GRAPH GENERATION

by

Charles Joseph Colbourn

Research Report CS-77-37

Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada

November 1977

Graph Generation

by

Charles Joseph Colbourn

A THESIS

PRESENTED TO THE UNIVERSITY OF WATERLOO

IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

MASTER OF MATHEMATICS

IN

THE DEPARTMENT OF COMPUTER SCIENCE

WATERLOO, ONTARIO, CANADA

I hereby declare that I am the sole author of this thesis. I authorize the University of Waterloo to lend it to other institutions or individuals for the purpose of scholarly research.

Signature Charles J Colbourn

I further authorize the University of Waterloo to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Signature Charles J Colbourn

The University of Waterloo requires the signatures and addresses of all persons using or photocopying this thesis. Please sign below, and give address and date.

Graph Generation

Abstract

Exhaustive generation of lists of graphs is discussed. The presentation is given primarily as a survey of previous work; appropriate further research is proposed. The state of the art in exhaustive generation is that no algorithms requiring less than exponential time per output graph have been given, with the exception of generating restricted classes such as trees. Emphasis in this study is placed on the relation of graph generation to other fields of study. The relation between enumeration and generation is considered; this yields a technique for isomorph rejection in some generation methods. The relation between existential and exhaustive generation is also discussed; a proposal for a constructive correspondence is set forth which may supply an efficient family of algorithms for restricted classes of graphs.

A recent efficient method due to Read can be modified to generate rooted trees, graphs, digraphs, multigraphs, tournaments, connected graphs, graphs with a given subgraph, set systems, or hypergraphs. This specialisation of methods has a functional dual, generalisation, which is

examined. Similar improvements to the graph generator proposed by Heap, and the generation method for locally restricted graphs originally set forth by Farrell, are also discussed.

The importance of graph generation within the wider context of combinatorial generation is considered. Throughout this work, a survey of the state of the art is provided, with especial reference to the interrelations of the many independent investigations in the literature.

Acknowledgements

I shall, I'm sure, be unable to generate an exhaustive list of the many people who have been so very helpful. I wish to thank especially my supervisor, Kelly Booth, for his patience and support. His ability to view methods from a general perspective has proven invaluable.

My debt to Ron Read, whose work motivated much of this thesis, will extend far beyond the duration of this work. His positive attitude throughout gave me an enjoyment of mathematics which I shall forever treasure. I owe much also to many enlightening conversations with Rudi Mathon. His uncanny ability to see instantly to the heart of a problem gave me clues to many fruitful lines of research.

Many conversations were engaged in with three of my colleagues, Dan Zlatin, Pat Dymond, and Marlene Jones Colbourn. Their unfailing help gave me the incentive to continue when the final product was yet a dream. I wish also to thank Marlene for the correction and clarification of the many parts of the first draft, where I couldn't see the forest for the trees. I would like to thank the many people with whom I discussed aspects of this work, a few of

whom are Ian Munro, Adrian Bondy, and Marty Tompa; also I thank the many people who answered my inquiries about previous work, notably: Derek Corneil, Kee Dewdney, K.R. James and W. Riha, Joshua Lederberg, David Matula, J.J. Seidel, Donald Shaver, and A. Trojanowski. I wish also to express my appreciation to Elizabeth Creith Zlatin for the graphics employed in the dedication.

In conclusion, I gratefully acknowledge that financial support was received from the Ontario Graduate Scholarship Committee, and the National Research Council of Canada.

Contents

Table of Illustrations

2 Cait/ Zlatin 9/77

Chapter 1: Introduction

1.1 The problems; the aims

We examine the known methods of exhaustive generation of graphs, especially the production of a list of all nonisomorphic graphs on n vertices. The investigation of these methods is closely related to many other graphical generation problems. We therefore incorporate an investigation of the wider topic of generation of graphs with given parameters. The relation of diverse topics such as enumeration and existential generation to exhaustive generation is shown.

Our initial goal was to provide both theoretical and practical improvements to existing algorithms. In the study of graph generation methods, we felt this aim was best served by considering candidates for isomorph rejection strategies. We desired to bypass the isomorph elimination phase which is the dominant cost of the practical methods in use today. In the course of the investigation, we searched in vain for an adequate survey of graph generation techniques. Our primary goal from then

on was to produce both a comprehensive and comprehensible survey of these techniques. Our initial aim, although now secondary, is still important.

The reader being introduced to this discipline may be unprepared both for the number and quality of the available methods. Our aim thus incorporates the requirement that a logical structure be imposed on the format of this study to enable the reader to group methods into similar classes. The strategy for this logical structure will be explained as a preface to the thesis, in section 1.3.1.

Our final aim is to highlight at all stages the common general techniques rather than specific implementations of methods. The techniques to which we refer are used both for building new methods, and for adapting old methods to other classes.

One constraint is that the thesis avoids the detailed examination of particular methods which would inhibit our emphasis on general techniques.

1.2 Current use and future application

The reader may be unfamiliar with a number of applications for these methods. The need for catalogues of graphs arises in such diverse studies as chemistry, network

design, numerical analysis, and graph theory.

1.2.1 Graph theory

Many authors ([Read76b], [Baker74], for example) have observed that it is common practice to hypothesise a relation concerning structures, and then verify that this relation holds for all small structures. One generally proceeds to a proof or disproof of the conjecture based in no small way on this verification. The verification of conjectures requires a catalogue of the appropriate class of graphs. This approach to research is used extensively in graph theory due to the existence of many long standing conjectures. Two famous examples are the Four Colour Conjecture (now Theorem), and the Reconstruction Conjecture ([Bondy76]). This motivation is best expressed in [Faradzev76], where Faradzev states that "combinatorial theory as a scientific discipline finds itself in a botanical stage of development and therefore has a strong need for a large quantity of facts (a herbarium)".

One can make a stronger statement concerning the use of catalogues. Harary has stated that the existence of these catalogues often suggest conjectures to the mathematician, who then applies theoretical methods in investigating the conjecture ([Harary77]).

