the hundreds or thousands of vertices.

References

[ArZuUsFa74] V. L. Arlazarov, I. I. Zuev, A. V. Uskov, and I. A. Faradzev, An algorithm for the reduction of finite non-oriented graphs to canonical form. *Zh. vȳchisl. Mat. mat. Fiz.* 14 (1974) 737–743.

[BaKaLu83] L. Babai, W. M. Kantor, and E. M. Luks, Computational complexity and the classification of finite simple groups. In: Proceedings of the 24th Annual Symposium on the Foundations of Computer Science (1983) 162–171.

[Bo90] H. Bodlaender, Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees. *J. Algorithms* 11 (1990) 631–643.

[Bo78] K. S. Booth, Isomorphism testing for graphs, semigroups, and finite automata are polynomially equivalent problems. *SIAM J. Comput.* 7 (1978) 273–279.

[CaFuIm92] Jin-yi Cai, Martin Fürer, and Neil Immerman, An optimal lower bound on the number of variables for graph identifications. *Combinatorica* 12 (1992) 389–410.

[CoGo70] D. G. Corneil and C. C. Gotlieb, An efficient algorithm for graph isomorphism. *JACM* 17 (1970) 51–64.

[DaLiSaMa04] P. T. Darga, M. H. Liffiton, K. A. Sakallah, and I. L. Markov, Exploiting structure in symmetry detection for CNF. In: Proceedings of the 41st Design Automation Conference (2004), 530–534.

[DaSaMa04] P. T. Darga, K. A. Sakallah, and I. L. Markov, Faster Symmetry Discovery using Sparsity of Symmetries. In: Proceedings of the 45th Design Automation Conference (2004), 149–154.

[FiMa80] I. S. Filotti and J. N. Mayer, A polynomial-time algorithm for determining the isomorphism of graphs of fixed genus. In: Proceedings of the 12th ACM Symposium on Theory of Computing (1980), 236–243.

[GoMiWi91] O. Goldreich, S. Micali, and A. Wigderson, Proofs that yield nothing but their validity, or all languages in NP have zero-knowledge proof systems. *JACM* 38 (1991) 690–728.

[Gr10] M. Grohe, Fixed-point definability and polynomial time on graphs with excluded minors. In: Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science (2010), 179–188.

[Gr12] M. Grohe, Structural and Logical Approaches to the Graph Isomorphism Problem, In: Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (2012), 188.

[JuKa07] T. Junttila and P. Kaski, Engineering an efficient canonical labeling tool for large and sparse graphs. In: Proceedings of the 9th Workshop on Algorithm Engineering and Experiments and the 4th Workshop on Analytic Algorithms and Combinatorics (2007), 135–149.

[JuKa11] T. Junttila and P. Kaski, Conflict Propagation and Component Recursion for Canonical Labeling. In: Proceedings of the 1st International ICST Conference on Theory and Practice of Algorithms (2011), 151–162.

[KaSc03] V. Kaibel and A. Schwartz, On the complexity of polytope isomorphism problems. *Graphs and Combinatorics* 19 (2003) 215–230.

[LPFe09] J. L. López-Presa and A. Fernández Anta, Fast algorithm for graph isomorphism testing. In: Proceedings of the 8th International Symposium on Experimental Algorithms (2009), 221–232.

[JPFe11] J. L. López-Presa, A. Fernández Anta, and L. Núñez Chiroque, Conauto-2.0: Fast isomorphism testing and automorphism group computation. Preprint 2011. Available at `http://arxiv.org/abs/1108.1060`.

[Lu81] A. Lubiw, Some NP-complete problems related to graph isomorphism. *SIAM J. Comput.* 10 (1981) 11–21.

[Lu82] E. Luks, Isomorphism of graphs of bounded valence can be tested in polynomial time. *J. Comp. System Sci.* 25 (1982) 42–65.

[Mc78] B. D. McKay, Computing automorphisms and canonical labelings of graphs. In: Combinatorial Mathematics, Lecture Notes in Mathematics, 686. Springer-Verlag, Berlin (1978), 223–232.

[Mc80] B. D. McKay, Practical graph isomorphism. *Congr. Numer.* 30 (1980) 45–87.

[McPi12] B. D. McKay and A. Piperno, `nauty Traces`, Software distribution web page. `http://cs.anu.edu.au/~bdm/nauty/` and `http://pallini.di.uniroma1.it/`.

[McPi12b] B. D. McKay and A. Piperno, nauty and Traces User's Guide (Version 2.5). available at [McPi12].

[McPi13] B. D. McKay and A. Piperno, Practical graph isomorphism II, arXiv:1301.1493..

[Mi78] G. L. Miller, On the $n^{\log n}$ isomorphism technique. In: Proceedings of the 10th ACM Symposium on Theory of Computing (1978) 51–58.

[Mi80] G. L. Miller, Isomorphism testing for graphs of bounded genus. In: Proceedings of the 12th ACM Symposium on Theory of Computing (1980), 225–235.

[PaRe69] R. Parris and R. C. Read, A coding procedure for graphs. Scientific Report. UWI/CC 10. University of West Indies Computer Centre, 1969.

[PeRo97] E. Petrank and R. M. Roth, Is code equivalence easy to decide? *IEEE Trans. Inform. Th.* 43 (1997) 1602–1604.

[Pi08] A. Piperno, Search space contraction in canonical labeling of graphs. Preprint 2008–2011. Available at `http://arxiv.org/abs/0804.4881`.

[Po88] I. N. Ponomarenko, The isomorphism problem for classes of graphs that are invariant with respect to contraction (Russian). *Zap. Nauchn. Sem. Leningrad. Otdel. Mat. Inst. Steklov. (LOMI)* 174 (1988) no. Teor. Slozhn. Vychisl. 3, 147–177.

[ReCo77] R. C. Read and D. G. Corneil, The graph isomorphism disease. *J. Graph Theory* 1 (1977) 339–363.

[STPi94] J. Shawe-Taylor and T. Pisanski, Homeomorphism of 2-complexes is graph isomorphism complete. *SIAM J. Comput.* 23 (1994) 120–132.

Section 2.3
The Reconstruction Problem

Josef Lauri, University of Malta, Malta

INTRODUCTION

In the first volume of the *Journal of Graph Theory* published in 1977, the journal editors wrote, "The foremost currently unsolved problem in Graph Theory is, in our considered opinion, the Reconstruction Conjecture." One might agree or disagree with this assessment, but surely one must admit that this is an inviting problem, made all the more tantalizing by the fact that, although thirty-five years have passed since the editors of the *Journal of Graph Theory* expressed their views, and several researchers have made efforts to attain, at least, some partial reconstruction results, we are still nowhere near any complete solution of this problem. In the first section below we shall give the two main variants of the Reconstruction Problem, and in the subsequent sections we shall expand on these definitions hoping, this way, to provide a panoramic view of the present state of knowledge on the Reconstruction Problem.

In this section all graphs are assumed to be simple.

2.3.1 Two Reconstruction Conjectures

Some classical problems in mathematics are of the following type. If the structure S' is associated with the given structure S, does S' determine S uniquely? In graph theory we ask what knowledge, short of its full incidence relations, is sufficient to determine the graph completely. The structure S would be the graph and S' could be its line-graph, or chromatic polynomial, or spectrum, say. The best known problem of this type in graph theory is the Reconstruction Problem.

Decks and Edge-Decks

We first need to introduce some definitions and notations connected with families of vertex-deleted and edge-deleted subgraphs of a graph.

DEFINITIONS

D1: Let G be a graph (assumed here to have no loops or multiple edges) on n vertices. For any vertex v of G, let $G - v$ denote the ***vertex-deleted subgraph*** of G that is obtained from G by removing v and all edges incident to v. Similarly, let e be an edge of G. The ***edge-deleted subgraph*** of G, denoted by $G - e$, is obtained from G by deleting the edge e.

D2: If A is a subset of vertices of G then $G - A$ will denote the graph obtained from G by deleting all vertices in A and any edge incident to at least one of them. If B is a subset of edges of G then $G - B$ will denote that subgraph of G obtained by deleting all the edges in B.

D3: The collection of all vertex-deleted subgraphs of G is called the ***deck*** of G, while the collection of all edge-deleted subgraphs of G is called the ***edge-deck*** of G.
NOTATION: The deck of G is denoted by $\mathcal{D}(G)$ and the edge-deck of G is denoted by $\mathcal{ED}(G)$.

Note that the graphs in the deck are unlabeled and if G contains isomorphic vertex-deleted subgraphs, then such subgraphs are repeated in $\mathcal{D}G$ according to the number of isomorphic subgraphs which G contains. The same holds for the edge-deck. Therefore the deck and the edge-deck are multi-sets, rather than sets, of isomorphism types of graphs.

EXAMPLE

E1: Figure 2.3.1 shows a graph and its deck.

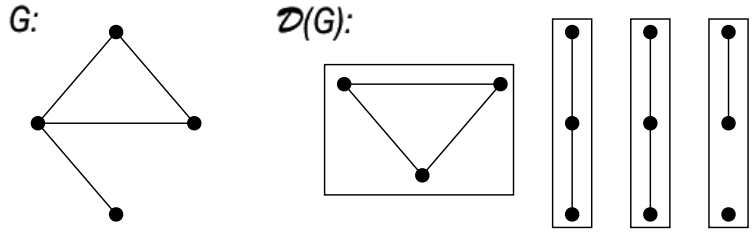


Figure 2.3.1: A graph and its deck.

Reconstructibility

Now suppose that H is another graph with $\mathcal{D}(H) = \mathcal{D}(G)$. The question we are interested in is: must H be isomorphic to G?

DEFINITION

D4: Any graph H with the same deck as G is called a ***reconstruction*** of G. If every reconstruction of G is isomorphic to G, then G is said to be ***reconstructible***.

EXAMPLE

E2: A graph that is not reconstructible is K_2, the graph consisting of just one edge, because if H is the graph consisting of two isolated vertices, then clearly H is a reconstruction of K_2, but not isomorphic to K_2. The Reconstruction Conjecture asserts that these are the only non-reconstructible graphs.

The Reconstruction Conjecture

C1: [Ke57, Ul60] *Every graph with at least three vertices is reconstructible.*

Another way of looking at reconstruction is by saying that a graph G is reconstructible if it can be uniquely (up to isomorphism) determined from $\mathcal{D}(G)$. Note that the problem stated in this way is not about finding an efficient algorithm for reconstructing G from $\mathcal{D}(G)$. In principle, given the deck, one can consider all graphs on n vertices to check which of them have the given deck. The question remains one of uniqueness, that is, whether this search will find only one graph with the given deck. The deck is a collection of isomorphism types with appropriate multiplicities, and the question is whether the isomorphism type of G can be determined uniquely from this collection.

DEFINITIONS

Closely related to the Reconstruction Problem is the *Edge-Reconstruction Problem.*

D5: An ***edge-reconstruction*** of G is a graph which has the same edge-deck as G, and G is said to be ***edge-reconstructible*** if every edge-reconstruction of G is isomorphic to it.

The Edge-Reconstruction Conjecture

C2: [Ha64] *Every graph with at least four edges is edge-reconstructible.*

EXAMPLE

E3: The graph $G = K_3 \cup kK_1$ is not edge-reconstructible because, if H is the graph $K_{1,3} \cup (k-1)K_1$, then H is an edge-reconstruction of G that is not isomorphic to it. Also, $G = 2K_2$ is not edge-reconstructible because if $H = P_3 \cup K_1$ (where P_3 is the path on three vertices), then $\mathcal{ED}(G) = \mathcal{ED}(H)$ but $G \ncong H$.

The Edge-Reconstruction Conjecture asserts that these are the only graphs that are not edge-reconstructible.

REMARK

R1: Two important surveys on the Reconstruction Problem were published in 1977 and 1978 [BoHe77, Na78]. These give the state of knowledge on this problem till that date, together with a complete bibliographic list. Since then, a number of survey or expository articles have been published [El88, Ma88, La87, Bo91]. The two early surveys and the most recent one should be consulted by anyone who intends to do serious work in this field. The book [LaSc03] also contains four chapters on the reconstruction problem.

Comparing Reconstruction with Edge-Reconstruction

Intuition seems to suggest that it is easier to reconstruct a graph from its edge-deck than from its deck; there are generally more graphs in the edge-deck, and edge-deleted subgraphs are generally more nearly like the original graph than vertex-deleted subgraphs. This intuitive notion is borne out by the following theorem of Greenwell which, essentially, says that if the Reconstruction Conjecture is true then so is the Edge-Reconstruction Conjecture.

FACTS

F1: [Gr71] [***Greenwell's Theorem***] Let G be a graph without isolated vertices. The deck of G is edge-reconstructible, that is, $\mathcal{D}(G)$ is uniquely determined from $\mathcal{ED}(G)$. Therefore, if G is reconstructible, then it is also edge-reconstructible.

Therefore, for graphs without isolated vertices, if the Reconstruction Conjecture is true, then so is the Edge-Reconstruction Conjecture. (We shall henceforth tacitly assume, unless otherwise stated, that any graph to be reconstructed has no isolated vertices.) So, if we can prove a result for vertex reconstruction, say that a certain class of graphs is reconstructible, then this result would automatically hold for edge-reconstruction.

Another result which shows that the problem of edge-reconstruction is a special case of reconstruction is the following theorem of Hemminger.

F2: [He69] A graph is edge-reconstructible if and only if its line graph is reconstructible and is not K_3.

However, the Edge-Reconstruction Problem holds its own considerable independent interest because several results are known in edge-reconstruction which have not yet been proved for vertex-reconstruction, and some elegant proof techniques have been developed for edge-reconstruction. We shall consider these techniques in some detail in a later section.

Reconstruction and Graph Symmetries

It is well to emphasize here that at the heart of the difficulty of reconstructing a graph G from its deck is the symmetry of G and of the subgraphs in its deck. As an illustration, let us consider an extreme case. Suppose that the vertices of G are labeled $1, \ldots, n$ and that these labels are preserved on every vertex-deleted subgraph. Then, clearly, the graph G can be uniquely reconstructed by considering any three subgraphs in its deck and 'superimposing' them accordingly. This happens because the labelings have essentially removed all symmetries of G and its subgraphs, and therefore all ambiguities of how these subgraphs are embedded inside G. A later section of this chapter will bring out more clearly the role of the automorphism of G in the edge-reconstruction of G.

But the situation just described does, in fact, occur quite often, and considering how this happens should help one understand where the difficulty of reconstruction lies.

DEFINITION

D6: Let k be a fixed but arbitrary integer. Then the graph G is said to have ***property*** A_k if, whenever A and B are distinct k-sets of vertices of G, the graphs $G - A$ and $G - B$ are not isomorphic. In other words, if G has n vertices, then any two subgraphs of G induced by different sets of $n - k$ vertices are not isomorphic.

FACTS

F3: If G has property A_{k+1} then it has property A_k and if it has property A_1 then its automorphism group is trivial.

F4: [Ko71, Mü76, Bo90] For a fixed k almost every graph has property A_k, meaning that the proportion of labeled graphs on n vertices which have this property tends to 1 as n goes to ∞.

F5: [My88, Bo90] If a graph G has property A_3 then it can be reconstructed uniquely from any three subgraphs in its deck.

REMARK

R2: Considering why the last fact holds helps to gain insight into the Reconstruction Problem. Let $G-u, G-v, G-w$ be any three subgraphs from the deck of G. Since G also has property A_2, we can identify v in $G-u$ and u in $G-v$ as the only vertices which give $G-u-v \simeq G-v-u$. Also, by property A_3, there is a unique isomorphism from $G-u-v$ to $G-v-u$. This isomorphism labels the two graphs uniquely, and we have the situation of a labeled graph which we described above. By comparing the two graphs $G-u, G-v$ we can then clearly put the vertex u back in $G-u$ and join it to its neighbors in G. The only uncertainty is whether or not u is adjacent to v. But this can be resolved by repeating the above with $G-u$ and $G-w$ instead of $G-v$.

2.3.2 Some Reconstructible Parameters and Classes

Faced with the olympian task of settling either way the Reconstruction Conjectures, most mathematicians have settled on attacking the conjectures partially, by showing that certain graph parameters or graph classes are reconstructible. We shall here present the most important parameters or classes which have been shown to be reconstructible, focusing mainly on results which have been obtained since the surveys mentioned in Section 1 and on those classes and parameters with which we shall not be dealing in the next three sections.

Reconstructible Parameters

DEFINITION

D7: A graph parameter $\mathcal{P}$ is said to be reconstructible (or edge-reconstructible) if, for any graph G with the value p for the parameter, any reconstruction (or edge-reconstruction) of G also has value p for $\mathcal{P}$. Equivalently, $\mathcal{P}$ is reconstructible from $\mathcal{D}(G)$ (or $\mathcal{ED}(G)$) if it is uniquely determined by the deck (or edge-deck).

FACTS

Recall that, by Greenwell's Theorem, for graphs without isolated vertices, a parameter is edge-reconstructible if it is reconstructible. For some of the following facts it is easy to show that the respective parameter is reconstructible.

F6: The number of vertices and edges are both reconstructible and edge-reconstructible. (See [LaSc03].)

F7: The degree sequence is reconstructible and edge-reconstructible. (See [Bo91, LaSc03].)

F8: Given a $G - v$ from the deck of G, the degree in G of the missing vertex and the degrees in G of the neighbors of v are reconstructible. (See [LaSc03].)

F9: Given a $G - e$ in the edge-deck of G, the degrees in G of the vertices with which the missing edge e is incident is edge-reconstructible.

NOTATION

Let H and G be two graphs. Then $\binom{G}{H}$ denotes the number of subgraphs of G isomorphic to H.

FACTS

Perhaps the single most useful result in reconstruction has proven to be this very simple lemma referred to as *Kelly's Lemma*.

F10: [Ke57] Let G and H be graphs with G having more vertices than H. Then $\binom{G}{H}$ is reconstructible from $\mathcal{D}(G)$. Similarly, if G has at least as many vertices as H and strictly more edges than H, then $\binom{G}{H}$ is reconstructible from $\mathcal{ED}(G)$.

Reconstructible Classes

Recall that graphs which do not have isolated vertices and are reconstructible are, by Greenwell's Theorem, also edge-reconstructible.

Also, one must keep in mind that, when we say that the class of graphs $\mathcal{C}$ is reconstructible, one is only given the deck or the edge-deck, and not the information that the graph to be reconstructed is in $\mathcal{C}$. Reconstruction here usually proceeds in two steps, first determining from the deck that the graph is in $\mathcal{C}$, then using this extra piece of information to prove that G is reconstructible. The following is a more exact definition of these two stages.

DEFINITIONS

D8: A class $\mathcal{C}$ of graphs is said to be ***recognizable*** or ***edge-recognizable*** if, for any graph $G \in \mathcal{C}$, any reconstruction, or edge-reconstruction, of G is also in $\mathcal{C}$. Equivalently, $\mathcal{C}$ is recognizable or edge-recognizable if it can be determined from $\mathcal{D}(G)$ or $\mathcal{ED}(G)$ whether or not G is in $\mathcal{C}$.

D9: A graph $G \in \mathcal{C}$ is said to be ***weakly reconstructible*** or ***weakly edge-reconstructible*** if any reconstruction, or edge-reconstruction, of G which is also in $\mathcal{C}$ is isomorphic to G. Equivalently, G is weakly reconstructible or weakly edge-reconstructible if it can be determined uniquely from the deck, or edge-deck, with the extra information that G is in $\mathcal{C}$.

This two-step process was essential in practically all proofs of reconstructibility of the following classes.

FACTS

F11: Regular graphs are reconstructible. (See [LaSc03].)

F12: Disconnected graphs are reconstructible. (See [LaSc03].)

F13: [Ke57] Trees are reconstructible.

F14: [Bo69b] Separable graphs (that is, graphs with connectivity 1) without vertices of degree 1 are reconstructible.

F15: [Zh88] The reconstruction conjecture is true if all 2-connected graphs are reconstructible.

F16: [Mc77] A computer search has shown that all graphs on nine or fewer vertices are reconstructible.

F17: [FiMa78, FiLa81, La81] Maximal planar graphs are reconstructible.

F18: [Gi76] Outerplanar graphs are reconstructible.

F19: [GoMc81] If all but at most one eigenvalue of G is simple and the corresponding eigenvectors are not orthogonal to the all-1's vector, then G is reconstructible. In particular, if G and its complement share no eigenvalue, then G is reconstructible.

F20: [Yu82] If there exists a subgraph $G - v$ of G none of whose eigenvectors is orthogonal to the all-1's vector, then G is reconstructible.

F21: [Fa94] Planar graphs with minimum degree at least 3 are edge-reconstructible.

F22: [Zh98a, Zh98b] Any graph of minimum degree 4 that triangulates a surface is edge-reconstructible. Any graph that triangulates a surface of characteristic at least 0 is edge-reconstructible. A graph G that triangulates a surface Σ of characteristic $\chi(\Sigma)$ is edge-reconstructible if $|V(G)| \geq -43\chi(\Sigma)$.

F23: [FaWuWa01] Series parallel networks (that is, 2-connected graphs without a subdivision of K_4) are edge-reconstructible.

F24: [Ch71] If a graph has property A_2 then it is reconstructible.

F25: [ElPyXi88] Claw-free graphs are edge-reconstructible.

F26: [MyElHo87] Bidegreed graphs are edge-reconstructible.

REMARKS

R3: A ***claw-free graph*** is one which has no induced subgraph isomorphic to $K_{1,3}$. This result made essential use of Nash–Williams' Lemma, which we shall discuss below.

R4: A ***bidegreed graph*** is a graph whose vertices can have only one of two possible degrees (the degrees have to be consecutive numbers, otherwise edge-reconstruction is trivial). The next step after this result would be the edge-reconstruction of tridegreed graphs (again, if the three degrees are not consecutive, then edge-reconstruction is easy). However, even the most elementary instance of this case, that is, degrees equal to 1, 2, and 3, seems to be extremely difficult to tackle [Sc85].

R5: As a next step, after the above series of results, attempting to reconstruct bipartite graphs seems worthwhile. However, no progress has been achieved to date either in reconstructing or edge-reconstructing bipartite graphs, in spite of the fact that, in Bondy and Hemminger's survey, this was even then suggested as a worthwhile problem.

R6: Several of the proofs of the above results involved long arguments very specific to the class of graphs under consideration, although in some cases common techniques began to emerge. But if these results are viewed as a step-wise attempt at solving the Reconstruction Problem for all graphs, then these results would simply be nibbles at a big mountain. What makes these results interesting, really, is the fact that use is made of the properties of such classes, and often new properties have to be unearthed. In Sections 4 and 5 we shall see sets of results which are more general and less specific to particular classes of graphs.

2.3.3 Reconstructing from Less than the Full Deck

The proofs of most of the reconstructibility results given above use much less information than is given by the full deck or edge-deck. This situation is best epitomized by the reconstruction of trees. Trees have been shown to be reconstructible by deleting only their endvertices (vertices of degree 1) [HaPa66], or only their peripheral vertices (vertices at maximum distance from the center of the tree) [Bo69a]. This immediately suggests, for graphs with many endvertices, reconstructibility from only the endvertex-deleted subgraphs.

Endvertex-Reconstruction

DEFINITION

D10: The ***endvertex-deck*** of a graph G is the collection of graphs $G - v$ for all vertices v with degree 1 in G. A graph G is ***endvertex-reconstructible*** if it is uniquely determined by its endvertex-deck.

FACTS

F27: [HaPa66] Trees are endvertex-reconstructible.

One natural question which arises is therefore whether a graph with a sufficiently large number of endvertices is necessarily endvertex-reconstructible. A negative result of Bryant, however, puts paid to any such hopes.

F28: [Br71] For any integer k there is a graph with k endvertices which is not endvertex-reconstructible.

However, this is not the end of the story for endvertex-reconstructibility. Toward the end of Section 5 we shall present a result which indicates that it is the proportion of endvertices in a graph that determines its endvertex-reconstructibility.

Reconstruction Numbers

Again noting that not all graphs in the deck are usually needed for reconstruction, Harary and Plantholt [HaPl85] introduced the definition of reconstruction numbers.

DEFINITIONS

D11: The ***reconstruction number*** of a graph G, denoted by $\mathrm{rn}(G)$, is the least number of subgraphs in the deck of G which guarantees that G is uniquely determined. The ***edge-reconstruction number***, denoted by $\mathrm{ern}(G)$, is analogously defined.

D12: Let $\mathcal{C}$ be a class of graphs. The ***class reconstruction number*** of a graph G in $\mathcal{C}$, denoted by $\mathcal{C}\mathrm{rn}(G)$, is the minimum number of subgraphs in the deck of G which, together with the information that G is in $\mathcal{C}$, guarantees that G is uniquely determined. The ***class edge-reconstruction number***, denoted by $\mathcal{C}\mathrm{ern}(G)$, is analogously defined.

Since almost every graph has property A_3, when discussing above the relationship between reconstruction and symmetries we have already met the first of the following results that answers in the positive a question raised by Harary and Plantholt in their paper.

FACTS

F29: [My88, Bo90] Almost every graph has reconstruction number equal to 3.

The next two results imply that there is no disconnected graph with c vertices in each component and reconstruction number equal to $c+1$. They also raise the natural question of investigating the gap between 3 and $c+1$ for the reconstruction number of disconnected graphs. Thus, let G be a disconnected graph consisting of a number of copies of H with $|V(H)| = c$. Determine the number $g = g(c)$ such that if $\mathrm{rn}(G) \geq g$ then $H = K_c$ but there is a G, with $H \neq K_c$, such that $\mathrm{rn}(G) = g - 1$. The last result above shows that $g \leq c$. Also, is there a constant g_0 such that if G is a disconnected graph with $\mathrm{rn}(G) \geq g_0$ then G must be a union of complete graphs?

F30: [My90] A disconnected graph with components not all isomorphic has reconstruction number 3. If all components are isomorphic and have c vertices each, then the reconstruction number can be equal to $c+2$.

F31: [AsLa02] If the reconstruction number of a disconnected graph is at least $c+1$ then G must consist of copies of K_c.

F32: [BaBaHo87] If $\mathcal{C}$ is the class of total graphs and G is in $\mathcal{C}$, then $\mathcal{C}\mathrm{rn}(G)$ equals 1.

F33: [My90] The reconstruction number of trees is 3.

Harary and Lauri [HaLa88] have conjectured that if $\mathcal{C}$ is the class of trees and T is a tree then $\mathcal{C}\mathrm{rn}(T)$ is at most 2.

F34: [HaLa87] If $\mathcal{C}$ is the class of maximal planar graphs and G is maximal planar then $\mathcal{C}\mathrm{rn}(G)$ is at most 2. Those maximal planar graphs with class reconstruction number equal to 1 are characterized.

F35: Almost every graph has edge-reconstruction number equal to 2. (See [LaSc03].)

F36: [Mo95] Let G be a disconnected graph. If G contains a pair of nontrivial, nonisomorphic components, then $\text{ern}(G)$ is at most 3. If, furthermore, G is not a forest and contains a component other than K_3 and $K_{1,3}$, then $\text{ern}(G)$ is at most 2. If the components of G are all isomorphic and contain k edges, then the edge-reconstruction number can be as high as $k + 2$.

Remarks analogous to those made above concerning the reconstruction number of disconnected graphs also can be made here.

F37: [Mo93] Every tree with at least 4 edges has edge-reconstruction number at most 3.

FURTHER REMARKS

R7: An intriguing question which has hardly been given any attention is the relationship between $\text{rn}(G)$ and $\text{ern}(G)$. While we have seen results which say that edge-reconstruction is implied by reconstruction, no such relationship seems to exist between these two parameters. In fact, the edge-reconstruction number for a graph could be greater than its reconstruction number.

R8: Reconstruction numbers are also interesting from another point of view. We have seen that lack of symmetry favors reconstruction, but whereas this should imply that high symmetry makes reconstruction more difficult, highly symmetric graphs are regular, and these are trivially reconstructible. Reconstruction numbers seem to put this in a better perspective because, while graphs with property A_3 have reconstruction number 3, it seems [My88] that regular graphs are the candidates for being the graphs with the largest reconstruction number. It should also be pointed out that the reconstruction number of regular graphs is not yet known.

R9: In her thesis [My88], Myrvold calls the reconstruction number the ***ally reconstruction number*** and she also defines another parameter which she calls the ***adversary reconstruction number***. If **A** and **B** are two players, the reconstruction number can be seen as the smallest number of graphs from the deck which **A** can give to **B** such that the latter can determine the graph uniquely; here **A** and **B** are allies. However, we can also ask for the largest number of subgraphs which **A** can give **B** such that **B** cannot determine the graph uniquely; here **A** and **B** are adversaries. The adversary reconstruction number of a graph is equal to 1 plus this last number. In other words, a graph G has adversary reconstruction number k if and only if any k subgraphs from the deck of G determine it uniquely; equivalently, no other graph has these same k subgraphs in its deck. Only partial results have been obtained on the adversary reconstruction number, which seems even more difficult to tackle than the (ally) reconstruction number, and this might perhaps be an area for further interesting research.

Set Reconstruction

In 1964, Harary [Ha64] suggested another way of reconstructing by not using the full deck when he made the following conjecture.

C3: [Set Reconstruction Conjecture] *Any graph G with $n \geq 4$ vertices can be reconstructed uniquely from its set of nonisomorphic subgraphs $G - v$.*

In other words, one is now only given *one* graph from each isomorphism class in the deck, and one does not know how many times each given graph appears in the deck.

DEFINITION

D13: A graph or a parameter which can be determined from the respective set of non-isomorphic subgraphs is said to be ***set reconstructible***.

FACTS

F38: [Ma76] The number of edges and the set of degrees of a graph is set reconstructible.

F39: [Ma76] For every graph in which no vertex of minimum degree lies on a triangle, the degree sequence is set reconstructible.

F40: [Ma76] The degree sequence of any graph with minimum degree at most 3 is set reconstructible.

F41: [Ma76] The connectivity of any graph is set reconstructible.

F42: [Ma76] Disconnected graphs are set reconstructible.

F43: Separable graphs (that is, graph of connectivity 1) without vertices of degree 1 are set reconstructible.

F44: [Ma70] Trees are set reconstructible.

F45: [Gi76] Outerplanar graphs are set reconstructible.

F46: [ArCo74] Unicyclic graphs (that is, graphs having only one cycle) are set reconstructible.

REMARK

R10: The idea of set reconstruction can also be applied to edge-reconstruction, that is, only one copy of each isomorphism type in the edge-deck is given. When a parameter or a class of graph is so reconstructible we say that it is ***set edge-reconstructible***. We highlight a few results in set edge-reconstructibility.

FACTS

F47: [Ma76] The degree sequence of a graph is set edge-reconstructible.

Delorme, Favaron, and Rautenbach have improved this result of Manvel.

F48: [DeFaRa02] The degree sequence of a graph with at least four edges is uniquely determined by the set of degree sequences of its edge-deleted subgraphs with one well-described class of exceptions. Moreover, the multiset of the degree sequences of the edge-deleted subgraphs determines the degree sequence of the graph.

F49: [AnDiVe96] If a graph G with at least four edges has at most two non-isomorphic edge-deleted subgraphs, then G is set edge-reconstructible.

Reconstruction from the Characteristic Polynomial Deck

In [Sc79], Schwenk proposed the problem of reconstructing a graph from the characteristic polynomial of each subgraph in the deck, which we shall call the ***polynomial deck***. He also showed that the answer to this general problem is in the negative, that is, a graph is not necessarily reconstructible from the polynomial deck. But he suggested a weakening of the problem so that what is required is the reconstruction of the characteristic polynomial of G from its polynomial deck. This problem is still open, and we here limit ourselves to presenting four results from the few which have been obtained.

FACTS

F50: [Sc79] The characteristic polynomial of any graph is reconstructible up to a constant from the polynomial deck.

F51: If a subgraph in the deck of G has a characteristic polynomial with repeated roots, then the characteristic polynomial of G is reconstructible from its polynomial deck. (See [LaSc03].)

F52: [CvLe98] The characteristic polynomial of a tree is reconstructible from its polynomial deck.

F53: [Sc02] If a graph of order n has at least $n/3$ vertices of degree 1 then its characteristic polynomial is reconstructible from its polynomial deck.

Reconstructing from k-Vertex-Deleted Subgraphs

A k-vertex-deleted subgraph of G is a subgraph obtained from G by deleting k of its vertices and all edges incident to them. We shall have more to say about this mode of reconstruction in Section 5 when we consider k-edge-deleted subgraphs. We here limit ourselves to one result.

F54: [Ta89] Let $k \geq 3$ be an integer. Then the degree sequences of all sufficiently large graphs are determined by their k-vertex-deleted subgraphs. In particular, this result is true for all graphs on at least $f(k)$ vertices, where $f(k)$ is a certain function which is asymptotic to ke.

2.3.4 Tutte's and Kocay's Results

If Kelly's Lemma were true for all spanning subgraphs of G (that is, subgraphs with the same number of vertices of G) then this would solve the reconstruction problem. In [Tu79], however, Tutte managed to show that Kelly's Lemma can be extended to certain classes of spanning subgraphs, and this has had very important consequences. In [Ko81], Kocay managed to obtain Tutte's results with proofs that were much easier. We shall here present a sketch of Kocay's method. A fuller treatment is given in [Bo91] and [LaSc03].

Kocay's Parameter

DEFINITION

D14: Let G be a graph and $\mathcal{F} = (F_1, F_2, \ldots, F_k)$ a sequence of graphs (we do not exclude the possibility that different F_i could be isomorphic). A ***cover*** of G by $\mathcal{F}$ is a sequence $\mathcal{G} = (G_1, G_2, \ldots, G_k)$ of subgraphs of G (not necessarily distinct) such that: (i) $G_i \simeq F_i,\ i = 1, \ldots, k$ and (ii) $G = \cup_i G_i$. The number of covers of G by $\mathcal{F}$ is denoted by $c(\mathcal{F}, G)$.

FACTS

[Tu79, Ko81] (See also [Bo91, LaSc03].)

F55: Let G be a graph and let $\mathcal{F} = (F_1, F_2, \ldots, F_k)$ be a sequence of graphs with each $|V(F_i| < |V(G)|$. Let $\kappa(\mathcal{F}, G)$ be the parameter defined by

$$\sum_X c(\mathcal{F}, X)\binom{G}{X}$$

where the summation is taken over all isomorphism types X of graphs such that $|V(X)| = |V(G)|$. Then $\kappa(\mathcal{F}, G)$ is reconstructible.

The following results are then obtained by defining a suitable choice for the F_i.

F56: The number of 1-factors of G is reconstructible.

F57: The number of spanning trees of G is reconstructible.

F58: The number of Hamiltonian cycles of G is reconstructible.

F59: The number of 2-connected spanning subgraphs of G with a specified number of edges is reconstructible.

Characteristic and Chromatic Polynomials

DEFINITIONS

D15: An ***elementary graph*** is a graph in which every component is either an edge or a cycle. For any graph X, $c(X)$ denotes the number of components of X and $s(X)$ the number of cycles.

FACTS

The proofs of the first two of the following results, due to Sachs and Whitney, respectively, can be found in [Bi93].

F60: Let the characteristic polynomial of G be

$$\lambda^n + a_1\lambda^{n-1} + a_2\lambda^{n-2} + \ldots + a_n$$

Then each coefficient a_i is given by

$$a_i = \sum_X (-1)^{c(X)} 2^{s(X)} \binom{G}{X}$$

where the summation extends over all isomorphism types X of elementary graphs on i vertices.

F61: Let the chromatic polynomial of G be

$$b_1x + b_2x^2 + \ldots + b_nx^n$$

Then each coefficient b_i is given by

$$b_i = \sum_X (-1)^{|E(X)|} \binom{G}{X}$$

where the summation extends over all isomorphism types X of graphs on n vertices and i components.

From these characterizations of the characteristic and chromatic polynomials together with the previous reconstruction results, the next important results follow. [Tu79, Ko81] (See also [Bo91, LaSc03].)

F62: The characteristic polynomial is reconstructible.

F63: The chromatic polynomial is reconstructible.

2.3.5 Lovász's Method and Nash–Williams's Lemma

It is quite arguable that the deepest and most general results obtained in reconstruction are those which we shall be presenting in this section. In 1972, Lovász [Lo72] published a beautiful two-page paper in which he showed that if a graph has one half more than the largest possible number of edges, then it is edge-reconstructible. This paper made a surprising and elementary use of the inclusion-exclusion principle. In 1977, Müller [Mü77], using the same method as Lovász, obtained a stronger conclusion for edge-reconstructibility. In his survey of 1978 [Na78], Nash–Williams proved a lemma from which Lovász's and Müller's results follow (but still applying the method introduced by Lovász). We shall here give these results. A more extended treatment can be found in [El88] or [Bo91] or [LaSc03].

The Nash–Williams Lemma

DEFINITIONS

In the following, let G and H be graphs which are assumed to share the same vertex-set V, and let X be a subset of the edge-set of G.

D16: A ***homomorphism*** from G to H is a permutation of V such that any edge of G is mapped into an edge of H. The number of such homomorphisms is denoted by $[H]_G$.

D17: A ***homomorphism with forbidden*** X is a permutation of V such that all edges in $E(G) - X$ are mapped into edges of H but all edges in X are mapped into non-edges in H. The number of such homomorphisms is denoted by $[H]_{G\setminus X}$.

REMARK

R11: Note that $[H]_{G\setminus X}$ is quite different from $[H]_{G-X}$; the latter counts homomorphisms from $G-X$ to H, that is, where all edges in $E(G)-X$ are mapped into edges of H, but all edges in X can be mapped either into edges or non-edges of H; this is unlike the case of $[H]_{G\setminus X}$ where all the edges of X must be mapped into non-edges of H.

FACTS

F64: [Lo72] Let G, H, and X be as above. Then

$$[H]_{G\setminus X} = \sum_{Y\subset X} (-1)^{|Y|}[H]_{G-X+Y}$$

F65: *Nash–Williams' Lemma* Let G, H and X be as above, and suppose that G and H have the same edge-deck. Then

$$[H]_G = |\mathrm{Aut}(G)| + (-1)^{|X|}([H]_{G\setminus X} - [G]_{G\setminus X})$$

(See [Bo91,LaSc03].)

The following is an important corollary to Nash–Williams's Lemma. (See [Bo91, LaSc03].)

F66: Suppose G, H and X are as in Nash–Williams's Lemma, and assume that $G \not\simeq H$. Then,

(i) if $|X|$ is odd, then $[H]_{G\setminus X} > 0$;
(ii) if $|X|$ is even, then $[G]_{G\setminus X} > 0$.

From this, Lovász and Müller's results follow.

F67: [Lo72] Let G be a graph such that $|E(G)| > \binom{n}{2}/2$. Then G is edge-reconstructible.

F68: [Mü77] Let G be a graph such that $2^{|E(G)|-1} > n!$. Then G is edge-reconstructible.

Perhaps the most striking result in edge-reconstruction obtained by these methods is the following.

F69: [Py90] A Hamiltonian graph with a sufficiently large number of vertices is edge-reconstructible.

Structures Other than Graphs

The real power and generality of the above methods appear with the realization that edge-reconstruction can be generalized to the reconstruction, up to some group of isomorphisms, of a combinatorial object or structure from its subobjects, again given up to isomorphism.