Many other graph theoretic applications of catalogues have appeared in the literature; the reader should refer to [Weinberg71] for an appreciation of them.

1.2.2 Networks

The network designer's problem is to produce good networks given constrained resources. He may wish to maximise some parameter of the network over all available networks. If the designer can specify a mandatory set of criteria, he can maximise over the graphs in an available catalogue. He may wish to do this in preference to accepting a heuristic suboptimal solution; if so, he will require a catalogue ([Weinberg71]).

1.2.3 Chemistry

The chemist's problem can be viewed as the dual of the network designer's. The network designer is concerned with synthesis, the chemist with analysis. The chemist will be interested in the analysis of the structure of a molecule based on some information obtained experimentally in the laboratory. There will typically be a few graphs which satisfy the known information. An exhaustive list of these graphs will often enable the chemist to deduce the structure of his molecule. These observations were made by Lederberg in his pioneering papers [Lederberg64] and

[Lederberg65].

This application is a currently active one; the artificial intelligence program DENDRAL employs just such techniques ([Feigenbaum71]). Another active application area is the use of catalogues of blocks in physical chemistry ([Sykes66], [Domb67]). The need for exhaustive catalogues in chemistry is compelling. It was expressed in [Dyson47], and was rapidly under investigation ([Gordon48]). A text on chemical information systems, [Lynch71], supplies the following impressive figures. In 1971, over 40,000 chemical journals were in print, referencing over 500,000 chemical compounds a year, over 75,000 of which were first synthesised in that year.

1.2.4 Numerical analysis

It has been shown that rooted trees find application in the production of Runge Kutta formulae for numerical solution of differential equations ([Lawson70]). Catalogues of rooted trees will be required for the derivation of higher order Runge Kutta methods; moreover, the existence of such catalogues will facilitate the automation of this process.

1.2.5 A compendium of other applications

In the design of algorithms, one often wishes to analyse the expected time complexity of a method. In the event that one does not wish such a formal analysis, it is still useful to test the program on a representative data sample. It is an unfortunate truth that, with graph algorithms in particular, many authors have made unreasonable conjectures about the performance of their method following testing on a sample of their own choice rather than a truly random sample. A case in point is the graph isomorphism problem, which has taken on the character of a "disease" as a result ([Read76a]). One way of avoiding this problem is to employ a random data sample from existing catalogues as test data.

Another application arises in the study of electronics, the circuit layout problem. Some criteria for the characteristics of a circuit layout will be expressed; a search of a catalogue of planar graphs will reveal an exhaustive set of candidates with the desired structure.

1.3 An overview of the research

We next outline the format which the survey and results will follow. We present this in preparation for the survey work itself to acquaint the reader with the basic underlying observations on which the work is based.

The field of graph generation has previously been examined as a collection of specialised methods whose interrelations were unknown, or at best not widely known. The net result has been a duplication of research and methods, each specialised to one particular application.

The usefulness of this study will be increased if the reader can view these interrelations and maintain them in the foreground in the remainder of this work. We therefore present a strategy which is both a justification for the study and a method of interpreting it.

1.3.1 A strategy for examining graph generation

Mathematicians employ two fundamental techniques in the investigation of problems. We refer to them as the dual techniques of generalisation and specialisation. The method of generalisation is to examine methods for a restricted subproblem of the original problem. One then attempts to apply a more general form of the method to a less restricted subproblem. The dual process, specialisation, is to consider general methods and exhibit efficient restrictions of them to subproblems.

Another approach is to employ observations in related topics to assist in the analysis at hand. One may, for example, consider parallels between enumeration and

generation problems. One may also employ existential generation methods as partial solutions to their exhaustive counterparts.

1.3.2 An overview of the chapters

In this chapter we have informally introduced the field of graph generation and some applications of its methods. We have proposed some techniques which will be used to motivate a unified presentation. Here we present a brief look ahead to show how our strategy is used to subdivide the area into manageable segments.

The second chapter introduces prerequisites to the understanding of the work. It will also consider where this study lies in relation to the wider field of combinatorial generation. The third chapter focuses on the generation of a restricted class of graphs, trees. A logical progression of generation methods from very restricted to less restricted classes is traced out. The strategic motivation for the examination of tree methods is the anticipation of generalising them to larger classes.

The fourth chapter provides a short introduction to the related field of existential generation. In the latter portion of the chapter, a correspondence is developed between existential and exhaustive techniques. The fifth

chapter examines the successes and failures of the direct approach. Observations from graph enumeration are employed to devise a scheme for isomorph rejection. Some novel observations due to Read are shown to generalise to other methods.

The sixth chapter involves the specialisation of direct methods to restricted classes. A representative sample of restricted classes together with a statement of the success of specialisation methods for the classes is included. The latter portion is dedicated to exhibiting the relation of this work to other active research in generation of configurations. The seventh chapter concludes the thesis. It is a statement of our success in realising our avowed goals, and a suggestion of further topics.

1.3.3 New results

This thesis is primarily, but not exclusively, a survey of known results. We have supplied small extensions to previous work. All methods discussed which are not ours are referenced in the appropriate place.

1.4 Historical notes

A quick reading list has been compiled for the reader

interested in tracing the development of graph generation methods.

Cayley, a pioneer in the field of tree enumeration, seems tc have been the first researcher. His classic work, [Cayley1857], alludes to the relation between generation and enumeration and offers insight into deriving enumeration formulae from generative approaches. This work is continued in [Cayley1859], where he investigates binary trees.

Cayley also published the first known catalogue of trees in [Cayley1875]. Trees on fewer than ten vertices are listed.

In an investigation of polyhedra, Bruckner catalogued 3-connected trivalent planar graphs for $n \leq 16$ ([Bruckner1900]); (much later, this list was extended by Grace using a computer ([Grace65])).