DEFINITION

D18: More exactly, define a ***structure*** to be a triple (D, Γ, E) where D is a finite set, Γ is a group of permutations acting on D, and E is a subset of D. By edge-reconstruction we mean here that the subsets $E - x$ are given, up to 'translation' by the group Γ, and the question is whether E can be reconstructed uniquely, again up to action by the group Γ.

Therefore in edge-reconstruction for graphs, D would be the set of all possible $\binom{n}{2}$ edges on n vertices, E would be the edges which define the graph to be reconstructed, and Γ would be the full symmetric group with its induced action on the unordered, distinct pairs of vertices.

An extended treatment of edge-reconstruction seen in this light can be found in [Bo91, LaSc03].

FACTS

We shall here present some results obtained by viewing edge-reconstruction in this more general setting. The first result, although a straightforward application of Nash–Williams' Lemma to structures, gives some life to the problem of reconstructing from the endvertex-deck.

F70: [LaSc03] Let H be a graph with minimum degree 2, and let G be obtained from H by adding k endvertices such that no two have a common neighbor. Then G is endvertex-reconstructible if either $k > V|H|/2$ or $2^{k-1} > \mathrm{Aut}(H)$.

REMARK

R12: This result in conjunction with Bryant's negative result leads to the natural question asking what is the minimum proportion of endvertices required to guarantee endvertex-reconstructibility.

DEFINITION

D19: The following generalizes Müller's result not only to structures, but also in this fashion: instead of removing one edge at a time, k edges at a time are removed. Let us call this ***k-edge-reconstruction***.

FACTS

F71: [AlCaKrRo89] Let (D, Γ, E) be a structure such that $2^{|E|-k} > |\Gamma|$. Then the structure is k-edge-reconstructible.

The paper [AlCaKrRo89] contains several other results of this type and should be studied carefully by anyone who is interested in extending the reconstruction of structures in the direction of k-edge-reconstruction. Many of the results in this paper have been extended more recently by Radcliffe and Scott. They consider the reconstruction of a subset of the integers modulo n, Z_n, or of the reals, R, up to translation from the collection of its subsets of a given size, also given up to translation. The following summarizes their important results.

F72: [RaSc98] Suppose p is prime. The every subset of Z_p is reconstructible from the collection of its 3-subsets.

F73: [RaSc98] For arbitrary n, almost all subsets of Z_n are reconstructible from the collections of their 3-subsets.

F74: [RaSc98] For any n, every subset of Z_n is reconstructible from its $9\alpha(n)$-subsets, where $\alpha(n)$ is the number of distinct prime factors of n.

F75: [RaSc99] A locally finite subset of R (that is, a subset which contains only finitely many translates of any given finite set of size at least 2) is reconstructible from its 3-subsets.

Another recent result of this type is the following. Here subsets of R^2 are considered, and any two such subsets are considered isomorphic if one can be transformed into the other by a translation or a rotation by a multiple of 90 degrees.

F76: [Ra02] Any finite subset A of the plane R^2 is uniquely determined by at most 5 of its subsets of cardinality $|A|-1$, given up to isomorphism; that is, in the terminology of graph reconstruction, A has reconstruction number 5.

The Reconstruction Index of Groups

Looking at edge reconstruction in this guise, that is, as reconstruction of structures (D, Γ, E), has led some authors to focus attention more directly on the permutation group Γ.

DEFINITION

D20: The ***reconstruction index*** $\rho(\Gamma, D)$ of the permutation group Γ acting on D is the smallest t such that for any $E \subset D$ with $|E| \geq t$, the structure (D, Γ, E) is edge-reconstructible.

The Edge-Reconstruction Conjecture therefore states that, if $Y = \{1, 2, \ldots, n\}$ and D is the set of unordered, distinct pairs of Y, and if $S_n^{(2)}$ is the symmetric group of Y acting on these pairs, then $\rho(S_n^{(2)}, D) = 4$.

FACT

The following is one result obtained on the reconstruction index of permutation groups.

F77: [Mn98] The reconstruction index of an abelian group is 4 and the reconstruction index of Hamiltonian groups is 5.

Other works which deal with the reconstruction index of groups are [Ca96, Ma96, Mn87, Mn92, Mn95]

2.3.6 Digraphs

The situation with the reconstruction of digraphs is quite different from that of graphs, for here it has been shown that the conjecture is false.

FACTS

F78: [St77, Ko85] There exists an infinite family of tournaments which are not reconstructible.

A positive partial result has been obtained by Harary and Palmer.

F79: [HaPa67] Tournaments on at least five vertices which are not strongly connected are reconstructible. (See also [BoHe77].)

Some more positive results on the reconstructibility of tournaments can be found in [DeGu90, Gu96, Vi99].

Therefore the problem here should be to investigate which digraphs are reconstructible, or to determine what reconstruction question one should ask for digraphs. In fact, Ramachandran has noted that all non-reconstructible digraphs which have been discovered to date have the property that they would be reconstructible if, with every $D - v$, one is also given the in-degree and the out-degree in D of the missing vertex v.

DEFINITION

D21: [Ra97] A digraph D is said to be ***N*-reconstructible** if it is uniquely determined by the triples $(D - v_i, \deg_{\text{in}}(v_i), \deg_{\text{out}}(v_i))$, for all vertices v_i of D.

And Ramachandran goes on to make the following conjecture.

C4: [The N-Reconstruction Conjecture for Digraphs] *Every digraph is N-reconstructible.*

No counterexamples to this conjecture are known.

2.3.7 Illegitimate Decks

As we have already said, the reconstruction problem is not about finding an efficient algorithm to determine G from its deck, but the question is one of uniqueness: is there only one graph with the given deck?

However, there is one problem in graph reconstruction which falls naturally within the setting of computational complexity.

DEFINITION

D22: A collection of graphs $G_1, G_2, \ldots, G_n$ each on $n - 1$ vertices is said to be an ***illegitimate deck*** if there is no graph G having the given collection as deck. The ***illegitimate deck problem*** is to determine whether or not such a given collection of graphs is indeed the deck of some graph.

FACTS

F80: [Ma82] Determining whether a given collection of graphs is an illegitimate deck is at least as hard as the isomorphism problem.

F81: [HaPlSt82] The graph isomorphism problem is polynomially equivalent to the illegitimate deck problem for regular graphs.

More information about the relationship between the computational complexities of the legitimate deck problem and the graph isomorphism problem can be found in [KöScTo93, KrHe94].

2.3.8 Recent Results

We briefly outline the most important results in reconstruction obtained since the publication of the *Handbook of Graph Theory, First Edition*. Notably, Asciak, Francalanza, Lauri and Myrvold [AsFrLaMy10] have given a survey of open questions regarding reconstruction numbers. At about the same time, Bowler, Brown, and Fenner have commenced a systematic attack on the adversary reconstruction number.

FACTS

F82: [BiKwYu07] The class of planar graphs is recognizable.

F83: [BoBrFe10] There exist pairs of graphs on n vertices with $2\lfloor\frac{1}{3}(n-1)\rfloor$ common cards for every $n \geq 10$.

F84: There exist pairs of trees on n vertices with $2\lfloor\frac{1}{3}(n-5)\rfloor$ cards in common for $n \geq 8$.

F85: [BoBrFeMy11] The maximum number of cards in common between a connected and a disconnected graph on n vertices is $\lfloor\frac{n}{2}\rfloor+1$. Graphs that attain this upper bound are characterized.

F86: [AsLa10] If G is a connected graph all of whose components are isomorphic and contain k edges, and if G has edge-reconstruction number at least $k+1$, then the components of G are isomorphic to the star $K_{1,k}$.

CONJECTURES

C5: [BoBrFe10] The largest possible number of cards that two graphs on n vertices can have in common is $2\lfloor\frac{1}{3}(n-1)\rfloor$. Affirming this would imply that any such graph can be reconstructed from any $2\lfloor\frac{1}{3}(n-1)\rfloor+1$ of its cards.

C6: [BaWe10] They also conjecture that $\mathrm{drn}(T) \leq 2$ for all but finitely many trees T.

C7: [AsLa10] For a disconnected graph all of whose components are isomorphic to H, suppose that $\mathrm{ern}(G) > 3$. Then H is isomorphic to the star $K_{1,k}$.

REMARK

R13: The reconstruction number and the edge-reconstruction number of all graphs with $n \leq 11$ vertices are given by [RiRa11], along with computational results on reconstruction numbers associated with the removal of more than one vertex at a time.

DEFINITION

D23: The ***degree associated reconstruction number*** $\mathrm{drn}(G)$ of a graph G is the minimum number of cards required to reconstruct G if, along with any card $G-v$, one is also given the degree of the missing vertex v. Note that $\mathrm{drn}(G)$ is equivalent to the class reconstruction number of graphs with a given value of m as the number of edges.

FACTS

F87: [Ra06] If G is a connected non-regular graph and kG is the disconnected graph made up of k copies of G then $\mathrm{drn}(kG) \leq 1 + darn(G)$. If G is r-regular of order $n > 2$ then $\mathrm{drn}(kG) \leq n + 2 - r$.

F88: [BaWe10] The degree associated reconstruction number equals 2 for all caterpillars, except stars and one 6-vertex example.

References

[AlCaKrRo89] N. Alon, Y. Caro, I. Krasikov, and Y Roditty, Combinatorial reconstruction problems, *J. Combin. Theory (Ser. B)* 47:153–161, 1989.

[AnDiVe96] L. D. Andersen, S. Ding, and P. D. Vestergaard, On the set edge-reconstruction conjecture, *J. Combin. Math. Combin. Comput.* 20:3–9, 1996.

[ArCo74] E. Arjomandi and D. G. Corneil, Unicyclic graphs satisfy Harary's conjecture, *Canad. Math. Bull.* 17:593–596, 1974.

[AsFrLaMy10] K. J. Asciak, M. A. Francalanza, J. Lauri, and W. Myrvold, A survey of some open questions in reconstruction numbers, *Ars Combin.* 97:443–456, 2010.

[AsLa02] K. J. Asciak and J. Lauri, On disconnected graphs with large reconstruction number, *Ars Combin.* 62:173–181, 2002.

[AsLa10] K. J. Asciak and J. Lauri, On the edge-reconstruction number of disconnected graphs, *Bull. ICA* 63:87-110, 2011.

[BaBaHo87] D. W. Bange, A. E. Barkauskas, and L. H. Host, Class-reconstruction of total graphs, *J. Graph Theory* 11:221–230, 1987.

[BaWe10] Degree-associated reconstruction number of graphs, *Discrete Math.* 310:2600–2612, 2010.

[Bi93] N. L. Biggs, *Algebraic Graph Theory*, Cambridge University Press, 1993.

[BiKwYu07] M. Bilinski, Y. S. Kwon, and X. Yu, On the reconstruction of planar graphs, *J. Combin. Theory Ser. B* 97:745–756, 2007.

[Bo90] B. Bollabás, Almost every graph has reconstruction number 3, *J. Graph Theory* 14:1–4, 1990.

[Bo69a] J. A. Bondy, On Kelly's congruence theorem for trees, *Proc. Camb. Phil. Soc.* 65:387–397, 1969.

[Bo69b] J. A. Bondy, On Ulam's conjecture for separable graphs, *Pacific J. Math.* 31:281–288, 1969.

[Bo91] J. A. Bondy, A graph reconstructor's manual, In *Surveys in Combinatorics* (A. D. Keedwell, ed.), 221–252. Cambridge University Press, 1991.

[BoHe77] J. A. Bondy and R. L. Hemminger, Graph reconstruction—A survey, *J. Graph Theory* 1:227–268, 1977.

[BoBrFe10] A. Bowler, P. Brown, and T. Fenner, Families of pairs of graphs with a large number of common cards, *J. Graph Theory* 63:146–163, 2010.

[BoBrFeMy11] A. Bowler, P. Brown, T. Fenner, and W. Myrvold, Recognizing connectedness from vertex-deleted subgraphs, *J. Graph Theory* 67:285–299, 2011.

[Br71] R. M. Bryant, On a conjecture concerning the reconstruction of graphs, *J. Combin. Theory* 11:139–141, 1971.

[Ca96] P. J. Cameron, Stories from the age of reconstruction, *Congr. Num.* 113:31–41, 1996.

[Ch71] P. Z. Chinn, A graph with p points and enough distinct $p-2$-order subgraphs is reconstructible, In *Recent Trends in Graph Theory*, M. Capobianco, J. B. Frechen, and M. Krolik, editors, 71–73. Springer-Verlag, 1971.

[CvLe98] D. M. Cvetković and M. Lepović, Seeking counterexamples to the reconstruction conjecture for the characteristic polynomial of graphs and a positive result, *Bull. Cl. Sci. Math. Nat. Sci. Math.* 23:91–100, 1998.

[DeFaRa02] C. Delorme, O. Favaron, and D. Rautenbach, On the reconstruction of the degree sequence, *Discrete Math.* 259:293–300, 2002.

[DeGu90] D. C. Demaria and C. Guido, On the reconstruction of normal tournaments, *J. Combin. Inform. System Sci.*15(1–4):301–323, 1990.

[El88] M. N. Ellingham, Recent progress in edge reconstruction, *Congr. Numer.* 62:3–20, 1988.

[ElPyXi88] M. N. Ellingham, L. Pyber, and Y. Xingxing, Claw-free graphs are edge reconstructible, *J. Graph Theory* 12:445–451, 1988.

[Fa94] H. Fan, Edge reconstruction of planar graphs with minimum degree at least three—IV, *Systems Sci. Math. Sci.* 7:218–222, 1994.

[FaWuWa01] H. Fan, Y-L. Wu, and C. K. Wang, On fixed edges and the edge reconstruction of series parallel networks, *Graphs Combin.* 17(2):213–225, 2001.

[FiLa81] S. Fiorini and J. Lauri, The reconstruction of maximal planar graphs, I: Recognition, *J. Combin. Theory (Ser. B)* 30:188–195, 1981.

[FiMa78] S. Fiorini and B. Manvel, A theorem on planar graphs with an application to the reconstruction problem, II, *J. Combin. Inf. Sys. Sci.* 3(3):200–216, 1978.

[Gi76] W. B. Giles, Point deletions of outerplanar blocks, *J. Combin. Theory (Ser. B)* 20:103–116, 1976.

[GoMc81] C. D. Godsil and B. D. McKay, Spectral conditions for reconstructibility of a graph, *J. Combin. Theory (Ser. B)* 30:285–289, 1981.

[Gr71] D. L. Greenwell, Reconstructing graphs, *Proc. Amer. Math. Soc.* 30:431–433, 1971.

[Gu96] C. Guido, A larger class of reconstructible tournaments, *Discrete Math.* 152(1–3):171–184, 1996.

[Ha64] F. Harary, On the reconstruction of a graph from a collection of subgraphs, In M. Fiedler, Editor, *Theory of Graphs and Its Applications* Proc. Symposium Smolenice, 1963, pages 47–52, Academic Press, 1964.

[HaLa87] F. Harary and J. Lauri, The class-reconstruction number of a maximal planar graph, *Graphs and Combinatorics* 3:45–53, 1987.

[HaLa88] F. Harary and J. Lauri, On the class-reconstruction number of trees, *Quart. J. Math. Oxford (2)* 39:47–60, 1988.

[HaPa66] F. Harary and E. M. Palmer, The reconstruction of a tree from its maximal subtrees, *Canad. J. Math.* 18:803–810, 1966.

[HaPa67] F. Harary and E. M. Palmer, On the problem of reconstructing a tournament from subtournaments, *Monatsh. Math.* 71:14–23, 1967.

[HaPl85] F. Harary and M. Plantholt, The graph reconstruction number, *J. Graph Theory* 9:451–454, 1985.

[HaPlSt82] F. Harary, M. Plantholt, and R. Statman, The graph isomorphism problem is polynomially equivalent to the legitimate deck problem for regular graphs, *Caribbean J. Math.* 1(1):15–23, 1982.

[He69] R. L. Hemminger, On reconstructing a graph, *Proc. Amer. Math. Soc.* 20:185–187, 1969.

[Ke57] P. J. Kelly, A congruence theorem for trees, *Pacific J. Math.* 7:961–968, 1957.

[KöScTo93] J. Köbler, U. Schöning, and J. Torán, *The Graph Isomorphism Problem: Its Structural Complexity*, Birkhäuser, 1993.

[Ko81] W. L. Kocay, On reconstructing spanning subgraphs, *Ars Combinatoria* 11:301–313, 1981.

[Ko85] W. L. Kocay, On Stockmeyer's non-reconstructible tournaments, *J. Graph Theory* 9:473–476, 1985.

[Ko71] A. D. Korshunov, Number of nonisomorphic graphs in an n-point graph, *Math. Notes of the Acad. of Sciences of the USSR* 9:155–160, 1971.

[KrLeTh02] I. Krasikov, A. Lev, and B. D. Thatte, Upper bounds on the automorphism group of a graph, *Discrete Math.* 256(1–2):489–493, 2002.

[KrHe94] D. Kratsch and L. A. Hemaspaandra. On the complexity of graph reconstruction, *Math. Systems Theory* 27(3):257–273, 1994.

[La81] J. Lauri, The reconstruction of maximal planar graphs II: Reconstruction, *J. Combin. Theory (Ser. B)* 30:196–214, 1981.

[La87] J. Lauri, Graph reconstruction—some new techniques and new problems, *Ars Combinatoria (Ser B)* 24:35–61, 1987.

[LaSc03] J. Lauri and R. Scapellato, *Topics in Graph Automophisms and Reconstruction* Cambridge University Press, 2003.

[Lo72] L. Lovász, A note on the line reconstruction problem, *J. Combin. Theory (Ser. B)* 13:309–310, 1972.

[Ma82] A. Mansfield, The relationship between the computational complexities of the legitimate deck and isomorphism problems, *Quart. J. Math. Oxford (2)* 33:345–347.

[Ma70] B. Manvel, Reconstruction of trees, *Canad. J. Math.* 22:55–60, 1970.

[Ma76] B. Manvel, On reconstructing graphs from their sets of subgraphs, *J. Combin. Theory (Ser. B)* 21:156–165, 1976.

[Ma88] B. Manvel, Reconstruction of graphs: progress and prospects, *Congr. Numer.* 63:177–187, 1988.

[Ma96] P. Maynard, *On Orbit Reconstruction Problems*, PhD thesis, University of East Anglia, Norwich, 1996.

[Mc77] B. D. McKay, Computer reconstruction of small graphs, *J. Graph Theory* 1:281–283, 1977.

[Mn87] V. B. Mnukhin, Reconstruction of k-orbits of a permutation group, *Math. Notes* 42:975–980, 1987.

[Mn92] V. B. Mnukhin, The k-orbit reconstruction and the orbit algebra, *Acta. Applic. Math.* 29:83–117, 1992.

[Mn95] V. B. Mnukhin, The reconstruction of oriented necklaces, *J. Combin., Inf. & Sys. Sciences* 20(1–4):261–272, 1995.

[Mn98] V. B. Mnukhin, The k-orbit reconstruction for abelian and hamiltonian groups, *Acta. Applic. Math.* 52:149–162, 1998.

[Mo93] R. Molina, The edge reconstruction number of a tree, *Vishwa Int. J. Graph Theory* 2(2):117–130, 1993.

[Mo95] R. Molina, The edge reconstruction number of a disconnected graph, *J. Graph Theory* 19(3):375–384, 1995.

[Mü76] V. Müller, Probabilistic reconstruction from subgraphs, *Comment. Math. Univ. Carolinae* 17:709–719, 1976.

[Mü77] V. Müller, The edge reconstruction hypothesis is true for graphs with more than $n \log_2 n$ edges, *J. Combin. Theory (Ser. B)* 22:281–283, 1977.

[My88] W. J. Myrvold, *Ally and Adversary Reconstruction Problems*, PhD thesis, University of Waterloo, 1988.

[My89] W. J. Myrvold, The ally-reconstruction number of a disconnected graph, *Ars Combinatoria* 28:123–127, 1989.