A hiatus in research then occurred, lasting until 1946. In that year, Kagno published the first catalogue of graphs on fewer than seven vertices ([Kagno46]). Research did not commence in earnest, however, until the appearance of a catalogue of digraphs on fewer than six vertices, produced by computer ([Read66]). At the same time, Heap succeeded in extending Kagno's catalogue to include the

seven and eight vertex graphs, also using a computer ([Heap72]). In 1966, the generation of regular graphs by computer was also introduced in the literature ([Izbicki67]).

Chapter 2: Definitions

In this chapter, we shall introduce terminology and prerequisite material required throughout the thesis. We also supply a list of references concerning related material.

2.1 Prerequisites

The terminology of graph theory has traditionally been complicated by an abundance of synonymous terms; despite the current trend toward standardisation, the area of graph generation still suffers from a lack of universally accepted terms. We endeavour to remedy this problem by employing a standard nomenclature throughout.

The structure of solution methods is also characterised so as to avoid undue repetition when methods are introduced.

2.1.1 The basic types of problems

In combinatorics as a mathematical discipline, one is concerned with problems of existence and enumeration. The

former asks if there is a structure with certain parameter values; the latter asks how many such structures exist. There is a strong parallel with problems in combinatorial computing. The first class of methods we call existential generation. These problems require, given a set of parameter values, the production of at least one structure satisfying the parameters, if any such exist. The second class is called exhaustive generation (or simply generation) methods. These problems require an exhaustive list of the nonisomorphic structures which meet the given constraints.

The area of generation has been referred to by many names. A few are: constructive enumeration ([Faradzev76]), enumeration ([Evans67]), and construction.

A further problem is the topic of probabilistic methods. The theoretical aspects of probabilistic methods are discussed in [Erdos74]. This dynamic area of combinatorics treats the problem of determining how likely a certain structure is to have a specified property. There is an algorithmic counterpart to these methods, but the parallel is not as direct in this case. Probabilistic generation algorithms address themselves to the problem of supplying some graph G from a list of all nonisomorphic graphs with given parameter values such that the selection

of every graph on the list is equally likely.

Each of these methods will surface in some investigations within the thesis, especially in relation to exhaustive generation.

2.1.2 The structure of existing methods

Existing methods for combinatorial generation problems have a reasonably fixed format despite many surface differences. The typical algorithm is a two step process. Starting with an initial list of some structures, one applies some operations on the structures from the list to produce new structures. One then eliminates isomorphs (also called duplicates) from this new list of structures. This process is performed repeatedly until no new structures are produced. This second step is called isomorph elimination, and is performed so as to maintain only one representative from each isomorphism class. This desire is discussed in [Golomb60]; we quote in preference to attempting to state it more clearly:

> "Every collector's basic desire is to have 'one of each'. This is true not only in philately and numismatics, but almost anywhere that classification into categories applies."

These methods subdivide naturally into two classes of

algorithms. One can define the content of a structure in some convenient manner; for graphs, this will typically be the number of edges, or vertices. In the inductive algorithms, the operations either uniformly increase or decrease the content. Algorithms which preserve content are called incremental. Incremental algorithms appear to be new candidates for graph generation methods. An operation in an incremental algorithm is one by which a local (or incremental) change is made in the structure to produce a new structure.

2.1.3 Isomorph elimination vs. rejection

As we noted, the second step of the generation procedures is isomorph elimination. Elimination of isomorphs requires the testing of graph isomorphism pairwise with every other graph in the list. This can be accomplished by direct application of graph isomorphism testing on each possible pair of graphs, or the graphs can be coded and the list searched for this code. The isomorph elimination step is generally divided into two phases, one of coding the graph, and one of sorting or searching the output lists.

Both the isomorphism and the coding problem have been recently surveyed in [Read76a]. The selection of coding

methods in preference to pairwise isomorphism testing is discussed in [McKay76]. Recent heuristic studies on the coding problem can be found in [Proskurowski74], [Overton75], and [McKay76]. It is instructive to observe that early investigations of canonical forms for graphs appear in the investigation of chemical systematics ([Gordon48]).

In the early development of combinatorial algorithms, it was observed that backtrack did not cope well with the problem of producing isomorphic output configurations ([Swift60]). This led Swift to propose a method called isomorph rejection; it was to be used in conjunction with backtrack, introduced in [Walker60]. The philosophy of isomorph rejection is fundamentally different from that of isomorph elimination. Elimination methods are run as independent steps to remove isomorphs from the output. Isomorph rejection, on the other hand, rejects computations which produce isomorphic results prior to execution. Isomorph rejection techniques have been employed with success on combinatorial problems ([Wells71], [Whitehead73]). The application of rejection in graph generation has not yet been fully examined.

We will not concern ourselves with the details of elimination methods. One must simply remember that the

elimination step divides naturally into two phases; one phase codes the graph, the other sorts or searches the output list to check whether duplications exist.

2.2 An annotated guide to background material

With few exceptions, the graph theoretic definitions used in this study can be found in [Harary69], and the combinatorial ones in [Hall67]. An excellent discussion of related enumeration problems can be found in [Harary73]. For definitions concerning the isomorphism and coding problems, see [Read76a], and [Read76b]. The Polya enumeration theory is somewhat more difficult. The initial paper by Polya, [Polya37], is a strikingly clear presentation of both theory and applications. Polya theory has been developed well in the expository paper [deBruijn64]. Some further investigations appear in [deBruijn71]. A gentler treatment of the subject is given in the introductory text [Liu68].

Trees demand a more detailed introduction. The appearance of trees in many computational problems has left a wealth of restricted classes of trees; an appropriate understanding of this fact can be obtained from [Aho74], for example. In keeping with tradition, we shall introduce our own subset of the available definitions.

A *tree*, or *free tree*, is a connected graph with no cycles. A *rooted tree* is a tree in which one vertex has been distinguished; this vertex is called the *root*. The *height* of a rooted tree is the length of the longest path from the root to a leaf; the height of a vertex is the length of the path from that vertex to the root. A *planted tree* is a rooted tree in which an ordering has been applied to the vertices adjacent to the root. A *plane planted tree* is a planted tree drawn in the plane. This is equivalent to applying an ordering to the adjacencies of all vertices, not just those of the root.