[My90] W. J. Myrvold, The ally-reconstruction number of a tree with five or more vertices is three, *J. Graph Theory* 14:149–166, 1990.

[MyElHo87] W. J. Myrvold, M. N. Ellingham, and D. G. Hoffman, Bidegreed graphs are edge reconstructible, *J. Graph Theory* 11(3):281–302, 1987.

[Na78] C. St. J. A. Nash–Williams, The reconstruction problem. In L. W. Beineke and R. J. Wilson, editors, *Selected Topics in Graph Theory*, Chapter 8, Academic Press, London, 1978.

[Py90] L. Pyber, The edge-reconstruction of hamiltonian graphs, *J. Graph Theory* 14:173–179, 1990.

[RaSc98] A. J. Radcliff and A. D. Scott, Reconstructing subsets of Z_n, *J. Combin. Theory (Ser. A)* 83(2):169–187, 1998.

[RaSc99] A. J. Radcliff and A. D. Scott, Reconstructing subsets of reals, *Electronic J. Combin.* (1):Research Paper 20, 7pp., 1999.

[Ra97] S. Ramachandran, N-reconstructibility of nonreconstructible tournaments, *Graph Theory Notes of N.Y.* 32:23–29, 1997.

[Ra06] S. Ramachandran, Reconstruction number for Ulam's conjecture, *Ars Combin.* 78:289–296, 2006.

[Ra02] D. Rautenbach, On a reconstruction problem of Harary and Manvel, *J. Combin. Theory (Ser. A)* 99:32–39, 2002.

[RiRa09] D. Rivshin and S. Radziszowski, The vertex and edge graph reconstruction numbers of small graphs, *Australas. J. Combin* 45:175–188, 2009.

[RiRa11] D. Rivshin and S. Radziszowski, Multi-vertex deletion graph number reconstructions, *J. Combin. Math. Combin. Comput.* 78:303–321, 2011.

[Sc85] J. Schönheim, personal communication.

[Sc79] A. J. Schwenk, Spectral reconstruction problems, in *Topics in Graph Theory*, Vol 328 of Annals New York Academy of Science, pages 183–189, New York Academy of Sciences, 1979.

[Sc02] I. Sciriha, Polynomial reconstruction and terminal vertices, *Linear Algebra and its Applications* 356:145–156, 2002.

[St77] P. K. Stockmeyer, The falsity of the reconstruction conjecture for tournaments, *J. Graph Theory* 1:19–25, 1977.

[Ta89] R. Taylor, Reconstructing degree sequences from k-vertex-deleted subgraphs, *Discrete Maths.* 79:207–213, 1989/90.

[Tu79] W. T. Tutte, All the king's horses—a guide to reconstruction, in J. A. Bondy and U. S. R. Murty, editors, *Graph Theory and Related Topics*, Academic Press, 1979.

[Ul60] S. M. Ulam, *A Collection of Mathematical Problems*, Wiley (Interscience), New York, 1960.

[Vi99] P. Vitolo, The reconstruction of simply disconnected tournaments, *J. Combin. Inform. System Sci.* 24(2–4):65–77, 1999.

[Yu82] H. Yuan, An eigenvector condition for reconstructibility, *J. Combin. Theory (Ser. B)* 32:245–256, 1982.

[Zh98a] Y. Zhao, On the edge reconstruction of graphs embedded in surfaces. II, *J. London Math. Soc.* 57:268–274, 1998.

[Zh98b] Y. Zhao, On the edge reconstruction of graphs embedded in surfaces. III, *J. Combin. Theory (Ser. B)* 74:302–310, 1998.

[Zh88] Yang Yong Zhi, The reconstruction conjecture is true if all 2-connected graphs are isomorphic, *J. Graph Theory* 12:237–243, 1988.

Section 2.4
Recursively Constructed Graphs

Richard B. Borie, University of Alabama
R. Gary Parker, Georgia Institute of Technology
Craig A. Tovey, Georgia Institute of Technology

INTRODUCTION

The core idea of recursively constructed graphs is captured in Definition 1, but the substantial literature on the subject has motivated a considerable breadth and variety of notational distinctions.

NOTATION: All graphs in this section are simple, and an edge with endpoints x and y is denoted (x, y).

DEFINITIONS

D1: A ***recursively constructed graph class*** is defined by a set (usually finite) of primitive or ***base graphs***, in addition to one or more operations that compose larger graphs from smaller subgraphs. Each operation involves either fusing specific vertices from each subgraph or adding new edges between specific vertices from each subgraph.

D2: Each graph in a recursive class has a corresponding ***decomposition tree*** that shows how to build it from base graphs.

REMARK

R1: Graphs in these classes possess a modular structure, so fast algorithms can often be designed to solve hard problems restricted to these classes. The algorithms typically proceed by solving the desired problem on the base graphs, then employ dynamic programming to combine solutions for small subgraphs into a solution for a larger graph. The construction of these algorithms is the subject of Section 10.4.

2.4.1 Some Parameterized Families of Graph Classes

Trees

DEFINITION

D3: The graph with a single vertex r (and no edges) is a ***tree*** with root r (the sole base graph). Let (G, r) denote a tree with root r. Then $(G_1, r_1) \oplus (G_2, r_2)$ is a tree formed by taking the disjoint union of G_1 and G_2 and adding an edge (r_1, r_2). The root of this new tree is $r = r_1$.

TERMINOLOGY NOTE: Technically, the pairs (G, r) in Definition D3 denote *rooted trees*. However, the specification of distinguished vertices r_1 and r_2 (and hence r) is relevant here only as a vehicle in the recursive construction.

EXAMPLE

E1: Figure 2.4.1 illustrates the recursive construction of trees.

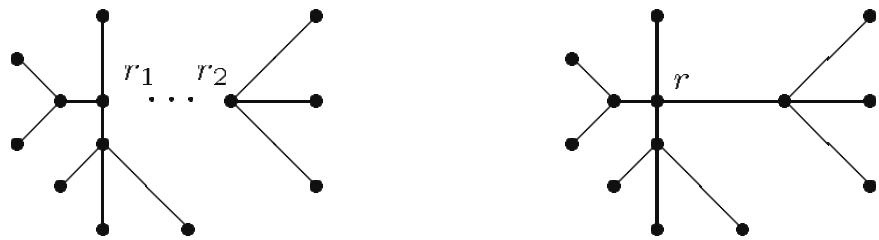


Figure 2.4.1: Recursive construction of a tree.

Series-Parallel Graphs

From a non-recursive perspective, a graph is series-parallel if it has no subgraph homeomorphic to K_4 [Du65]. The graph on the left of Figure 2.4.2 is not series-parallel; the offending subgraph is identified by bold edges. Removal of two edges, as indicated, yields the graph to the right which is series-parallel.

Figure 2.4.2: Non-series-parallel and series-parallel graphs.

Following, we give a recursive definition of this class.

DEFINITION

D4: A ***series-parallel graph*** with distinguished ***terminals*** l and r is denoted (G, l, r) and is defined recursively as follows:

- The graph consisting of a single edge (v_1, v_2) is a series-parallel graph (G, l, r) with $l = v_1$ and $r = v_2$.
- The ***series operation*** $(G_1, l_1, r_1) \odot_s (G_2, l_2, r_2)$ forms a series-parallel graph by identifying r_1 with l_2. The terminals of the new graph are l_1 and r_2.
- The ***parallel operation*** $(G_1, l_1, r_1) \odot_p (G_2, l_2, r_2)$ forms a series-parallel graph by identifying l_1 with l_2 and r_1 with r_2. The terminals of the new graph are l_1 and r_1.
- The ***jackknife operation*** $(G_1, l_1, r_1) \odot_j (G_2, l_2, r_2)$ forms a series-parallel graph by identifying r_1 with l_2; the new terminals are l_1 and r_1.

COMPUTATIONAL NOTE: The jackknife operation can also be specified where the new terminals, after composition, are defined to be l_1 and l_2.

EXAMPLE

E2: The three operations defining series-parallel graphs are demonstrated in Figure 2.4.3. The pair-specific composition is on the left; the result is shown to the right. Terminal vertices are circled and labeled.

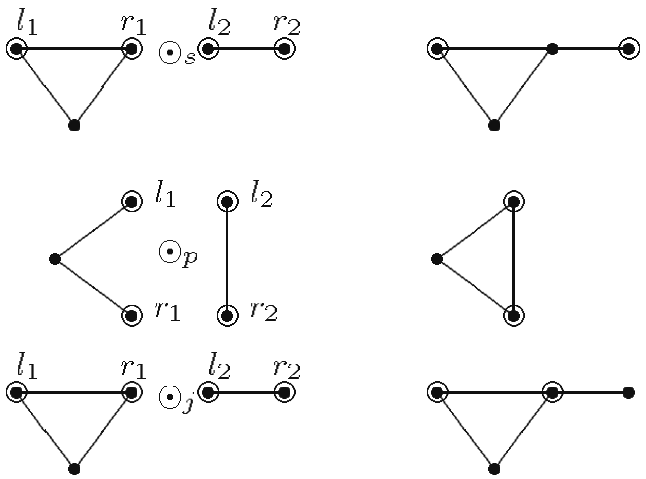


Figure 2.4.3: Composition operations for series-parallel graphs.

k-Trees and Partial k-Trees

DEFINITIONS

D5: The k-vertex complete graph, K_k, is a ***k-tree***. A k-tree with $n+1$ vertices ($n \geq k$) is constructed from a k-tree on n vertices by adding a vertex adjacent to all vertices of one of its K_k subgraphs, and only to those vertices.

D6: A ***partial k-tree*** is a subgraph of a k-tree.

TERMINOLOGY NOTE: In a given construction of a k-tree, the original K_k subgraph is referred to as its ***basis***.

D7: A graph is ***chordal*** (or ***triangulated***) if it contains no induced cycles of length greater than 3.

D8: A graph is ***perfect*** if every induced subgraph has chromatic number equal to the size of its maximum clique.

FACTS

F1: Trees are 1-trees, and forests are partial 1-trees.

F2: Series-parallel graphs are partial 2-trees.

F3: Any K_k subgraph of a k-tree can act as its basis.

F4: All k-trees are chordal graphs and, hence, perfect (because every chordal graph is perfect).

EXAMPLES

E3: A 3-tree is shown on the left in Figure 2.4.4, and a partial 3-tree is shown to the right.

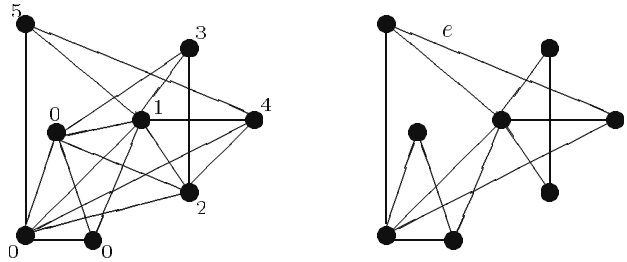


Figure 2.4.4: Construction of a 3-tree and a partial 3-tree.

Demonstrated by the graph to the left in Figure 2.4.4 is the "creation" of a 3-tree following a small number of composition operations starting from the basis given by an initial K_3 identified by vertex labels of 0. At each step, a new (consecutively labeled) vertex is added. Observe that if edge e is eliminated from the graph on the right in Figure 2.4.4, a partial 2-tree is created.

E4: The graph on the left in Figure 2.4.5 is series-parallel; it is a subgraph (and hence a partial 2-tree) of the 2-tree on the right. The dotted edges complete the 2-tree where the construction is verified by the labels on the vertices that are interpreted similarly as for Figure 2.4.4.

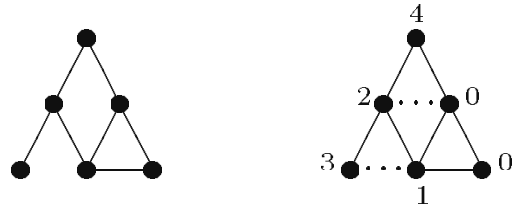


Figure 2.4.5: A series-parallel graph and a 2-tree.

Halin Graphs

DEFINITION

D9: A ***Halin graph*** is a planar graph having the property that its edge set E can be partitioned as $E = \langle T, C \rangle$, where T is a tree with no vertex of degree 2 and C is a cycle including only and all leaves of T.

FACTS

F5: Halin graphs are contained in the class of partial 3-trees.

F6: The set of Halin graphs is not closed under the taking of subgraphs, i.e., some subgraphs of Halin graphs are not Halin graphs.

EXAMPLES

E5: A Halin graph is given in Figure 2.4.6, with the cycle edges drawn on the outer face; their removal yields a tree satisfying the stated degree stipulation.

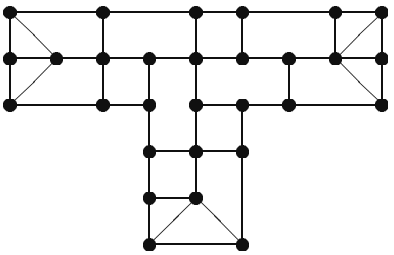


Figure 2.4.6: A Halin graph.

E6: The graph G on the left in Figure 2.4.7 is a 3-tree; vertex labels guide the construction as before. But this graph G is not a Halin graph. However, by removing one edge, we obtain the subgraph G' on the right, which is both a partial 3-tree and a Halin graph. The edges shown in bold form a tree of the Halin graph, and the cycle edges can be easily traced through the leaves of this tree.

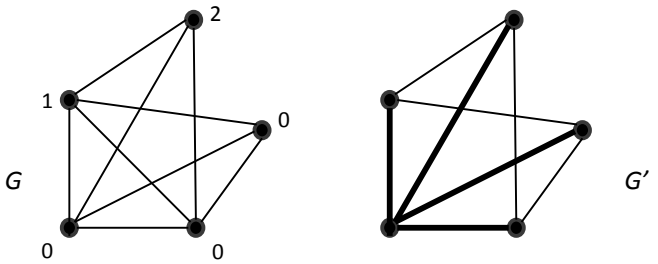


Figure 2.4.7: Non-Halin graph and Halin graph.

Bandwidth-k Graphs

DEFINITION

D10: A graph $G(V, E)$ is a ***bandwidth-k graph*** if there exists a vertex labeling $h : V \to \{1, 2, \ldots |V|\}$ such that $\{u, v\} \in E \Rightarrow |h(u) - h(v)| \leq k$. (Bandwidth is discussed in §9.4.)

EXAMPLE

E7: A bandwidth-3 graph is shown to the left in Figure 2.4.8; displayed to the right is a bandwidth-2 graph.

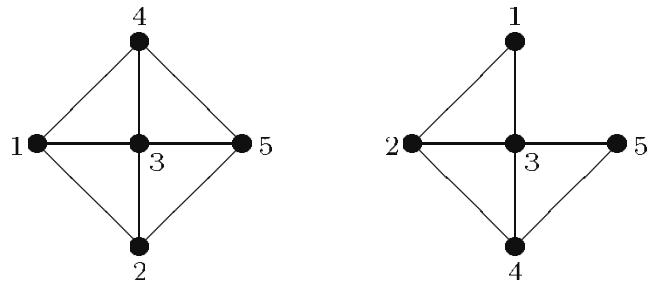


Figure 2.4.8: Bandwidth-3 and bandwidth-2 graphs.

Treewidth-k Graphs

The seminal works by Robertson and Seymour (cf. [RoSe86-a], [RoSe86-b], [RoSe91-a]) are commonly identified as being responsible for motivating the creation of the graph classes in this section. Most notable is the concept of treewidth, which played a key role in the authors' work on graph minors culminating, ultimately, in the proof of Wagner's conjecture, a topic addressed in §2.4.2.

DEFINITIONS

D11: A ***tree-decomposition*** of a graph $G = (V, E)$ is a pair $(\{X_i \mid i \in I\}, T)$, where $\{X_i \mid i \in I\}$ is a family of subsets of V, and T is a tree with vertex set I such that:

- $\bigcup_{i \in I} X_i = V$,
- for all edges $(x, y) \in E$ there is an element $i \in I$ with $x, y \in X_i$, and
- for all triples $i, j, k \in I$, if j is on the path from i to k in T, then $X_i \bigcap X_k \subseteq X_j$.

D12: The ***width*** of a given tree-decomposition is measured as $\max_{i \in I}\{|X_i| - 1\}$.

D13: The ***treewidth*** of a graph G is the minimum width taken over all tree-decompositions of G.

D14: A graph G is a ***treewidth-k graph*** if it has treewidth no greater than k.

REMARK

R2: Trivially, every graph, G, has a tree-decomposition that is defined by a single vertex (representing G itself). On the other hand, we are interested in tree-decompositions and, hence, their graphs, in which the X_i are small (i.e., graphs with small treewidth).

EXAMPLE

E8: A sample tree-decomposition is shown in Figure 2.4.9. For the stated graph, G, one family of suitable vertex sets can be given by: $X_1 = \{v_1, v_2, v_3\}, X_2 = \{v_2, v_7, v_8\}$, $X_3 = \{v_2, v_3, v_7\}, X_4 = \{v_3, v_5, v_7\}, X_5 = \{v_3, v_4, v_5\}$, and $X_6 = \{v_5, v_6, v_7\}$. An appropriate tree T is shown next and then on the right side of Figure 2.4.9, the relevant subgraphs of G induced by the stated pair $(\{X_i\}, T)$ are displayed. Moreover, the graph G has treewidth 2; in fact, the graph is series-parallel.

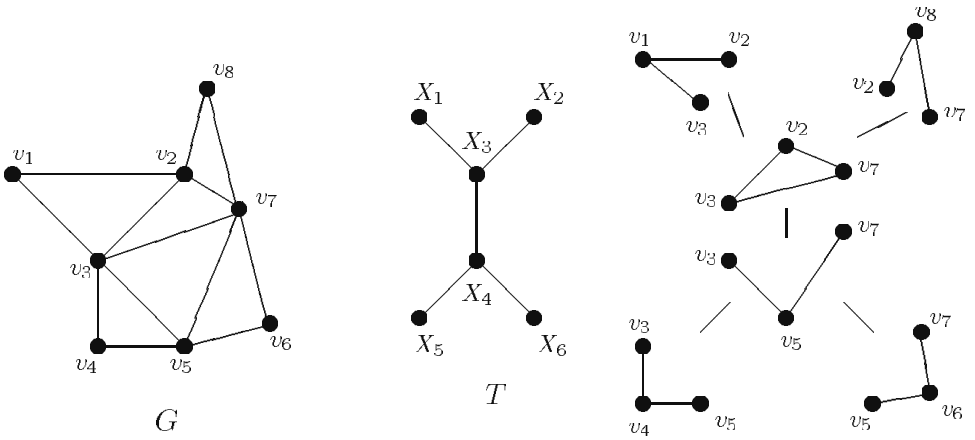


Figure 2.4.9: A sample tree-decomposition.

Pathwidth-k Graphs

DEFINITIONS

D15: A ***path-decomposition*** is a tree-decomposition whose tree is a path.

NOTATION: A path-decomposition is often denoted simply by a sequence of vertex subsets of V, say $\{X_1, X_2, \ldots, X_t\}$, listed in order defined by their position on the path.

D16: The ***width*** of a path-decomposition is $\max_{1 \le i \le t}\{|X_i| - 1\}$.

D17: The ***pathwidth*** of a graph G is the smallest width taken over all path-decompositions of G.

D18: A ***pathwidth-k graph*** is a graph that has pathwidth no greater than k.

EXAMPLE

E9: A sample path-decomposition is shown in Figure 2.4.10. The vertex-sets X_i and the first edge occurrences are displayed below the corresponding vertices of T.