A *k-ary tree* is a plane planted tree with the restriction that each vertex has k distinct *points of attachment* for sons, not all of which must be in use. Thus a vertex having a single son has k different possibilities, each of which produces a unique tree. When k=2, the tree is *binary*. A *complete* k-ary tree is a k-ary tree in which every vertex has exactly 0 or k sons.

The requisite algebra for this study is minimal; see [Birkhoff65] or [Lipson75].

Backtrack is a popular generation technique. It has shown many promising applications in graph algorithms ([Tarjan72], [Read73], for example). The application of

backtrack as a generation technique appears in connection with block designs ([Gibbons76]). Another related generation application has appeared for semigroups ([Plemmcns67]); many other applications are listed in the excellent survey [Corneil74]. The use of backtracking in graph generation problems is small, but it should be kept in mind.

One promising existential technique, hill climbing, has had some success in the generation of set packings, Latin squares, and block designs ([Colbourn77], [Tompa75], [Shaver73]).

Chapter 3 : Trees

Efficient methods for restricted classes of graphs often generalise in a nice way to larger families. Two examples of interest are the use of tree isomorphism algorithms to produce both a planar graph isomorphism algorithm ([Hopcroft72]), and an interval graph isomorphism algorithm ([Booth76]). This is sufficient motivation in itself; one initial benefit in the search for good methods is that efficient generation methods for planted trees and rooted trees have been examined in much more detail than the general problems. For these reasons, we discuss both classical and novel methods for tree generation. The only previous surveys of tree generation methods known to us appear in [Scoins67] and [Scoins68].

3.1 The enumeration of trees

There is a strong relation between problems of generation and enumeration. The scope of this correspondence is perhaps realised most fully in the generation of various forms of trees, notably binary trees. We introduce enumeration methods for binary trees and

related families. Although other enumeration methods for trees are somewhat constructive, the resulting construction methods are inductive algorithms requiring an inordinate amount of isomorph elimination. The path by which efficient algorithms for general classes of trees are obtained is via the efficient methods for plane planted and binary trees. For the enumeration methods of the other classes, the reader is referred to [Harary73].

3.1.1 The Catalan numbers and binary trees

One advantage of having explicit enumeration formulae for structures in a given class is that when two apparently different classes have the same enumerator, often *a posteriori* a one-one constructive correspondence can be shown. Such a procedure was used to show that a subclass of the trivalent plane planted trees are in direct correspondence with a subclass of the plane planted trees ([Harary64]). Many more intuitive correspondences were then found between the two classes ([deBruijn67]).

We are here concerned with the most famous (and, apparently, most common) enumerator of trees. This enumerating function produces the Catalan numbers (some authors, notably [Wells71], refer to them as the Segner numbers).

The n'th Catalan number is C(2n,n)/(n+1). Some of the many structures enumerated by the n'th Catalan number are: binary trees on n vertices, stack sortable permutations on n elements, ballot sequences and difference sequences with n terms, complete binary trees on 2n+1 vertices, plane trees rooted at an endvertex with n+1 edges, and trivalent plane trees rooted at an endvertex with n vertices of degree three ([Rotem75], [Rotem77], [Knott77], [Harary64], [deBruijn67], [Wells71]). Similar observations have been noted for k-ary trees ([Klarner69]). The importance of this is that when an efficient constructive correspondence is shown, one can generate one type of structure by generating the other. These methods have been applied with great success; some of the resulting generation algorithms will be discussed next.

3.2 The planted plane tree

In this section, we discuss the generation of plane planted trees, and some restrictions of this class.

3.2.1 Binary trees -- more definitions

Every interior vertex v has two adjacencies x and y of greater distance than v from the root. The vertex v is the *father* of x and y; the vertices x and y are the *left and right sons* of v.

3.2.2 Binary Trees I

One obvious generation procedure for binary trees is the classical method. To find all binary trees on n vertices, one first finds the lists for all smaller numbers of vertices. Each binary tree is created by taking a root, and attaching an i-vertex tree to its left attachment, and a (n-i-1)-vertex tree to its right for all i-vertex and (n-i-1)-vertex trees.

This algorithm is representative of the inductive algorithms, with the exception that no elimination is required. The initial list L(0) is the null graph. Each list can then be obtained given that all smaller lists have been found. Producing complete binary trees is equally straightforward. We modify the base step to let L(1) be the trivial tree on one vertex, and require that the root have both subtrees non-null.

There is one drawback. In order to generate lists on n vertices, the generation of lists on all smaller number of vertices is required. This may create insurmountable time and storage overhead.

3.2.3 Binary Trees II

The method we discuss here is developed in [Rotem75] and [Rotem77].

A stack sortable permutation is a permutation which does not have a subsequence a,b,c such that b > a > c. A sequence B = b(i) is a ballot sequence iff b(i) ≤ n - i and b(i) ≥ b(i-1). Each stack sortable permutation corresponds to a ballot sequence in the following way. If we let b(i) be the number of elements in the stack sortable permutation P which are to the right of i in P and are greater than i, B is a ballot sequence. This correspondence is one-one ([Rotem77]).

We define a difference sequence as follows. A sequence D = d(i) is a difference sequence iff the sum of its elements is n, d(1) > 0, and each d(i) ≥ 0. The relation with ballot sequences is direct. The first element of D is n - b(1). For 2 ≤ i ≤ n, d(i) is simply b(i-1) - b(i). This correspondence is one-one.

Many sequence types have been introduced which have the same enumerator as binary trees. Can we exhibit a one-one correspondence between one of these classes and the class of binary trees? The answer is shown to be positive by the following algorithm. This method will take a difference sequence D = d(i), and produce a binary tree.

Maintain an initially empty stack S of vertices during the execution of the algorithm. For the first entry in D, d(1), create the first vertex, and append a path of d(1)-1 left sons to it. The last vertex in this path is labelled 1, and the other vertices are stacked in turn following the path away from the root. Then for $2 \leq i \leq n$ we do the following. If d(i) = 0, we pop the stack and label the popped vertex i, and return. Otherwise, a right son r is appended to the vertex labelled i-1, and a path of d(i)-1 left sors is appended to r. The vertex furthest from r on this path is labelled i. The other vertices are stacked starting at r and following the new path to its end at i.

The generation of ballot sequences can be done in linear time, and the transformations to difference sequences and binary trees are also linear (Note that each element is pushed and popped at most once on S).

The algorithm is an improvement over the classical method. The storage requirements are linear in n.

A similar derivation of this method appeared after Rotem's work in [Ruskey77a].

3.2.4 Binary Trees III

In an investigation of numbering schemes for binary

trees [Knott77] develops a generation algorithm for binary trees.

A tree permutation is defined to be a permutation which can be written in the form xBA where x is a single element, A is a tree permutation of the elements x+1,...,n, and B is a tree permutation of the elements 1,2,...,x-1. Given a tree permutation P, one can produce a binary tree as follows. The permutation on 0 elements is the null graph. Otherwise, label the root of the tree x, find the left son by producing a binary tree from A, and the right son from B. A binary tree produces a tree permutation simply by traversing it in preorder. The tree permutations of [Knott77] are precisely the stack sortable permutations of [Rotem77]. Knott's algorithm for generating the list of tree permutations is order n^2. We conclude that Rotem's algorithm is the appropriate generation method for binary trees.

3.2.5 Complete Binary Trees

An improvement on the classical algorithm for complete binary trees has been published in [Wells71]. A correspondence has been shown between complete binary trees and partitions without crossing. The definition is quite complicated, but the concept is straightforward. One

partitions the elements {1,2, ..., n} into classes which contain only even, or only odd, numbers, and no class contains all the even (odd) numbers. Draw a circle with points 1 through n on the perimeter, such that i is adjacent to i-1 and i+1 (using n+1 as 1). Edges are added outside the circle connecting points. Connected points are in the same class. The partition is a partition without crossing if no two lines cross in the drawing and no additional line can be drawn without crossing an existing one. Given a partition without crossing P, a complete binary tree is constructed as follows.

The left son of the root is a vertex labelled 0, the right son a vertex labelled 1. Then, for the lowest numbered 'unscanned' vertex v, if it is in a class of cardinality greater than one, we do the following. Let s be the next highest number in v's class. Let t be the number closest to v which is not yet in the tree. (The absolute value of v and t differ by one). The left son of v is the even number of s or t; the right son the odd. This is done until all vertices are in the tree.

This correspondence is shown to be one-one in [Wells71], and a construction algorithm for partitions without crossing is presented.

3.2.6 k-ary Trees

In the obvious way, the classical algorithm for binary trees can be modified to generate k-ary trees by taking all partitions of n-1 into k parts (parts may be empty). This method compounds the difficulties noted for binary trees. We discuss an alternative algorithm.

Given a k-ary tree, we define an augmenting set for the tree as v(1),v(2), ..., v(l) where v(1) is the root, v(i) is the rightmost son of v(i-1) for $2 \le i \le l$, and v(l) is an endvertex. Recall from the definition that in a k-ary tree, the edges are strictly ordered, and therefore adding to the right of an empty subtree is allowed (that is, say v has one son; there is a distinction between the son being in the i'th position rather than the j'th for $i \neq j$). Thus, we define the augmenting bounds of a vertex in the augmenting set as follows. The augmenting bound of v(l) is [1,k]; for $1 \le i < l$, the augmenting bound of v(i) is [v(i+1)+1,k]. We generate k-ary trees on n vertices as follows

```
LIST L(1), L(2), ..., L(n);
L(1) := {the trivial tree};
for i := 2 until n do
    L(i) := ø;
```

```
for each t ∈ L(i-1) do
    Let v(1), ..., v(l) be the augmenting set for
    t;
    for j := 1 until l do
        Let [m,k] be the augmenting bounds for
        v(j);
        for posn := m until k do
            s := t;
            In s, append a son to v(j) as its
            posn'th son.
            Add s to L(i);
```

Each k-ary tree is generated at least once. To see that each tree is generated exactly once, note that there is a unique tree from which it could be produced. This is combined with the uniqueness of the trees in L(n-1) to give uniqueness in L(n).

Recently, new methods for k-ary trees based on the methods of Rotem and Knott have been announced ([Ruskey77b], [Trojanowski77a], [Trojanowski77b]).

3.2.7 Planted plane trees

The problem of generating plane planted trees is essentially different from generation of k-ary trees. We must now cope with isomorphism; many selections in the

generation of k-ary trees are not available, since they produce isomorphic results. In particular, augmenting bounds do not have relevance in a plane planted tree.

The generation of plane planted trees is a topological problem, in that only non-null subtrees are considered in the 'ordering' of a vertex's descendants. Thus, a trivial modification of the method for k-ary trees suffices to generate them. We define an augmenting set as before, but only allow additions in the leftmost augmenting position. The algorithm to generate plane planted trees proceeds as follows:

```
LIST L(1), ..., L(n);
L(1) := {the trivial tree};
for i := 2 until n do
    L(i) := ø;
    for each t ∈ L(i-1) do
        Let v(1),...,v(l) be the augmenting set for t;
        for j := 1 until l do
            s := t;
            In s, append a vertex to v(j) as its
            rightmost branch;
            Add s to L(i);
```

Note that, as before, there is a unique predecessor in

L(n-1) for each graph in L(n), hence each plane planted tree is produced exactly once.

A recursive implementation of this method is given in [Scoins68].

3.2.8 Another class of planted plane trees

A generation method for a subclass of the planted plane trees is discussed in [deBruijn67]. De Bruijn and Morselt restrict the class of planted plane trees to those rooted at an endvertex. They show a correspondence between planted plane trees rooted at an endvertex and binary trees in which the root has degree one, and all interior vertices have degree three (a trivalent plane tree). We discuss the most intuitive of the correspondences, the 'geometric' correspondence.