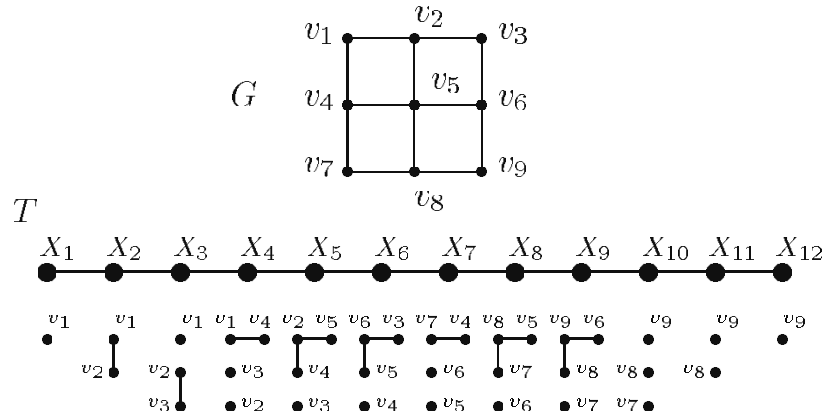


Figure 2.4.10: A path-decomposition.

Branchwidth-k Graphs

DEFINITIONS

D19: A ***branch-decomposition*** of a graph $G = (V, E)$ is a pair (T, f), where T is a tree in which every non-leaf vertex has exactly three neighbors and f is a bijection from the leaves of T to E.

D20: If the degree of every non-leaf vertex in T is *at least* 3, the pair (T, f) is called a ***partial branch-decomposition***.

D21: Let (T, f) be a branch decomposition of a graph $G = (V, E)$. The ***order*** of an edge e of T is the number of vertices v in V such that there exist leaves l_1 and l_2 of T residing in different components of $T - e$, where $f(l_1)$ and $f(l_2)$ are both incident on v.

D22: The ***width*** of a branch decomposition (T, f) is the maximum order of the edges of T.

D23: The ***branchwidth*** of G is the minimum width taken over all branch-decompositions of G.

D24: A graph G is a ***branchwidth-k graph*** if it has branchwidth no greater than k.

FACTS

F7: [RoSe91-a] A graph G is branchwidth-0 if and only if every component of G has at most one edge.

F8: [RoSe91-a] A graph G is branchwidth-1 if and only if every component of G has no more than one vertex with degree greater than or equal to 2.

F9: [RoSe91-a] A graph G is branchwidth-2 if and only if G has treewidth no greater than 2.

EXAMPLE

E10: A branchwidth-2 graph is shown on the left in Figure 2.4.11 (edges are numbered); its branch-decomposition is given to the right.

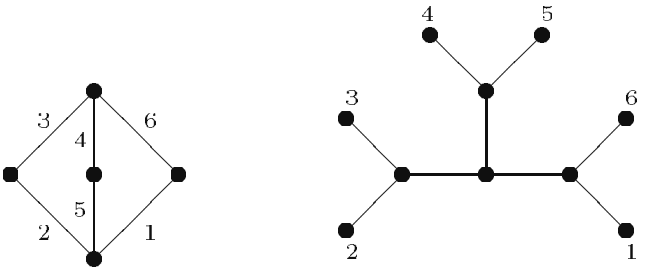


Figure 2.4.11: Branchwidth-2 graph and its branch-decomposition.

k-Terminal Graphs

DEFINITIONS

D25: A ***k-terminal graph*** $G = (V, T, E)$ has a vertex set V, an edge set E, and a set of *distinguished terminals* $T = \{t_1, t_2, \ldots, t_{|T|}\} \subseteq V$, where $|T| \leq k$.

D26: A ***k-terminal recursively structured class*** $C(B, R)$ is specified by a set B of base graphs and a finite rule set $R = \{f_1, f_2, \ldots, f_n\}$, where each f_i is a *recursive composition operation*.

EXAMPLE

E11: A construction for a 2-terminal graph is shown in Figure 2.4.12. Vertices are labeled in order to clarify how constituent subgraphs compose; terminals are denoted by doubly circled vertices.

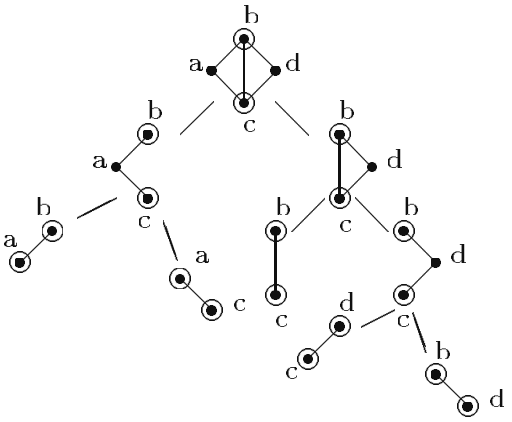


Figure 2.4.12: Recursive construction of a 2-terminal graph.

REMARKS

R3: Typically, for some k, B is the set of connected k-terminal graphs (V, T, E) with $V = T$. But each such base graph is trivially composed of individual edges, so it is reasonable and hence convenient to simply use $C(R)$ to denote $C(B, R)$, where B is a singleton consisting only of edges (i.e., K_2).

R4: The notion of ***composition*** typically permitted in the context of k-terminal graphs can be described in a more formal way. For $1 \leq i \leq m$, let $G_i = (V_i, T_i, E_i)$, such that $V_1, \ldots, V_m$ are mutually disjoint vertex sets. Let $G = (V, T, E)$ as well. Then a ***valid vertex mapping*** is a function $f : \cup_{1 \leq i \leq m} V_i \to V$ such that:

- Vertices from the same G_i remain distinct:

$$v_1 \in V_i, v_2 \in V_i, f(v_1) = f(v_2) \Rightarrow v_1 = v_2$$

- Only (not necessarily all) terminals map to terminals:

$$v \in V_i, f(v) \in T \Rightarrow v \in T_i$$

- Only terminals can merge:

$$v_1 \in V_{i_1}, v_2 \in V_{i_2}, i_1 \neq i_2, f(v_1) = f(v_2) \Rightarrow v_1 \in T_{i_1}, v_2 \in T_{i_2}$$

- Edges are preserved:

$$(\exists i)(\{v_1, v_2\} \in E_i) \Leftrightarrow \{f(v_1), f(v_2)\} \in E$$

NOTATION: If f is a valid vertex mapping, then the corresponding m-ary composition operation (denoted by f) is generally written $f(G_1, \ldots, G_m) = G$.

Cographs

DEFINITION

D27: A ***cograph*** is defined recursively as follows:

- A graph with a single vertex is a cograph.
- If G_1 and G_2 are cographs, then the disjoint union $G_1 \cup G_2$ is a cograph.
- If G_1 and G_2 are cographs, then the cross-product $G_1 \times G_2$ is a cograph, which is formed by taking the union of G_1 and G_2 and adding all edges (v_1, v_2) where v_1 is in G_1 and v_2 is in G_2.

TERMINOLOGY NOTE: Cographs are also referred to as ***complement reducible graphs***.

FACTS

F10: [CoLeBu81] The complement of any cograph is also a cograph.

F11: [CoLeBu81] All cographs are perfect.

EXAMPLE

E12: A cograph construction is demonstrated in Figure 2.4.13. The relevant operations are signified at each node of the decomposition tree (left) for the graph G shown on the right.

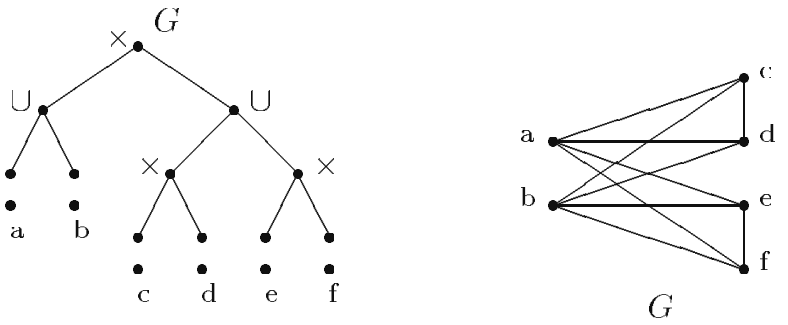


Figure 2.4.13: Cograph construction.

Cliquewidth-k Graphs

The graph parameter *cliquewidth* was introduced in [CoEnRo93] and formed a seminal concept in linking research in graph theory and logic.

DEFINITION

D28: Let [k] denote the set of integers $\{1, 2, \ldots, k\}$. A ***cliquewidth-k graph*** is defined recursively as follows:

- Any graph G with $V(G) = \{v\}$ and $l(v) \in [k]$ is a cliquewidth-k graph.
- If G_1 and G_2 are cliquewidth-k graphs and $i, j \in [k]$ with $i \neq j$, then:
  - The disjoint union $G_1 \cup G_2$ is a cliquewidth-k graph.
  - The graph $(G_1)_{i \times j}$ is a cliquewidth-k graph, where $(G_1)_{i \times j}$ is formed from G_1 by adding all edges (v_1, v_2) such that $l(v_1) = i$ and $l(v_2) = j$.
  - The graph $(G_1)_{i \to j}$ is a cliquewidth-k graph, where $(G_1)_{i \to j}$ is formed from G_1 by switching all vertices with label i to label j.

REMARK

R5: Definition D28 defines the *class* of cliquewidth-k graphs. The cliquewidth of a graph G is the smallest value of k such that G is a cliquewidth-k graph. A cliquewidth decomposition for a graph is a rooted tree such that the root corresponds to G, each leaf corresponds to a labeled, one-vertex graph, and each non-leaf node of the tree is obtained by applying one of the operations $\cup$, $i \times j$, or $i \to j$ to its child or children.

TERMINOLOGY NOTE: In this section, the term *clique* refers to *any* complete subgraph of the graph. In some other sections of this handbook, clique is defined to be a *maximal* subset of pairwise adjacent vertices of the graph.

TERMINOLOGY NOTE: Every tree is a treewidth-1 graph, so treewidth is a measure of how much a graph varies from a tree. Similarly, every clique is a cliquewidth-2 graph, so cliquewidth is a measure of how much a graph varies from a clique. This analogy forms the basis for coining the term *cliquewidth* (cf. [CoOl00]).

EXAMPLE

E13: A cliquewidth-3 construction is given in Figure 2.4.14. As in Example E12, the relevant operations are identified at each node of the decomposition tree (left) for the graph G shown on the right.

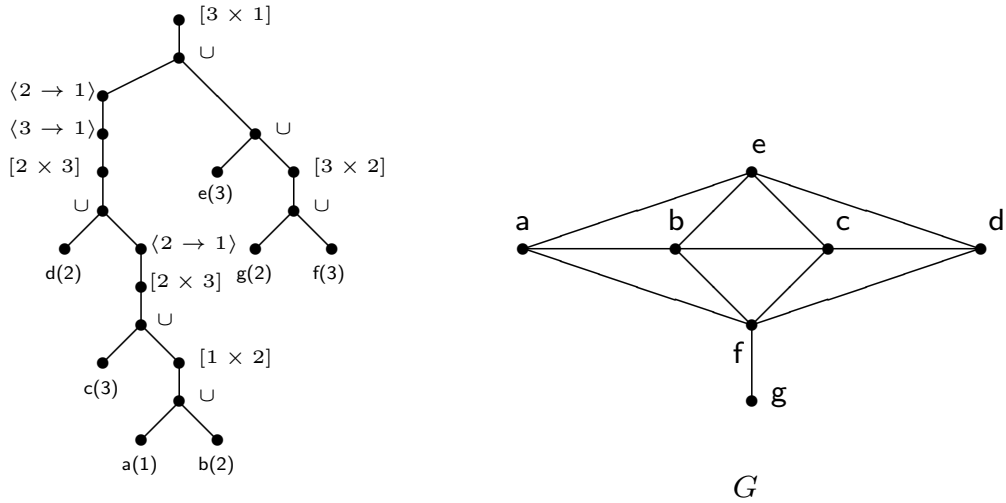


Figure 2.4.14: A cliquewidth-3 graph construction.

k-NLC Graphs

DEFINITION

D29: Let [k] denote the set of integers $\{1, 2, \ldots, k\}$ and let B denote a bipartite graph on [k] × [k]. A ***k-NLC (node-label-controlled) graph*** is defined recursively as follows:

- Any graph G with $V(G) = \{v\}$ and $l(v) \in [k]$ is a k-NLC graph.
- If G_1 and G_2 are k-NLC graphs and $i, j \in [k]$, then:
  - The join $G_1 \times_B G_2$ is a k-NLC graph, where $G_1 \times_B G_2$ is formed from $G_1 \cup G_2$ by adding all edges (v_1, v_2) where $v_1 \in V_1$, $l(v_1) = i$, $v_2 \in V_2$, $l(v_2) = j$, and (i, j) is an edge in E_B.
  - The graph $(G_1)_{i \to j}$ is a k-NLC graph, which is formed from G_1 by switching all vertices with label i to label j.

EXAMPLE

E14: The same graph G previously shown in Figure 2.4.14 is a 2-NLC graph. In Figure 2.4.15, its decomposition tree has leaves corresponding to the vertices a, b, c, d, e, f, and g with starting labels drawn from the set $k = \{1, 2\}$ as shown. Relevant operations are identified with the internal nodes of the tree, i.e., $(i \to j)$ for label switching and (i, j) indicating the specific edge from E_B inducing the stated composition.

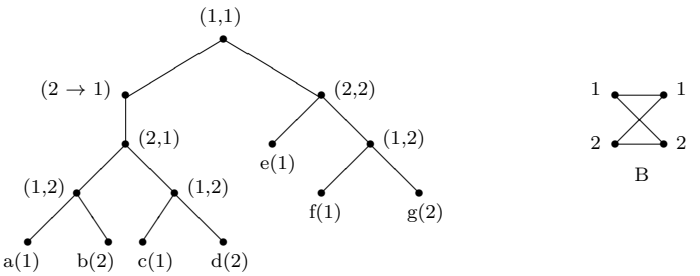


Figure 2.4.15: A 2-NLC graph construction.

k-HB Graphs

The *homogeneous balanced* graphs produce a modular decomposition upon application of a certain decomposition algorithm.

DEFINITION

D30: ***k-HB (homogeneous balanced) graphs*** are graphs for which there is a particular $O(n^{k+2})$-time top-down decomposition algorithm that constructs a pseudo-cliquewidth-$(k + 2^k)$ balanced decomposition.

REMARKS

R6: Top-down decomposition refers to a recognition algorithm that places a candidate graph at the root of a tree and then decomposes this graph into smaller subgraphs that become its children in the tree, and so on, recursively, until reaching the leaves of the tree.

R7: A pseudo-cliquewidth decomposition is similar to a k-NLC decomposition, except that the vertex labels used at one node in the tree are not enforced at other nodes.

R8: A balanced decomposition of an n-vertex graph is a decomposition tree that has height $O(\log\, n)$.

R9: The requirement that the decomposition must be balanced is more restrictive, while simultaneously the pseudo-cliquewidth condition is less restrictive. This trade-off yields the class of k-HB graphs. For more details on these matters, see [Jo03], [BoJoRaSp04].

R10: However, k-HB graphs are an ambiguously defined class due to the nondeterministic nature of this decomposition algorithm. On the other hand, the decomposition is guaranteed to succeed for every cliquewidth-k graph despite this nondeterminism, so every cliquewidth-k graph is a k-HB graph.

2.4.2 Equivalences and Characterizations

Relationships between Recursive Classes

A number of equivalences serve to relate many of the recursive graph classes defined in the previous subsection. Several of these are listed below. Unless a specific source is cited, a good general and fairly comprehensive reference for Facts F12 through F19 (and others) is [BrLeSp99].

FACTS

F12: A graph has treewidth at most k if and only if it is a partial k-tree.

F13: Every bandwidth-k graph is a pathwidth-k and thus a treewidth-k graph.

F14: The class of partial k-trees can be defined as a $(k+1)$-terminal recursive graph class (cf. [WiHe88], [Wi87]).

F15: 1-trees are trees in the usual sense and have treewidth 1.

F16: Trees are series-parallel graphs where only the jackknife operation is used.

F17: Series-parallel graphs in which only the series and parallel operations are used are precisely the 2-terminal series-parallel graphs.

F18: Series-parallel and outerplanar graphs are partial 2-trees and have treewidth 2.

F19: Halin graphs are contained in the class of partial 3-trees; they are also defined as a class of 3-terminal graphs by an appropriate choice of composition operations.

F20: [CoEnRo93] Cographs are precisely the cliquewidth-2 graphs.

F21: [CoRo05] Every treewidth-k graph has cliquewidth at most $3 \cdot 2^{k-1}$.

F22: [RoSe91-a] Every graph of branchwidth at most k has treewidth at most $3k/2$.

F23: [RoSe91-a] Every graph of treewidth at most k has branchwidth at most $k+1$.

F24: [Wa94] Cographs are exactly the 1-NLC graphs.

F25: [Wa94] Every treewidth-k graph has NLC width at most $2^{k+1}-1$.

F26: [Jo98] Every cliquewidth-k graph is a k-NLC graph.

F27: [Jo98] Every k-NLC graph is a cliquewidth-$2k$ graph.

F28: [Jo03], [BoJoRaSp04] Every cliquewidth-k graph is a k-HB graph.

Characterizations

Structural characterizations of recursive graph classes are generally stated in terms of forbidden subgraph minors.

DEFINITIONS

D31: An ***edge-extraction*** operation on a graph $G = (V, E)$ removes an edge e leaving a graph, $G - e$, with $V(G - e) = V$ and $E(G - e) = E - \{e\}$.

D32: The operation of ***edge-contraction*** produces a graph with edge-set $E - \{e\}$ but with a vertex-set obtained by replacing ("merging") the vertices defining e in G, thus creating a new single vertex where the latter inherits all of the adjacencies of the pair of replaced vertices, without introducing loops or multiple edges.

D33: A graph H is a ***minor*** of a graph G if and only if it can be obtained from G by a finite sequence of edge-extraction and edge-contraction operations.

REMARKS

R11: A result apparently first conjectured (but unpublished) by K. Wagner asserts the following: Suppose $\mathcal{F}$ is a graph class with the property that if G is in $\mathcal{F}$ and H is contained as a minor in G, then H is in $\mathcal{F}$, i.e., the class $\mathcal{F}$ is closed under minors. Then there exists a finite set $\{H_1, H_2, \ldots, H_k\}$ of graphs, the ***forbidden minors*** such that G is in $\mathcal{F}$ if and only if it contains no minor isomorphic to any member H_i for $1 \leq i \leq k$.

R12: Robertson and Seymour ([RoSe88-b]) confirmed Wagner's conjecture and with their proof established that any graph class $\mathcal{F}$ closed under minors can be recognized in polynomial time. Unfortunately, this outcome, although deep, is an existential one; we do not know the number of forbidden minors or their sizes in an arbitrary case.

R13: The class of partial k-trees is closed under minors and thus, by the Robertson–Seymour results is completely characterized by a finite set of forbidden minors.

R14: The forbidden minors for partial 3-trees are known (see Fact F33 below), but complete lists of explicit minors for partial k-trees are not known for values of $k \geq 4$.

FACTS

F29: [CoLeBu81] Cographs have no induced paths P_4.

F30: Trees are graphs having no K_3 minor.

F31: The class of partial 2-trees is characterized by a single forbidden minor: the complete graph K_4.

F32: The forbidden minors of outerplanar graphs are K_4 and $K_{2,3}$.

F33: The class of partial 3-trees has four forbidden minors: K_5 and the three graphs shown in Figure 2.4.16.

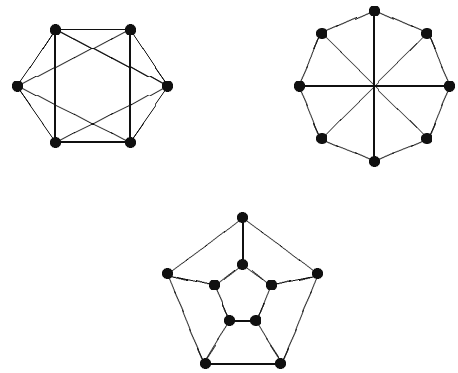


Figure 2.4.16: Forbidden minors of partial 3-trees.

2.4.3 Recognition

In order to solve graph problems on recursive classes and particularly, to do so efficiently, it is necessary that membership in the classes be ***quickly recognized***.

REMARKS

R15: Some recognition algorithms are direct and essentially ad hoc. For example, Halin graphs can be recognized by first testing for 3-connectivity and for planarity. Then simply embed the graph in the plane, select a largest cycle of edges that corresponds to a face on the plane embedding, remove these edges and test if the remaining graph is a tree of the stated form (see [CoNaPu83]).

R16: Partial 2-trees or series-parallel graphs are recognizable, unambiguously, by successive application of the following ***reduction operations*** (cf. [Du65]): replace any vertex of degree 2, say v_j, and its incident edges (v_i, v_j) and (v_j, v_k) by a new edge (v_i, v_k); replace any pair of multiple edges by a single edge; and eliminate any edges incident to a vertex of degree 1 unless only one edge remains. Then a single edge remains, upon an admissible application of these reduction operations, if and only if the original graph is a partial 2-tree; otherwise, the process will stop with either K_4 or a graph with a K_4 minor.

R17: Similar reduction operations have also been described in the case of partial 3-trees (cf. [ArPr86]) as well as for partial 4-trees ([Sa96]).

EXAMPLE

E15: An illustration of a successful reduction sequence is shown in Figure 2.4.17.

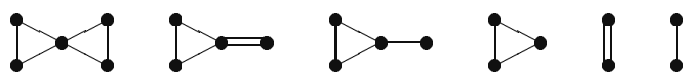


Figure 2.4.17: Reduction operations for a partial 2-tree.

Recognition of Recursive Classes

FACTS

F34: Trees can be recognized and their decomposition constructed in linear time.

F35: Series-parallel graphs can be recognized and their decomposition constructed in linear time.

F36: Treewidth-k, pathwidth-k, branchwidth-k, and bandwidth-k graphs can be recognized and their decompositions constructed in $O(n^{k+2})$ time.

COMPUTATIONAL NOTE: For fixed k the polynomial-time algorithms of Fact F36 are practical.

F37: The graph classes of Fact F36 can be recognized in *linear time* for fixed k.

COMPUTATIONAL NOTE: The corresponding algorithms referred to by Fact F37 are not practical because their running times possess enormous hidden constants.

F38: When $k \leq 4$, more practical linear-time recognition algorithms have been found for the graph classes in Fact F36 (cf. [MaTh91] for $k = 3$; [Sa96] when $k = 4$).

F39: When k is part of the problem instance, the recognition problems associated with the graphs of Fact F36 are $\mathcal{NP}$-complete.

F40: Branchwidth can be determined in polynomial time for planar graphs [SeTh94].

F41: Since partial k-trees are characterizable by a finite set of forbidden minors, they are polynomially recognizable (cf. [RoSe88-b]).

COMPUTATIONAL NOTE: Fact F41 was established in the graph minors results of Robertson and Seymour. However, the result is existential rather than constructive and so the actual exhibition of the implied algorithms remains elusive.

F42: [Wi87] Every k-terminal graph is a treewidth-k' graph for some k' that depends upon k and the particular set of recursive composition operations. For example, if m denotes the maximum *arity* of any operation, then $k' \leq km$.

TERMINOLOGY NOTE: The term "arity" refers to the number of operands. For example, a binary operation has arity 2.

F43: [CoPeSt85] Cographs can be recognized and their decomposition constructed in linear time.

F44: [CoHaLaReRo00] Recognition of cliquewidth-k graphs is solvable in polynomial time for $k \leq 3$, but this problem remains open for fixed $k \geq 4$.

F45: [FeRoRoSz09] Recognition of cliquewidth-k graphs is an $\mathcal{NP}$-complete problem for arbitrary k.

F46: [Jo00] Recognition of k-NLC graphs is solvable in polynomial time for $k \leq 2$, but this problem remains open for fixed $k \geq 3$.

F47: [GuWa05] Recognition of k-NLC graphs is $\mathcal{NP}$-complete for arbitrary k.

F48: [Jo03], [BoJoRaSp04] Algorithms for problems defined on k-HB graphs are robust with respect to cliquewidth-k graphs. That is, such an algorithm either determines the correct answer or reports that the decomposition was unsuccessful and hence the input graph is not a cliquewidth-k graph. The $O(n^{k+2})$-time decomposition algorithm for k-HB graphs is guaranteed to succeed for all cliquewidth-k graphs as well as some others.

References

[ArCoPr87] S. Arnborg, D. G. Corneil, and A. Proskurowski, Complexity of finding embeddings in a k-tree, *SIAM J. Algebraic and Discrete Methods* 8 (1987), 277–284.

[ArCoPrSe93] S. Arnborg, B. Courcelle, A. Proskurowski, and D. Sesse, An algebraic theory of graph reductions, *J. ACM* 40 (1993), 1134–1164.

[ArHePr94] S. Arnborg, S. Hedetniemi, and A. Proskurowski (eds.), Efficient algorithms and partial k-trees, special issue of *Discrete Applied Math.* 54 (1994).

[ArPr85] S. Arnborg and A. Proskurowski, Characterization and recognition of partial k-trees, *Congressus Numerantium* 47 (1985), 69–75.

[ArPr86] S. Arnborg and A. Proskurowski, Characterization and recognition of partial 3-Trees, *SIAM J. Algebraic and Discrete Methods* 7 (1986) 305–314.

[ArPrCo90] S. Arnborg, A. Proskurowski, and D. G. Corneil, Forbidden minors characterization of partial 3-trees, *Discrete Math.* 80 (1990), 1–19.

[BaOl98] L. Babel and S. Olariu, On the structure of graphs with few P_4s, *Discrete Applied Math.* 84 (1998), 1–13.

[BePi71] L. W. Beineke and R. E. Pippert, Properties and characterizations of k-trees, *Mathematika* 18 (1971), 141–151.

[BiRoSeTh91] D. Bienstock, N. Robertson, P. D. Seymour, and R. Thomas, Quickly excluding a forest, *J. Combin. Theory Series B* 52 (1991), 274–283.

[Bo88] R. B. Borie, Recursively constructed graph families: membership and linear algorithms, Ph.D. Dissertation, School of Information and Computer Science, Georgia Institute of Technology, 1988.

[Bo90] H. L. Bodlaender, Classes of graphs with bounded treewidth, *Tech. Rep. RUU-CS-86-22*, Department of Computer Science, Utrecht University, The Netherlands, 1990.

[Bo93] H. L. Bodlaender, A tourist guide through treewidth, *Acta Cybernetica* 11 (1993), 1–23.

[Bo96] H. L. Bodlaender, A linear-time algorithm for finding tree decompositions of small treewidth, *SIAM J. Computing* 25 (1996), 1305–1317.

[BoGiHaKl95] H. L. Bodlaender, J. R. Gilbert, H. Hafsteinsson, and T. Kloks, Approximating treewidth, pathwidth, and minimum elimination tree height, *J. Algorithms* 18 (1995), 238–255.

[BoJoRaSp04] R. B. Borie, J. L. Johnson, V. Raghavan, and J. Spinrad, Robust algorithms for some problems on graphs of bounded clique-width, manuscript, presented at *SIAM Conference on Discrete Mathematics* (2004).

[BoKl96] H. L. Bodlaender and T. Kloks, Efficient and constructive algorithms for the pathwidth and treewidth of graphs, *J. Algorithms* 21 (1996), 358–402.

[BoMo93] H. L. Bodlaender and R. H. Möhring, The pathwidth and treewidth of cographs, *SIAM J. Discrete Math.* 6 (1993), 181–186.

[BoPaTo91] R. B. Borie, R. G. Parker, and C. A. Tovey, Deterministic decomposition of recursive graph classes, *SIAM J. Discrete Math.* 4 (1991), 481–501.

[BrLeSp99] A. Brandstädt, V. B. Le, and J. Spinrad, *Graph Classes: A Survey*, SIAM Monographs on Discrete Math. and Applications, SIAM, Philadelphia (1999).

[Co90] B. Courcelle, The monadic second-order logic of graphs I: Recognizable sets of finite graphs, *Inform. and Comput.* 85 (1990), 12–75.

[Co92] B. Courcelle, The monadic second-order logic of graphs III: Tree-decompositions, minors, and complexity issues, *Inform. Théorique Appl.* 26 (1992), 257–286.

[Co95] B. Courcelle, The monadic second-order logic of graphs VIII: Orientations, *Annals of Pure and Applied Logic* 72 (1995), 103–143.

[Co96] B. Courcelle, The monadic second-order logic of graphs X: Linear orderings, *Theoretical Computer Science* 160 (1996), 87–143.

[CoEnRo93] B. Courcelle, J. Engelfriet, and G. Rozenberg, Handle-rewriting hypergraph grammars, *J. of Computer and Systems Sciences* 46 (1993), 218–270.

[CoHaLaReRo00] D. G. Corneil, M. Habib, J. M. Lanlignel, B. Reed, and U. Rotics, Polynomial-time recognition of clique-width ≤ 3 graphs, *Lecture Notes in Computer Science* 1776 (2000), 126–134.

[CoKi83] D. G. Corneil and D. G. Kirkpatrick, Families of recursively defined perfect graphs, *Congressus Numerantium* 39 (1983), 237–246.

[CoLeBu81] D. G. Corneil, H. Lerchs, and L. S. Burlington, Complement reducible graphs, *Discrete Applied Math.* 3 (1981), 163–174.

[CoMo93] B. Courcelle and M. Mosbah, Monadic second-order evaluations on tree-decomposable graphs, *Theoretical Computer Science* 109 (1993), 49–82.

[CoNaPu83] G. Cornuejols, D. Naddef, and W. R. Pulleyblank, Halin graphs and the travelling salesman problem, *Math Programming* 26 (1983), 287–294.

[CoOl00] B. Courcelle and S. Olariu, Upper bounds to the clique-width of graphs, *Discrete Applied Math.* 101 (2000), 77–114.

[CoPeSt84] D. G. Corneil, Y. Perl, and L. K. Stewart, Cographs: recognition, applications and algorithms, *Congressus Numerantium* 43 (1984), 249–258.

[CoPeSt85] D. G. Corneil, Y. Perl, and L. Stewart, A linear recognition algorithm for cographs, *SIAM J. Computing* 14 (1985), 926–934.

[CoRo05] D. G. Corneil and U. Rotics, On the relationship between clique-width and treewidth, *SIAM Journal on Computing* 34 (2005), 825–847.

[Du65] R. J. Duffin, Topology of series-parallel graphs, *J. Math. Anal. Appl.* 10 (1965), 303–318.

[FeRoRoSz09] M. R. Fellows, F. A. Rosamond, U. Rotics, and S. Szeider, Clique-width is $\mathcal{NP}$-complete, *SIAM J. on Discrete Mathematics* 23 (2009), 909–939.

[GoRo99] M. C. Golumbic and U. Rotics, On the clique-width of perfect graph classes, *Lecture Notes in Computer Science* 1665 (1999), 135–147.

[GrSk91] D. Granot and D. Skorin-Kapov, NC Algorithms for recognizing partial 2-trees and 3-trees, *SIAM J. Algebraic and Discrete Methods* 4 (1991), 342–354.

[GuWa05] F. Gurski and E. Wanke, Minimizing NLC-width is $\mathcal{NP}$-complete, *Lecture Notes in Computer Science* 3787 (2005), 69–80.

[HeYe87] X. He and Y. Yesha, Parallel recognition and decomposition of two-terminal series-parallel graphs, *Information and Computing* 75 (1987), 15–38.

[Jo98] O. Johansson, Clique-decomposition, NLC-decomposition, and modular decomposition relationships and results for random graphs, *Congressus Numerantium* 132 (1998), 39–60.

[Jo00] O. Johansson, NLC 2-decomposition in polynomial time, *International Journal of Foundations of Computer Science* 11 (2000), 373–395.

[Jo03] J. Johnson, Polynomial time recognition and optimization algorithms on special classes of graphs, Ph.D. Dissertation, Computer Science, Vanderbilt University, 2003.

[KaIsUe85] Y. Kajitani, A. Ishizuka, and S. A. Ueno, Characterization of the partial k-tree in terms of certain structures, *Proc. ISCAS '85* (1985), 1179–1182.

[Kl94] T. Kloks, Treewidth: computations and approximations, *Lecture Notes in Computer Science* 842 (1994).

[KlKr95] T. Kloks and D. Kratsch, Treewidth of chordal bipartite graphs, *J. Algorithms* 19 (1995), 266–281.

[MaTh91] J. Matoušek and R. Thomas, Algorithms for finding tree-decompositions of graphs, *J. Algorithms* 12 (1991), 1–22.

[Pr93] A. Proskurowski, Graph reduction and techniques for finding minimal forbidden minors, in *Proc. AMS Workshop on Graph Minors*, Seattle 1991, Contemporary Math. AMS 147, Graph Structure Theory (1993), 591–600.

[Re92] B. Reed, Finding approximate separators and computing treewidth quickly, in *Proc. 9th Symposium on Theoretical Aspects of Computer Science* (1992), 221–228.

[Re93] B. Reed, Treewidth and Tangles: A New Connectivity Measure and Some Applications, in *Surveys in Combinatorics* (1997), Cambridge University Press, 87–162.

[Ro74] D. J. Rose, On simple characterization of k-trees, *Discrete Math.* 7 (1974), 317–322.

[RoSe83] N. Robertson and P. D. Seymour, Graph minors. I. Excluding a forest, *J. Combin. Theory Series B* 35 (1983), 39–61.

[RoSe84] N. Robertson and P. D. Seymour, Graph minors. III. Planar treewidth, *J. Combin. Theory Series B* 36 (1984), 49–64.

[RoSe86-a] N. Robertson and P. D. Seymour, Graph minors. II. Algorithmic aspects of treewidth, *J. Algorithms* 7 (1986), 309–322.

[RoSe86-b] N. Robertson and P. D. Seymour, Graph minors. V. Excluding a planar graph, *J. Combin. Theory Series B* 41 (1986), 92–114.

[RoSe86-c] N. Robertson and P. D. Seymour, Graph minors. VI. Disjoint paths across a disc, *J. Combin. Theory Series B* 41 (1986), 115–138.

[RoSe88-a] N. Robertson and P. D. Seymour, Graph minors. VII. Disjoint paths on a surface, *J. Combin. Theory Series B* 45 (1988), 212–254.

[RoSe88-b] N. Robertson and P. D. Seymour, Graph minors. XX. Wagner's conjecture, manuscript (1988), *J. Combin. Theory Series B* 92 (2004), 325–357.

[RoSe90–a] N. Robertson and P. D. Seymour, Graph minors. IV. Treewidth and well-quasi-ordering, *J. Combin. Theory Series B* 48 (1990), 227–254.

[RoSe90-b] N. Robertson and P. D. Seymour, Graph minors. IX. Disjoint crossed paths, *J. Combin. Theory Series B* 49 (1990), 40–77.

[RoSe90-c] N. Robertson and P. D. Seymour, Graph minors. VIII. A Kuratowski theorem for general surfaces, *J. Combin. Theory Series B* 48 (1990), 255–288.

[RoSe91-a] N. Robertson and P. D. Seymour, Graph minors. X. Obstructions to tree-decompositions, *J. Combin. Theory Series B* 52 (1991), 153–190.

[RoSe91-b] N. Robertson and P. D. Seymour, Graph minors. XVI. Excluding a non-planar graph, manuscript (1991), *J. Combin. Theory Series B* 89 (2003), 43–76.

[RoSe92] N. Robertson and P. D. Seymour, Graph minors. XXII. Irrelevant vertices in linkage problems, manuscript (1992), *J. Combin. Theory Series B*, to appear.

[RoSe94] N. Robertson and P. D. Seymour, Graph minors. XI. Distance on a surface, *J. Combin. Theory Series B* 60 (1994), 72–106.

[RoSe95] N. Robertson and P. D. Seymour, Graph minors. XIII. The disjoint paths problem, *J. Combin. Theory Series B* 63 (1995), 65–110.

[RoSeTh94] N. Robertson, P. D. Seymour, and R. Thomas, Quickly excluding a planar graph, *J. Combin. Theory Series B* 62 (1994), 323–348.

[Sa96] D. P. Sanders, On linear recognition of treewidth at most four, *SIAM J. Discrete Math.* 9 (1996), 101–117.

[SaTu90] A. Satyanarayana and L. Tung, A characterization of partial 3-trees, *Networks* 20 (1990), 299–322.

[Sc88] P. Scheffler, What graphs have bounded treewidth?, in *Proc. Fischland Colloquium on Discrete Math. and Applications*, Rostock Math. Kolloq. (1988).

[SeTh94] P. D. Seymour and R. Thomas, Call routing and the ratcatcher, *Combinatorica* 14 (1994), 217–241.

[Wa94] E. Wanke, k-NLC graphs and polynomial algorithms, *Discrete Applied Math.* 54 (1994), 251–266.

[Wi87] T. V. Wimer, Linear algorithms on k-terminal graphs, Ph.D. Dissertation, Department of Computer Science, Clemson University, 1987.

[WiHe88] T. V. Wimer and S. T. Hedetniemi, K-terminal recursive families of graphs, *Congressus Numerantium* 63 (1988), 161–176.

Section 2.5
Structural Graph Theory

Maria Chudnovsky, Columbia University

INTRODUCTION

The goal of this section is to survey some recent results in structural graph theory. One of the greatest achievements of structural graph theory to this day has been the Robertson–Seymour Graph Minor Project, which, in addition to answering a long standing open conjecture of Wagner, revolutionized the field and laid foundation to a large body of research that is being conducted today. For this very reason, numerous excellent surveys on the topic of minors have been written over the years (see, for example, [Di05, Re97, Th99]), and so in this section we concentrate on other aspects of structural graph theory. The bulk of the section is devoted to the area of induced subgraphs, which received a fair amount of attention in the past 10 years, due to the proof of the Strong Perfect Graph Conjecture, a famous open question, posed by Claude Berge in 1961 [Be61], which was finally solved in the early 2000s [ChRoSeTh06]. We conclude the section with a survey of new results on tournament immersion.

DEFINITIONS

All our graphs are finite and simple (except when we explicitly say otherwise). Let us start with some definitions. We omit some very basic and standard definitions here; the reader is referred to [Di05] or [We01] for those. Let G be a graph. We denote its vertex set by $V(G)$, and its edge set by $E(G)$.

D1: The ***complement*** of G, denoted by G^c, is a graph with vertex set$V(G)$, such that two vertices are adjacent in G^c if and only if they are non-adjacent in G.

D2: A ***clique*** in G is a set of vertices of G, all pairwise adjacent.

D3: A ***stable set*** in G is a set of vertices, all pairwise non-adjacent. Thus, $S \subseteq V(G)$ is a stable set in G if and only if it is a clique in G^c.

D4: The largest size of a clique in G, the ***clique number*** of G, is denoted by $\omega(G)$, and the largest size of a stable set, the ***stability number*** of G, by $\alpha(G)$.

D5: The ***chromatic number*** of G, denoted by $\chi(G)$, is the smallest number k for which the vertices of G can be colored with k colors, so that no two adjacent vertices receive the same color.

D6: Given a graph H, we say that H is an ***induced subgraph*** of G if $V(H) \subseteq V(G)$, and $uv \in E(H)$ if and only if $uv \in E(G)$ for every $u, v \in V(H)$.

D7: Given a graph H, we say that the graph G is an ***H-free graph*** if no induced subgraph of G is isomorphic to H. For $X \subseteq V(G)$, the subgraph of G induced by X is denoted $G|X$.

D8: A ***component*** of G is a maximal connected subgraph of G.

D9: We say that the graph G is ***anticonnected*** if its complement G^c is connected.