Each trivalent plane tree is drawn in the plane so that for each vertex of degree three, its two sons are drawn one upward, one to the right. The downward line is the direction to the root. From this representation, a planted plane tree is constructed as follows. The vertices of the plane tree are all the vertices of degree three, the root, and a new vertex v which will be the root of the new planted plane tree. It is connected to the root of the trivalent plane tree. Every other vertex x is connected to

its 'up' son y, and all vertices reachable from y by following branches to the right only. Note that the plane embedding of the resultant tree is determined by the plane embedding of the original. A diagram is given here to illustrate the construction. The trivalent tree is in dashed lines; the planted plane tree in dotted lines. A numbering is supplied to show the correspondence. The vertex 1 is the root of the trivalent tree; the vertex 0 is the root, v, added in the construction.

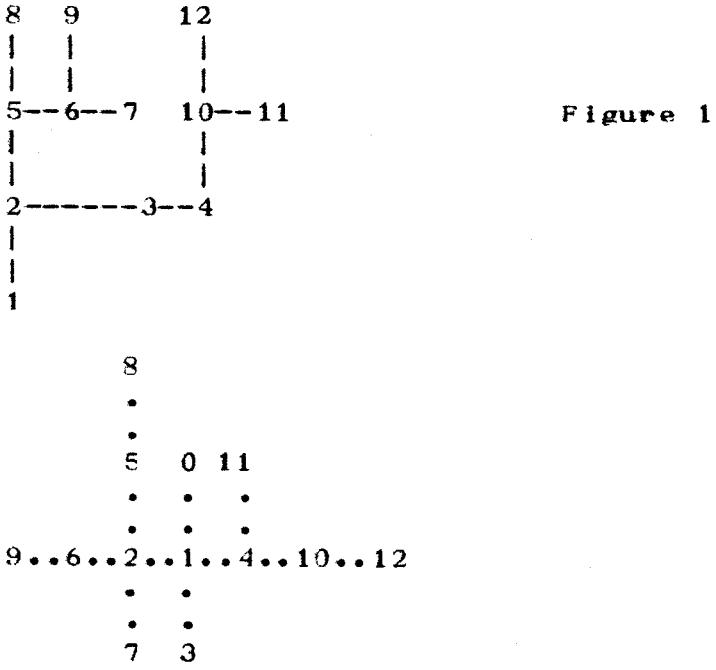


Figure 1

3.3 The rooted and the planted tree

We generalise the methods for plane planted trees to the more difficult cases of planted and rooted trees.

3.3.1 Rooted trees and planted plane trees

There is a (surjective) map from the set of planted plane trees onto the set of rooted trees, such that two planted plane trees map to the same rooted tree iff they are isomorphic without the ordering (plane) restriction. We wish to generate only one of the planted plane trees from each partition of the set of planted plane trees induced by this mapping. An ordering may be applied to a rooted tree to make it a planted plane tree (in fact, the planted plane tree with the 'largest' code of all planted plane trees which map to the given rooted tree) as described in [Read72]. A linear time algorithm to code trees has appeared in the literature ([Hopcroft72]).

We will not discuss the classical algorithm for rooted trees here; for a description of it, see [Read69].

3.3.2 Rooted Trees I

An algorithm for the generation of rooted trees is developed in [Read76b] which is similar to the algorithm for plane planted trees. The only modification is that as a tree is produced, one must verify that it is in canonical form. In [Read76b], it is shown that each canonical rooted tree is generated exactly once. The algorithm is nonlinear in the number of output graphs; we are motivated to find

an algorithm which does not produce the 'extra' trees (those not in canonical form). Read's method is a classic isomorph elimination algorithm. The question we pose concerns the existence of an isomorph rejection algorithm which bypasses the elimination step. The answer, discussed later, is affirmative.

3.3.3 Rooted Trees II

We next outline an algorithm for rooted trees from [Scoins68]. The algorithm employs a representation of trees called the height representation. This representation can be defined algorithmically. Perform a depth first search on the tree, commencing at the root. As a vertex is scanned for the first time, output its height. A rooted tree is (here) said to be in canonical form when its height representation is lexicographically maximal.

An implementation of Scoins' algorithm is given in [Scoins68]. The algorithm performs in time at worst linear in the size of the output list, and is therefore in some sense optimal.

A very recent improvement has been discovered by Bayer ([Read77]). We describe this method, since it generates trees while making no mention of trees. It generates a set of vectors in lexicographic order; it turns out that this

set of vectors is precisely the set of height representations of rooted trees. This is the first example of a general technique of representing graphs as vectors, and generating vectors.

We now introduce the vector production method. From a given n-vector from {1, 2, ..., n}, one produces the 'next' vector (if possible) as follows. Scan the vector right to left until an element is found which is at least three. Call this element r. Continue scanning until an element is found whose value is one less than that of r, and call this element l. A vector s is found by reading off the elements left to right starting at l until one reads the element before r. One then deletes all entries from l on in the original vector, and replaces them with sufficiently many copies of s to make the vector length n. The last copy of s is truncated, if necessary.

The method will compute a next vector unless r cannot be found, in which case no next vector exists. As initial vector, one supplies the vector whose i'th entry is i. One then applies the method until no next vector exists.

An illustration is helpful here. The following vectors are produced for the case n=5, in this order:

1 2 3 4 5
1 2 3 4 4
1 2 3 4 3

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

We have underlined the 'l' and 'r' in each vector (tree). One can easily verify that this is the list of all height representations of rooted trees on five vertices. This algorithm is the most efficient known; its worst case complexity is linear, but its average case complexity remains unknown.

3.3.4 Rooted Trees III

Two algorithms have been presented. One is the classic isomorph elimination algorithm. The second is an isomorph rejection algorithm. Although it is very efficient for trees, it does not seem to be a generally applicable method in the generation of graphs. One is interested in a method which performs efficient isomorph rejection as a means of characterising the kinds of algorithms appropriate for more difficult cases.