D10: An ***anticomponent*** of G is a maximal anticonnected induced subgraph of G.

2.5.1 Perfect Graphs

Let G be a graph. It follows immediately from the definitions of the clique number and the chromatic number that $\chi(G) \geq \omega(G)$. It is then natural to ask: for what graphs G does equality hold, that is, when is $\chi(G) = \omega(G)$? It turns out that the two parameters are equal in many natural classes of graphs: bipartite graphs (these are graphs whose vertex set is the union of two stable sets), complements of bipartite graphs [Ko16], comparability graphs [Di50](these are graphs whose vertices are elements of a given partially ordered set P, and two elements are adjacent if and only if they are comparable in P), and many others. However, here is an unfortunate example: take a graph G', and let G be the union of G' with a complete graph on $|V(G')|$ vertices. Clearly $\chi(G) = \omega(G) = |V(G')|$, and yet we have not learned anything about the structure of the subgraph G' of G.

Thus, to get a nice answer, we need to modify the question a little. In 1961 Claude Berge came up with what seems to be right modification, the notion of a "perfect graph" [Be61].

DEFINITIONS

D11: A graph G is a ***perfect graph*** if $\chi(H) = \omega(H)$ for every induced subgraph H of G. A graph that is not perfect is called ***imperfect***.

D12: A ***cycle of length*** n (where $n \geq 3$ is an integer) is the graph with vertex set $\{v_1, \ldots, v_n\}$, such that v_i is adjacent to v_j if and only if $|i - j| = 1 \ mod \ n$. We denote this graph by C_n.

EXAMPLES

E1: Bipartite graphs, their complements, comparability graphs, and many other natural classes of graphs are all perfect, but the pathological example we constructed at the end of the last paragraph may not be.

These two examples of imperfect graphs will be important in what follows.

E2: It is easy to see that if $n \geq 5$ is odd, then $\omega(C_n) = 2$ and $\chi(C_n) = 3$, and therefore C_n is imperfect.

E3: Let us next consider C_n^c, again with $n \geq 5$ odd. In this graph, the largest clique has size $\lfloor \frac{n}{2} \rfloor$, and the chromatic number is $\lceil \frac{n}{2} \rceil$ (since the size of the largest stable set is two, and so at most two vertices can be colored with a given color), so C_n^c is also imperfect.

Fact F1 arose as a conjecture of Berge. It is now known as the ***Weak Perfect Graph Theorem***.

FACT, FIRST CONJECTURED BY BERGE

F1: (Lovász [Lo72]). A graph G is perfect if and only if G^c is perfect.

DEFINITIONS

D13: Let G be a graph, and H be an induced subgraph of G. We say that H is a ***hole*** if H is isomorphic to C_n for some integer $n \geq 4$; moreover, H is an ***odd hole*** if n is odd, and an ***even hole*** if n is even.

D14: Similarly, H is an ***antihole*** if H is isomorphic to C_n^c for some integer $n \geq 4$; also H is an ***odd antihole*** if n is odd, and an ***even antihole*** if n is even.

D15: Let us say that a graph is a ***Berge graph*** if G has no odd holes and no odd antiholes (this terminology is due to Chvátal).

Fact F2 also arose as a conjecture of Berge. Since every induced subgraph of a perfect graph is perfect, the "only if" direction of Fact F2 follows immediately. The "if" direction remained open for more than 40 years, until it was proved in the early 2000s. This fact is commonly known as the ***Strong Perfect Graph Theorem***.

ANOTHER FACT, ALSO FIRST CONJECTURED BY BERGE

F2: (Chudnovsky, Robertson, Seymour, and Thomas [ChRoSeTh06]). A graph G is perfect if and only if it has no odd holes and no odd antiholes.

Outline of the Proof of the Strong Perfect Graph Theorem

Since its introduction by Berge, the theory of perfect graphs became a very active area of graph theory (largely motivated by the attempts to prove Fact F2), and so we will spend some time here discussing the ideas of that proof. The key idea is to "describe all Berge graphs".

The main part of the proof of Fact F2 is devoted to proving a theorem that says that every Berge graph is either "basic" (meaning that it belongs to a well-understood class of graphs, all of whose members are perfect), or admits a "useful decomposition" (this is a decomposition that cannot occur in an imperfect Berge graph with a minimum number of vertices). Let us call this a "decomposition theorem".

Any theorem of this form would imply Fact F2; let us quickly run through the implication. Assume that Fact F2 is false; then there exists an imperfect Berge graph G with $|V(G)|$ minimum. Apply the theorem to G. Then G is either basic (and therefore perfect, which is a contradiction), or admits a useful decomposition, which is again a contradiction by the definition of usefulness. The fact that a decomposition theorem of this form should exist had been a growing belief in the field for a number of years prior to the proof of Fact F2, and was finally formulated, but not published, as a conjecture by Cornuéjols, Conforti, and Vušković.

Let us now make this more precise.

DEFINITIONS

D16: Given a graph H, its ***line graph*** $L(H)$ is the graph with vertex set $E(H)$, and $ef \in E(L(H))$ if and only if the edges e and f share an endpoint in H.

D17: We say that G is a ***double-split graph*** if $V(G)$ can be partitioned into four sets $\{a_1, \ldots, a_m\}$, $\{b_1, \ldots, b_m\}$, $\{c_1, \ldots, c_n\}$, $\{d_1, \ldots, d_n\}$ for some $m, n \geq 2$, such that:

- a_i is adjacent to b_i for $1 \leq i \leq m$, and c_j is nonadjacent to d_j for $1 \leq j \leq n$.
- there are no edges between $\{a_i, b_i\}$ and $\{a_{i'}, b_{i'}\}$ for $1 \leq i < i' \leq m$, and all four edges between $\{c_j, d_j\}$ and $\{c_{j'}, d_{j'}\}$ for $1 \leq j < j' \leq n$.
- there are exactly two edges between $\{a_i, b_i\}$ and $\{c_j, d_j\}$ for $1 \leq i \leq m$ and $1 \leq j \leq n$, and these two edges have no common end.

D18: Let us say that a graph G is a ***Berge-basic graph*** if either G or G^c is bipartite, the line graph of a bipartite graph, or a double-split graph ("Berge-basic" is an ad hoc definition; we use it to avoid confusion with graphs that we would like to consider basic in other settings). It is not difficult to see (using theorems of König [Ko16]) that all Berge-basic graphs are perfect.

Next let us define the useful decompositions used in [ChRoSeTh06].

D19: A ***skew-partition*** in a graph G is a partition (A, B) of $V(G)$ such that A is not connected and B is not anticonnected. Skew-partitions were first introduced by Chvátal [Ch85].

D20: A ***proper 2-join*** (a special case of a decomposition defined by Cornuéjols and Cunningham [CoCu85]) in G is a partition (X_1, X_2) of $V(G)$ such that there exist disjoint nonempty $A_i, B_i \subseteq X_i$ $(i = 1, 2)$ satisfying:

- every vertex of A_1 is adjacent to every vertex of A_2, and every vertex of B_1 is adjacent to every vertex of B_2,

- there are no other edges between X_1 and X_2,
- for $i = 1, 2$, every component of $G|X_i$ meets both A_i and B_i, and
- for $i = 1, 2$, if $|A_i| = |B_i| = 1$ and $G|X_i$ is a path joining the members of A_i and B_i, then it has odd length ≥ 3.

D21: If $X \subseteq V(G)$ and $v \in V(G) \setminus X$, we say that v is ***X-complete*** if v is adjacent to every vertex in X, and that v is ***X-anticomplete*** if v has no neighbors in X.

D22: If $X, Y \subseteq V(G)$ are disjoint, we say that X is ***complete*** to Y (or the pair (X, Y) is ***complete***) if every vertex in X is Y-complete; and being ***anticomplete*** to Y is defined similarly.

D23: Finally, a ***proper homogeneous pair*** in G (a slight variation of a decomposition by Chvátal and Sbihi [ChSb87]) is a pair of disjoint nonempty subsets (A, B) of $V(G)$, such that, if A_1, A_2, respectively, denote the sets of all A-complete vertices and all A-anticomplete vertices in $V(G)$, and if B_1, B_2 are defined similarly, then:

- $A_1 \cup A_2 = B_1 \cup B_2 = V(G) \setminus (A \cup B)$ (and in particular, every vertex in A has a neighbor in B and a non-neighbor in B, and vice versa), and
- the four sets $A_1 \cap B_1, A_1 \cap B_2, A_2 \cap B_1, A_2 \cap B_2$ are all nonempty.

REMARK

R1: Note that if G admits a skew-partition then so does G^c, and the same holds for homogeneous pairs. However, a 2-join in G is substantially different from a 2-join in G^c. In fact, Fact F2 uses a slight variant of the skew-partition decomposition (also invariant under taking complements), called a *balanced skew-partition*, but the definition of that is somewhat technical, and we omit it here.

FACT

We can now state the decomposition theorem of [ChRoSeTh06].

F3: [ChRoSeTh06] Let G be a Berge graph. Then either

- G is Berge-basic, or
- G admits a balanced skew-partition, or
- G admits a proper homogeneous pair, or
- one of G, G^c admits a proper 2-join.

REMARKS

R2: In [Ch03, Ch06], the author showed that the proper homogeneous pair decomposition is in fact unnecessary, and Fact F3 remains true if we simply omit it. We will come back to this result later, in §2.5.4, since it contains a tool that has since been very useful in the study of induced subgraphs. The proof of Fact F3 occupies most of [ChRoSeTh06], which is over 150 pages long. Recently, Seymour and the author were able to shorten the proof of Fact F2 by about a third, proving a theorem that is similar to Fact F3 but uses more kinds of decompositions [ChSe09].

R3: Even though Fact F3 provides enough structural information about Berge graphs to prove Fact F2, it is not what we would call a "structure theorem" for Berge (or perfect) graphs. The problem is that while Fact F3 gives a way to break (or decompose) a big Berge graph into smaller pieces, this decomposition cannot be "reversed". We cannot use Fact F3 to prove a theorem that says "every perfect graph can be built from basic pieces by gluing them together via certain operations; and every graph built in this way is perfect". Thus, while we have a *decomposition theorem* for perfect graphs, we do not have a *structure theorem*. The question of explicitly describing the structure of perfect graphs is currently wide open, and any progress on it would be a breakthrough in the area.

2.5.2 Other Decomposition Theorems

This subsection is devoted to more decomposition results for graphs with certain induced subgraphs forbidden. The theorems in this subsection have not, so far, been turned into explicit structure theorems. Of course, a carefully crafted decomposition theorem is often enough to answer a specific question about a certain class of graphs (just as Fact F3 was strong enough to prove Fact F2); decomposition theorems also often have strong algorithmic consequences.

DEFINITION

D24: Let $\mathcal{F}$ be a family of graphs. We say that a graph G is ***$\mathcal{F}$-free*** if G is F-free for every $F \in \mathcal{F}$.

Defining $\mathcal{C}$ to be the family of all cycles of odd length at least five, and their complements, Fact F3, with the strengthening of [Ch03, Ch06], can be restated as follows:

FACT

F4: Let G be a $\mathcal{C}$-free graph. Then either

- G is Berge-basic, or
- G admits a balanced skew-partition, or
- one of G, G^c admits a proper 2-join.

DEFINITION

D25: A ***k-star cutset*** in a graph G is a partition (A, B) of $V(G)$ such that A is not connected, and there is a clique $K \subseteq B$, such that $|K| = k$, and every vertex of $B \setminus K$ has a neighbor in K. A 1-star cutset is usually called a ***star cutset***.

We observe that every star cutset is a skew-partition, and every skew-partition is a 2-star cutset (but the converse implications are false). In this subsection we also refer to a 2-*join decomposition*; it is a slight variant of the proper 2-joins that we have used so far, but we will not define it exactly.

Next, we discuss a number of decomposition theorems for $\mathcal{F}$-free graphs for various families $\mathcal{F}$. Here we use the word "basic" loosely, to mean that a graph belongs to some well-understood explicitly constructed class of graphs, all of which are $\mathcal{F}$-free for the family $\mathcal{F}$ in question.

DEFINITION

D26: Let $\mathcal{B}$ be the family of all cycles C_n where n is not divisible by 4; the $\mathcal{B}$-free graphs are called ***balanced graphs***.

FACT

F5: Conforti and Rao [CoRa92] proved a decomposition theorem for $\mathcal{B}$-free graphs, where they showed that every such graph is either basic ("strongly balanced") or admits a 2-join, or an "extended star cutset" (this is a variation on the star cutset theme).

This was a ground-breaking result, the first theorem of the kind. In particular it lead to a polynomial-time recognition algorithm for the class of balanced graphs.

Here are more results of a similar flavor.

DEFINITIONS

D27: Let $\mathcal{F}_{odd}$ be the family of cycles C_n with odd $n > 3$, and $\mathcal{F}_{even}$ the family of cycles C_n with even n. An $\mathcal{F}_{odd}$-free graph is called ***odd-hole-free***, and an $\mathcal{F}_{even}$-free graph is called ***even-hole-free***.

FACTS

F6: [CoCoVu04]. Every odd-hole-free graph is either basic, or admits a 2-join or a k-star cutset, for some $k \leq 2$.

In [CoCoKaVu02] a similar result was proved for $\mathcal{F}_{even}$-free graphs. More precisely:

F7: [CoCoKaVu02] Every even-hole-free graph is either basic, or admits a 2-join or admits a k-star cutset, for some $k \leq 3$.

REMARK

R4: Fact F7 was later strengthened in [daSiVu13], where only 2-joins and star cutsets are used (though the list of basic classes had to be extended). Each of these two theorems can be used to design a polynomial-time recognition algorithm for even-hole-free graphs; we will discuss those later, in §2.5.5.

Incidentally, here is another structural property of even-hole-free graphs (which is neither a structure theorem nor a decomposition theorem).

DEFINITION

D28: A vertex v in a graph G is a ***bisimplicial vertex*** if the set of neighbors of v in G is the union of two cliques.

FACT

F8: [AdChHaReSe08] Every non-null even-hole-free graph has a bisimplicial vertex.

Star cutsets (along with other "tightly structured" cutsets) are used in the proof of Fact F8 as a way to reduce the problem to a smaller graph and use inductive arguments.

2.5.3 Structure Theorems

In contrast with the previous subsection, here we survey some results that provide explicit structural descriptions of $\mathcal{F}$-free graphs for some families $\mathcal{F}$. In each of the classes of graphs now considered, the proof of the structure theorem follows a certain outline: first a decomposition theorem is proved, and then it is shown that the decompositions can be reversed and turned into "compositions" (that is, ways to glue two smaller graphs in a class in such a way that the resulting graph is also in the class). One then analyzes the effect of repeatedly performing the composition operations, starting from basic graphs, and an explicit structural description emerges. Theorems 6.2 and 7.3 of [Vu13] provide further examples of decomposition theorems that can be turned into compositions.

Claw-Free Graphs

DEFINITION

D29: A ***claw*** is the complete bipartite graph $K_{1,3}$, and a graph is ***claw-free*** if it is $K_{1,3}$-free (in other words, no vertex has three pairwise non-adjacent neighbors).

EXAMPLES

E4: Claw-free graphs are a generalization of line graphs (it is not difficult to see that if $G = L(H)$ for some graph H, then G is claw-free). But there are others. For example, the skeleton of the icosahedron (this is the unique 5-regular planar graph on 12 vertices) and the Schläfli graph (a highly symmetric 27-vertex graph that comes up naturally in the geometry of polytopes) are examples of claw-free graphs that are far from being line graphs.

DEFINITIONS

Another interesting subclass of claw-free graphs is called *circular interval graphs*.

D30: Let Σ be a circle, and let $F_1, \ldots, F_k \subseteq \Sigma$ be homeomorphic to the interval $[0, 1]$, such that no two of $F_1, \ldots, F_k$ share an end-point. Now let $V \subseteq \Sigma$ be finite, and let G be a graph with vertex set V in which for distinct $u, v \in V$, u is adjacent to v if and only if $u, v \in F_i$ for some i. Such a graph G is called a ***circular interval graph***. If in addition no three of $F_1, \ldots, F_k$ have union Σ, then G is a ***long circular interval graph***.

D31: A ***composition of strips*** is a generalization of a line-graph: given a graph H, in $L(H)$ every edge of H is replaced by a vertex, and vertices that correspond to edges that share an end are made adjacent; in a composition of strips that corresponds to H, every edge of H is replaced by a member of 1 of 15 prescribed families of graphs (a "strip"), and then certain edges are added between the subgraphs corresponding to edges of H that share an end.

FACT

The main theorem of [ChSe08] gives an explicit description of all claw-free graphs. To state this theorem precisely would take several pages, so let us instead describe it roughly.

F9: [ChSe08] Every connected claw-free graph G with $\alpha(G) \geq 4$ is either a ***a thickening of a long circular interval graph*** (this is a slight generalization of a long circular interval graph), or a ***composition of strips***.

Obviously, all graphs G with $\alpha(G) = 2$ are claw-free, so it remains to construct those claw-free graphs G for which $\alpha(G) = 3$. This turns out to be the most complex part of both the proof and the statement of the main theorem of [ChSe08]; graphs obtained in certain ways from the skeleton of the icosahedron are one class of claw-free graphs with stability number three, but there are many others that we will not describe here.

Quasi-Line Graphs

DEFINITION

D32: A graph is a ***quasi-line graph*** if each of its vertices is bisimplicial.

EXAMPLES

E5: Every line graph is quasi-line.

E6: Every quasi-line graph is claw-free.

E7: The 5-wheel (C_5, together with a vertex complete to its vertex set) is claw-free and not quasi-line.

E8: Long circular interval graphs are quasi-line and not line graphs.

FACT

The structure of quasi-line graphs is described in [ChSe12], and it is much simpler than that of general claw-free graphs. The main result of [ChSe12] states that:

F10: [ChSe12] There are only two kinds of connected quasi-line graphs: thickening of circular interval graphs (again, this is a slight generalization of circular interval graphs), and compositions of strips, with only two kinds of strips permitted.

REMARK

R5: In [ChPl13] an explicit structural description of **perfect claw-free graphs** is given (these were originally studies in [ChSb88] and [MaRe99]).

Bull-Free Graphs

DEFINITIONS

D33: The ***bull*** is the graph with vertex set $\{v_1, v_2, v_3, v_4, v_5\}$ where $\{v_2, v_3, v_4\}$ is a clique, v_1 is adjacent to v_2, v_4 is adjacent to v_5, and there are no other edges.

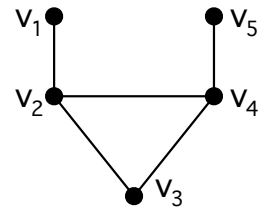


Figure 2.5.1: The graph known as the bull.

D34: A graph G is ***bull-free*** if no induced subgraph of G is isomorphic to the bull.

REMARK

R6: The structure of bull-free graphs was originally studied in connection with perfect graphs in [ChSb87], but that paper contains only decomposition theorems, and no structure theorem. The structure of general bull-free graphs was described in [Ch12, Ch12a], and we discuss it next.

FACTS

F11: Since the bull is isomorphic to its complement, the class of bull-free graphs is closed under complementation.

F12: [Ch12, Ch12a] There are basically two types of bull-free graphs, $\mathcal{T}_1$, and $\mathcal{T}_2$, which are described in what follows.

DEFINITIONS

D35: Let us first define the ***substitution operation***. Given disjoint graphs H_1 and H_2, each with at least two vertices, and $v \in V(H_1)$, we say that H is ***obtained from*** H_1 ***by substituting*** H_2 ***for*** v, or ***obtained from*** H_1 ***and*** H_2 ***by substitution*** (when the details are not important) if:

- $V(H) = (V(H_1) \cup V(H_2)) \setminus \{v\}$,
- $H|V(H_2) = H_2$,
- $H|(V(H_1) \setminus \{v\}) = H_1 \setminus v$, and
- $u \in V(H_1)$ is adjacent in H to $w \in V(H_2)$ if and only if u is adjacent in H_1 to v.