As with enumeration problems, the difficulty arises in coping with symmetry. The problem of exploiting known symmetries in enumeration is met by an elegant theorem of Polya ([Polya37]). The problem in generation, however, has

no generally efficient method of solution. This area is largely uncharted, despite the impetus provided in [Swift60]. Often the reason given is that computing the symmetries is more costly than a brute force solution to the basic problem. This, however, is not the case with trees. The automorphism group of a tree can be found in time linear in the number of vertices. (This will be proved in 3.4).

We define a numbering of the vertices of a canonical rooted tree as that numbering in which the root is numbered 1, and the adjacency matrix is maximal subject to this constraint.

Maximality implies that for any vertices v and w, if the height of v is less than the height of w, then v is numbered before w. The sons of any vertex are numbered such that the sons numbered later have degree no larger than that of the sons numbered earlier.

Before employing these observations in the design of an algorithm, we introduce two partitions of the vertex set which will be required. The automorphism partition is a partition in which two vertices are in the same class iff they are similar. We also introduce a coarser partition than this, the <u>degree partition.</u> Two vertices are in the

same degree class iff they are at the same height, have the same degree, and their fathers are in the same degree class.

We propose the following algorithm for the generation of rooted trees. Let L(n-1) be the list of all rooted trees in canonical form on n-1 vertices.

```
For each tree t in L(n-1) do
    Compute the automorphism partition of t.
    Compute the degree partition of t.
    Let H be the highest numbered interior vertex.
    For i := H until n do
        set s = t.
        if i is the lowest number in its automorphism
        class, and in its degree class, append a
        vertex labelled n to i in s, and add s to
        L(n).
```

We claim that this algorithm produces each canonical tree exactly once, and further that it produces only canonical trees.

We now prove that this algorithm performs as required.

Theorem 1: Every tree produced is canonical.

Proof:

We suppose (to the contrary) that a tree t is produced which is not in canonical form. In t, we let x, y be the first instance of two brothers whose interchange is required in placing t in canonical form. We now examine two cases.

Case 1: deg x < deg y, but x is numbered before y.

Consider the tree which has been constructed up until the step where the first vertex is to be appended to x. Let this tree be t'. In t', x and y are similar vertices since they are endvertices with the same father. In the next (deg x) - 1 steps, exactly (deg x) - 1 vertices will be appended to x. (If this were not the case, x would not have been the highest numbered interior vertex, and then no further vertex could be appended to it). In a later (deg x) - 1 steps, (deg x) - 1 vertices will be appended to y, for the same reason. Let this tree be s. In s, x and y are again similar. If deg y is to be greater than the degree of x, the addition to y must be performed in this step (otherwise, y will no longer be the highest numbered interior vertex). An addition to y is impossible, however, since x is numbered before y, and x and y are similar.

Contradiction.

Case 2: deg x = deg y; x is numbered before y in t, but y is numbered before x in the canonical form.

Let h(r,l) be the number of vertices in the subtree rooted at r of height l (from r), and let S(r,l) be the set of these vertices listed from left to right (according to the labelling of t). Since deg x = deg y, h(x,1) = h(y,1).

Let c,d be the first two vertices (ie, at lowest height and leftmost in the S at that height) in the subtrees rooted at x and y respectively such that they occur in the same position in their S, and deg c > deg d. (If no such pair exists, t is in canonical form). Let a be the father of c, and b the father of d. At some stage in the algorithm, when vertices are being appended to d, deg c = deg d. In this tree c and d are placed in the same class by the 'degree constraint' algorithm. Now c is numbered before d, so the degree constraint disallows the addition of another vertex to the adjacencies of d. But then deg d ≤ deg c.

Contradiction.

Then for every pair of sons of any vertex in t, the order in t agrees with the order in canonical form. This contradicts the assumption that such a tree t in non-canonical form is produced. □

Theorem 2: Every rooted tree is produced exactly once.

Proof:

We will show that each tree has a unique addition sequence of vertices to produce a rooted tree t. Let T be the set of rooted trees in canonical form which the algorithm fails to produce. Let t be a member of T with fewest vertices. Let the father of the vertex labelled n be the vertex v. Since t is not produced, either v is not the lowest numbered vertex in its automorphism partition at the previous step, or v is degree constrained to have degree less than that of some vertex x. If the first is true, then let y be the lowest numbered vertex in the automorphism partition of v. Consider the tree t' constructed by appending the vertex n to y rather than v. This tree is isomorphic to t, yet has larger code. This is a contradiction since t is canonical.

In the second case, let v be degree constrained so that deg v $\leq$ deg x for some vertex x. Now x is numbered before v. All ancestors of v and x are of the same degrees at each height. Therefore, by increasing the degree of v, a larger code could be constructed for t by numbering v before x. This is true since the code will remain unchanged until the point where the sons of v (or x) are to be listed, and v has more sons than x. Thus, t is not in canonical form; contradiction.

Therefore, no such t exists, and T = ø.

Each rooted tree is produced no more than once since each tree can only be produced by the addition of the highest numbered vertex. □

3.3.5 Planted trees

Recall that a planted tree is a rooted tree with an ordering imposed on the adjacencies of the root. A simple modification of the rooted tree algorithms suffices to generate all planted trees. We will discuss briefly the modifications to our algorithm (from 3.3.4). In computing the automorphism partition of an input tree, we disallow automorphisms mapping one adjacency of the root onto another. This is equivalent to labelling uniquely all adjacencies of the root. In the computation of degree classes, all vertices at distance 1 from the root form a singleton class.

Each subtree at height two will be a rooted tree in canonical form, and thus the tree itself will be a planted tree in canonical form.

3.4 The automorphism group of a tree

It was noted earlier that the automorphism group (partition) of a tree could be computed in time linear in the number of vertices. Although this appears to be straightforward in light of the existence of a linear tree isomorphism algorithm, the details of such an algorithm appear not to have been published. Corneil has published an autcmorphism partition algorithm for trees; we begin our study there.