D36: A graph is ***prime*** if it is not obtained from smaller graphs by substitution.

FACTS

F13: The bull is a prime graph.

F14: Substituting one bull-free graph for a vertex of another bull-free graph results in yet another bull-free graph.

If follows from Facts F13 and F14 that to understand bull-free graphs, it is enough to construct all prime bull-free graphs.

EXAMPLES

E9: Graphs with no clique of size three are bull-free.

E10: Graphs with no stable set of size three are bull-free.

E11: However, there are prime bull-free graphs that contain both a big clique and a big stable set: let $n \geq 3$ be an integer, and let G be a graph with vertex set $k_1, \ldots, k_n, s_1, \ldots, s_n$ where

- $\{k_1, \ldots, k_n\}$ is a clique,
- $\{s_1, \ldots, s_n\}$ is a stable set,
- for $i, j \in \{1, \ldots, n\}$, s_i is adjacent to k_j if and only if $i + j > n$.

This richness of examples suggests that the structure theorem, if it exists, should be quite complex; and indeed it is. Once again, it would take several pages to state the construction explicitly, so instead we will try to give the reader a flavor of the result.

DEFINITION

D37: A graph F is ***triangle-free*** if $\omega(F) \leq 2$.

INFORMAL DEFINITIONS

D38: Graphs in $\mathcal{T}_1$ consist of a triangle-free induced subgraph F, together with a disjoint union of cliques$K_1, \ldots, K_t$, and the edges between $V(F)$ and $\bigcup_{i=1}^{t} K_i$ are carefully controlled. Thus, even though graphs in $\mathcal{T}_1$ may have both a large clique (one of $K_1, \ldots, K_t$) and a large stable set (in F), these two structures tend to "live" in different parts of the graph.

On the other hand, in the class $\mathcal{T}_2$, large cliques and large stable sets happily intertwine.

D39: Let G be a graph. Let $a, b \in V(G)$ be distinct vertices, and let $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_m\}$ be disjoint subsets of $V(G)$ such that $A \cup B = V(G) \setminus \{a, b\}$. Let us now describe the adjacency in G.

- a is complete to A and anticomplete to B.
- b is complete to B and anticomplete to A.
- the adjacency between a and b is not specified.

- If $i, j \in \{1, \ldots, n\}$, and $i < j$, and a_i is adjacent to a_j, then a_i is complete to $\{a_{i+1}, \ldots, a_{j-1}\}$, and a_j is complete to $\{a_1, \ldots, a_{i-1}\}$.
- If $i, j \in \{1, \ldots, m\}$, and $i < j$, and b_i is adjacent to b_j, then b_i is complete to $\{b_{i+1}, \ldots, b_{j-1}\}$, and b_j is complete to $\{b_1, \ldots, b_{i-1}\}$.
- If $p \in \{1, \ldots, n\}$ and $q \in \{1, \ldots, m\}$, and a_p is adjacent to b_q, then a_p is complete to $\{b_{q+1}, \ldots, b_m\}$, and b_q is complete to $\{a_{p+1}, \ldots, a_n\}$.

Under these circumstances we say that G is a 1-***thin graph***.

D40: A variant of this is called a 2-*thin* graph, but we omit the definition here.

D41: Graphs in $\mathcal{T}_2$ are built from 1-thin and 2-thin graphs by gluing them together in prescribed ways.

D42: Another class of bull-free graphs, called $\mathcal{T}_0$, consists of a few sporadic bull-free graphs. We omit the precise definition of $\mathcal{T}_0$ in order to avoid getting too technical.

FACT

To state Fact F15 precisely we would need to define "expansions", but let us regard them in their non-technical sense, as "slight generalizations". The main result of the series [Ch12, Ch12a] is the following:

F15: [Ch12, Ch12a] Let G be a bull-free graph. Then either

- the graph G is obtained from smaller bull-free graphs by substitution, or
- one of the graphs G, G^c is an expansion of a member of $\mathcal{T}_0 \cup \mathcal{T}_1 \cup \mathcal{T}_2$.

Moreover, every graph obtained this way is bull-free.

REMARK

R7: In [ChPe13], a similar description of **perfect bull-free graphs** is obtained.

2.5.4 Trigraphs

In this subsection we describe an object slightly more general than a graph, called a *trigraph*. Trigraphs have proved to be a useful tool in describing structural properties of graphs with certain induced subgraphs excluded. Roughly, trigraphs are used to record the fact that the adjacency between a certain two vertices, say u and v, cannot be determined from the condition that the graph is $\mathcal{F}$-free for the family $\mathcal{F}$ in question. In most cases that means that each of u and v can be replaced by a set of vertices (with certain restrictions on it), say U and V, where the adjacencies between members of U and the members of V are arbitrary. This in turn gives rise to notions such as "thickening" or "expansion" that were briefly mentioned in §2.5.4.

Let us now be more formal.

DEFINITION

D43: A ***trigraph*** G consists of a finite set $V(G)$, called the ***vertex set*** of G, and a map $\theta : V(G)^2 \to \{-1, 0, 1\}$, called the ***adjacency function***, satisfying:

- for all $v \in V(G)$, $\theta_G(v, v) = 0$
- for all distinct $u, v \in V(G)$, $\theta_G(u, v) = \theta_G(v, u)$
- for all $u \in V(G)$, there exists at most one $v \in V(G) \setminus \{u\}$ such that $\theta_G(u, v) = 0$

The idea is that, while a graph has two kinds of vertex pairs uv with $u \neq v$, adjacent and non-adjacent ones, a trigraph has three kinds: adjacent (for which $\theta(u, v) = 1$), non-adjacent (for which $\theta(u, v) = -1$), and semi-adjacent (for which $\theta(u, v) = 0$). A good way to think of semi-adjacent vertex pairs is as vertex pairs whose adjacency is "undecided". "Deciding" the adjacency of the undecided pairs results in a graph.

A version of trigraphs was first used in [Ch03, Ch06], where the last condition of the present definition was omitted. However, it seems that in order to study families of graphs, the more restricted definition that we use here is both sufficient and much nicer to work with; see [Ch03, Ch06, ChSe08, ChSe12, ChPl13, Ch12, Ch12a, ChPe13, ChKi14, Pe13].

Using Trigraphs: Structure Theorems

Let us now explain the use of trigraphs in a little more detail. Suppose that we are trying to understand the structure of the class of $\mathcal{F}$-free graphs for some family $\mathcal{F}$ (denote this class by $\mathcal{G}_{\mathcal{F}}$), and we can prove a theorem of the following form: we describe a few classes of $\mathcal{F}$-free graphs, and then say that every graph in $\mathcal{G}_{\mathcal{F}}$ is obtained from members of these classes by "expanding" certain vertex pairs uv. This means that u and v are replaced by two disjoint sets of new vertices, U and V, respectively, with arbitrary adjacencies between members of U and members of V. In order for this construction to be explicit, we need to provide a description of all pairs $(G, \mathcal{P})$, where G is a graph in $\mathcal{G}_{\mathcal{F}}$, and where $\mathcal{P}$ is the set of vertex pairs of G that can be expanded. To accomplish that, instead of working with $\mathcal{G}_{\mathcal{F}}$, we consider the class $\mathcal{T}_{\mathcal{F}}$ of *$\mathcal{F}$-free trigraphs*; these are trigraphs that have the property that however the adjacency of the undecided pairs is decided, the resulting graph is $\mathcal{F}$-free. Now we prove a similar theorem for trigraphs in $\mathcal{T}_{\mathcal{F}}$, where the vertex pairs that can be expanded are precisely the semi-adjacent pairs of the trigraph. This summarizes the way in which trigraphs are used in a purely structural setting.

Using Trigraphs: Algorithms

More recently, trigraphs have been used in the setting of algorithms. Algorithms that are based on decomposition theorems usually work as follows: take a graph G, "break it apart" via a decomposition given by the theorem, construct two new graphs G_1, G_2, where each G_i consist of a "piece" of G, together with a few more vertices recording the information about the remainder of G (let's call this part of G_i the *marker* for G_i); process each of G_1, G_2 separately, and then put the results together to get an answer for G. It turns out that semi-adjacent pairs are a good way to keep track of the information recorded in the markers for G_1 and G_2. Trigraphs were used in this way in [ChTrTrVu14].

2.5.5 Recognition Algorithms

In this subsection, we discuss the question of testing if a given graph G is $\mathcal{F}$-free for a given family $\mathcal{F}$. Obviously this can be done in polynomial time if $\mathcal{F}$ is finite (just run through all subsets X of $V(G)$ of size at most $\max_{F\in\mathcal{F}}|V(F)|$ and check if the subgraph $G|X$ is isomorphic to a member of $\mathcal{F}$), so this question is only of interest when $\mathcal{F}$ is infinite. For brevity, let us say "testing for $\mathcal{F}$" when we mean "testing if a graph is $\mathcal{F}$-free".

After Fact F2 was proved, the following major question in the theory of perfect graphs remained open: given a graph G, test (in polynomial time) whether G is perfect. With Fact F2 in our ammunition bag, this turns into a question of testing if a given graph is Berge (we remind the reader that a graph is Berge if it is $\mathcal{C}$-free, where $\mathcal{C}$ is the family consisting of all odd cycles of length at least five, and their complements). This was done in [ChCoLiSeVu05], and we will describe the algorithm briefly (the remainder of this subsection is closely based on [ChSe07]).

DEFINITION

D44: A ***pyramid*** is a graph consisting of a triangle $\{b_1, b_2, b_3\}$, called the *base*, a vertex $a \notin \{b_1, b_2, b_3\}$, called the *apex*, and three paths P_1, P_2, P_3, such that for $i, j \in 1, 2, 3$

- the ends of P_i are a and b_i,
- if $i \neq j$ then $V(P_i) \setminus \{a\}$ is disjoint from $V(P_j) \setminus \{a\}$ and the only edge between them is b_ib_j, and
- at most one of P_1, P_2, P_3 has length one.

In this case we say that the pyramid is *formed* by the paths P_1, P_2, P_3.

Let $\mathcal{P}$ be the family of all pyramids.

FACTS

F16: It is easy to see that every pyramid contains C_n for some odd $n \geq 5$, and therefore every Berge graph is $\mathcal{P}$-free.

It turns out that:

F17: [ChCoLiSeVu05]. Testing for $\mathcal{P}$ is relatively easy and can be done in time $O(|V(G)|^9)$.

This is the first step of the algorithm of [ChCoLiSeVu05].

Testing for Pyramids: the Shortest Paths Detector

The idea is as follows. If G contains a pyramid, then it contains a pyramid P with the number of vertices smallest. We are going to "guess" (by trying all possibilities) some of the vertices of P in G, then find shortest paths in G between pairs of vertices that we guessed that were joined by a path in P, and then test whether the subgraph of G formed by the union of these shortest paths is a pyramid. If the answer is "yes", then G contains a pyramid, and we stop. Surprisingly, it turns out, that choosing the shortest paths with a little bit of care, we can guarantee that if the answer is "no", then there is

no pyramid in P. We call this general strategy of testing for a family $\mathcal{F}$ a *shortest-paths detector* for $\mathcal{F}$. Let us now be more precise.

DEFINITIONS

D45: For $u, v \in V(G)$ we denote by $d_G(u, v)$ the length of the shortest path of G between u and v.

D46: If P is a pyramid, formed by three paths P_1, P_2, P_3, with apex a and base $\{b_1, b_2, b_3\}$, we say its ***frame*** is the 10-tuple

$$a, b_1, b_2, b_3, s_1, s_2, s_3, m_1, m_2, m_3$$

where

- for $i = 1, 2, 3$, s_i is the neighbor of a in P_i
- for $i = 1, 2, 3$, $m_i \in V(P_i)$ satisfies $d_{P_i}(a, m_i) - d_{P_i}(m_i, b_i) \in \{0, 1\}$

D47: A pyramid P in G is ***optimal*** if there is no pyramid P' with $|V(P')| < |V(P)|$.

FACT

F18: [ChCoLiSeVu05] Let P be an optimal pyramid, with frame

$$a, b_1, b_2, b_3, s_1, s_2, s_3, m_1, m_2, m_3$$

Let S_1, T_1 be the subpaths of P_1 from m_1 to s_1, b_1, respectively. Let F be the set of all vertices non-adjacent to each of s_2, s_3, b_2, b_3.

1. Let Q be a path between s_1 and m_1 with interior in F, and with minimum length over all such paths. Then a-s_1-Q-m_1-T_1-b_1 is a path (say P_1'), and P_1', P_2, P_3 form an optimal pyramid.
2. Let Q be a path between m_1 and b_1 with interior in F, and with minimum length over all such paths. Then a-s_1-S_1-m_1-Q-b_1 is a path (say P_1'), and P_1', P_2, P_3 form an optimal pyramid.

Analogous statements hold for P_2, P_3.

F19: Fact F18 can be used to design an algorithm to test for $\mathcal{P}$:

- guess the frame $a, b_1, b_2, b_3, s_1, s_2, s_3, m_1, m_2, m_3$ of an optimal pyramid P of G, by trying all 10-tuples of vertices;
- find shortest paths between m_1 and b_1, and between m_1 and s_1, not containing any neighbors of s_2, s_3, b_2, and b_3; do the same for m_2, b_2, s_2 and m_3, b_3, s_3;
- test if the union of the six shortest paths, together with the vertex a, forms a pyramid.

Now, by Fact F18, the answer is "yes" if and only if G contains a pyramid.

REMARK

R8: The algorithm in [ChCoLiSeVu05] is similar; it was modified a little to bring the running time down to $O(|V(G)|^9)$.

Easily Detectable Configurations; Cleaning; Finding Odd Holes

The next idea in [ChCoLiSeVu05] is to use the shortest-path detector for odd holes. Unfortunately, there does not seem to be a theorem similar to Fact F18 for odd holes, and so, first, the graph needs to be "prepared" for using a shortest-paths detector. The first step is to test for $\mathcal{P}$, and a few other families $\mathcal{F}$ that are easy to test for, and such that every Berge graph is $\mathcal{F}$-free. Now we can assume that the graph in question is $\mathcal{F}$-free for all these $\mathcal{F}$. The next step is applying *cleaning*, a technique first proposed in [CoRa93]. The idea of cleaning is to find, algorithmically, polynomially many subsets $X_1, \ldots, X_k$ of $V(G)$, such that if G contains an odd hole, then for at least one value of $i \in \{1, \ldots, k\}$ the graph $G_i = G \setminus X_i$ contains an odd hole that can be found using a shortest-paths detector. Finally, applying a shortest-paths detector for odd holes to each of $G_1, \ldots, G_k$, we detect an odd hole if and only if G contains one.

REMARKS

R9: In addition to the algorithm just described, [ChCoLiSeVu05] contains another algorithm to test for Bergeness, which instead of a shortest-paths detector for odd holes, uses a decomposition theorem for odd-hole-free graphs from [CoCoVu04], but we will not describe this algorithm here.

R10: Both algorithms in [ChCoLiSeVu05] test for Bergeness, and not for the existence of an odd hole in a graph. The complexity of testing if a graph contains an odd hole is still unknown.

Testing for Even Holes

On the other hand, the problem of testing if a graph contains an even hole can be solved in polynomial time. There are three (!) known algorithms. One is due to Conforti, Cornuéjols, Kapoor, and Vušković [CoCoKaVu02a], another to Kawarabayashi, Seymour and the author [ChKaSe05], and a third one to da Silva and Vušković [daSiVu13]. All three algorithms use cleaning; the first algorithm uses a decomposition theorem of [CoCoKaVu02] for even-hole-free graphs, the second one is based on the shortest-paths detector, and the last one is again decomposition based [daSiVu13].

More Algorithms

There are two other kinds of graphs that are somewhat similar to the pyramid, called a *theta* and a *prism*.

DEFINITIONS

D48: A ***theta*** is a graph consisting of two non-adjacent vertices s, t and three paths P_1, P_2, P_3, each between s and t, such that the sets $V(P_1) \setminus \{s, t\}$, $V(P_2) \setminus \{s, t\}$, and $V(P_3) \setminus \{s, t\}$ are pairwise disjoint, and the union of every pair of P_1, P_2, P_3 is a hole.

D49: A ***prism*** is a graph consisting of two disjoint triangles $\{a_1, a_2, a_3\}$ and $\{b_1, b_2, b_3\}$ and three paths P_1, P_2, P_3, with the following properties:

- for $i = 1, 2, 3$, the ends of P_i are a_i and b_i,
- P_1, P_2, P_3 are pairwise disjoint, and

- for $1 \leq i < j \leq 3$, there are precisely two edges between $V(P_i)$ and $V(P_j)$, namely, $a_i a_j$ and $b_i b_j$.

FACT

F20: Let $\mathcal{T}$ be the family of all thetas, and $\mathcal{P}r$ the family of all prisms. Then every even-hole-free graph is $\mathcal{T} \cup \mathcal{P}r$-free. (This is easy to check.)

In view of Fact F20 prisms and thetas play a similar role for even-hole-free graphs to the one that pyramids play for odd-hole-free graphs. It turns out, however, that:

FACTS

F21: [MaTr05] Unlike with $\mathcal{P}$, the problem of testing for $\mathcal{P}r$ is NP-complete.

F22: [LeLiMaTr09] The problem of testing for $\mathcal{P}r$ is NP-complete even in graphs with clique number two.

On the other hand,

F23: [ChSe10a] Testing for $\mathcal{T}$ can be done in polynomial time.

F24: [MaTr05] Testing for $\mathcal{P} \cup \mathcal{P}r$ can be done in polynomial time.

F25: [ChKa08] Testing for $\mathcal{T} \cup \mathcal{P}r$ can be done in polynomial time.

The Three-in-a-Tree Problem

All the algorithms mentioned so far, except one, use variations on the ideas of cleaning and shortest paths detectors (or decomposition theorems), and that one exception is the algorithm for testing for $\mathcal{T}$. There the approach is different. In order to be able to test for $\mathcal{T}$, a slightly more general problem is studied: given a graph G, and three vertices v_1, v_2, v_3 of G, does there exist an induced subgraph T of G, such that T is a tree and $v_1, v_2, v_3 \in V(T)$? This is the *three-in-a-tree* problem.

It turns out that the answer to this question is "no" if and only if the graph admits a certain structure. This fact is then used to design a polynomial time algorithm for the three-in-a-tree problem. Now, if $\{v_1, v_2, v_3\}$ is a stable set of size three with a common neighbor w in G, the degree of each of v_1, v_2, v_3 in $G \setminus \{w\}$ is one, and the degree of w in G is three, then the answer to the three-in-a-tree problem with input $(G \setminus \{w\}, v_1, v_2, v_3)$ is "yes" if and only if G contains a theta using v_1, v_2, v_3, w.

On the other hand, if $\{v_1, v_2, v_3\}$ is a clique of size three, and no vertex of G has two neighbors in it, then the answer to the three-in-a-tree problem with input (G, v_1, v_2, v_3) is "yes" if and only if G contains a pyramid with base $\{v_1, v_2, v_3\}$. Thus, the algorithm to solve the three-in-a-tree problem can be used, after some pre-processing, to test both for $\mathcal{P}$ and for $\mathcal{T}$ (and this is the only algorithm known to test for $\mathcal{T}$). This result is particularly pleasing from the point of view of a structural graph theorist, because this is one of the few times that a structure (and not just a decomposition) theorem and an algorithm appear together in the study of graphs with forbidden induced subgraphs.

As we have seen, the complexity of testing for $\mathcal{F}$ varies with $\mathcal{F}$: for some families polynomial-time algorithms are known, while for others the problem can be shown to be NP-complete. An interesting open question is: what causes this difference? Can one characterize the families for which testing can be done efficiently?