3.4.1 Corneil's algorithm

The algorithm in [Corneil68] takes a partition of the vertex set, initially the partition induced by vertex degrees, and produces successive refinements of this partition. Each refinement is defined in terms of the previous, by placing two vertices (which are currently in the same class) in different classes iff the set of classes of the adjacent vertices of one is not the same as the set of classes of adjacent vertices of the other. This is done until no finer partition is obtained (that is, fewer than n times).

This method, which is the basis of the more general automorphism partition algorithm of [Corneil72], is not sufficient for the general class of graphs; however, Corneil has shown that it is sufficient for trees, in that

the finest partition produced is the automorphism partition. The algorithm operates in order n^2 time.

The extension of this algorithm to rooted trees is appealingly straightforward; all one is required to do is place the root in a singleton class at the start of the successive refinements.

3.4.2 Tree isomorphism

Corneil's algorithm constitutes a solution to the tree isomorphism problem. One is motivated to inquire whether this is the best one can do. An alternative algorithm has appeared in many disguises for tree isomorphism (for example, [Scoins68] and [Lederberg64]). A proof that this tree isomorphism method is linear was given in [Hopcroft72]. We briefly discuss their algorithm here; for details see Hopcroft and Tarjan's work.

One first uniquely roots the tree at a central vertex in linear time. One then assigns heights from the root to each vertex. Starting at the highest level, one works progressively down towards the root, numbering two vertices at level k the same iff they have the same degree, and the numbers assigned to their sons are the same.

3.4.3 The automorphism partition of a tree

We first present an example, demonstrating the labelling produced by the algorithm of [Hopcroft72], to observe how the numbering is performed.

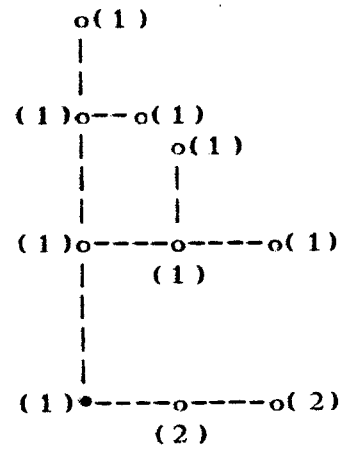


Figure 2

When one further partitions according to height, one obtains:

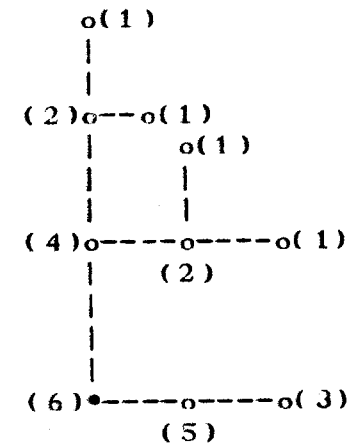


Figure 3

This numbering produces the automorphism partition of the tree for this example. One is led to ask if this is always the case. Many small examples refute this suspicion.

Consider, for example:

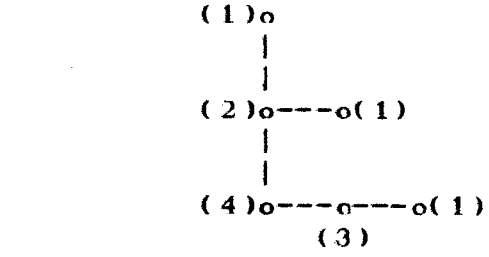


Figure 4

Since the algorithm doesn't produce the automorphism partition, what can we say about similarity? Let x, y be any two similar vertices. They are at the same height, and either x=y, or the father of x is similar to the father of y.

Starting with Hopcroft and Tarjan's labelling (in which vertex i has label $l_1(i)$), one relabels from the root upwards; we call this the l_2 labelling, in which vertex i is given label $l_2(i)$. The root is given the label 1. We start processing at height 1. In processing height i, two vertices x, y are given the same l_2 label iff their fathers have the same l_2 label and $l_1(x) = l_1(y)$. The l_2 labelling is, by the above observation, the automorphism partition.

This step is implemented in a similar manner to the iteration of the isomorphism algorithm. At each height, a bucket sort is done to place similar nodes in the same class, and then one assigns the same label to each label in that class. The bucket sort sorts label pairs of the form

$(l_1(x), l_2(f))$ where f is the father of x. Two vertices will have the same label pair iff they are similar.

This computation is linear time; to see this, simply observe that this added iteration performs the same sorts as the first. This result extends to the linear time computation of the automorphism partition of interval graphs ([Booth77]). The extension to planar graphs is open; the reader interested should take note of the linear time planar graph isomorphism algorithm ([Hopcroft74]).

3.4.4 The automorphism group of a tree

In light of the previous positive result, one is led to inquire whether the automorphism group of a tree can also be computed in linear time. This problem for graphs is known to be polynomially equivalent to the graph isomorphism problem ([Read76b]). For trees, however, the automorphism partition algorithm can be extended to list the generators of the automorphism group.

Having completed the l_2 labelling, one proceeds as follows. Perform a bucket sort of the l_2 labels of the vertices adjacent to the root. For each class of vertices with the same l_2 label, select one representative, x, from the class. For each other member of the class in turn, output a permutation mapping the subtree rooted at x onto

the subtree rooted at this vertex. Delete the root from the original tree, and retain only subtrees rooted at vertices which were selected as representatives of their l_2 class. Repeat this procedure recursively on each of the retained subtrees in turn.

The output with the standard representation of permutations may be of greater than linear length (consider, for example, an n vertex tree of height 1). Instead, one just outputs that part of the permutation which maps the subtrees, ignoring all vertices which are left fixed. At most n similarities will be shown; thus the output is of linear length in n.

Hence the computation of the automorphism group of a tree can be performed in time linear in the number of vertices.

3.5 Free trees

Read and Scoins have independently observed that the methods for rooted trees do not seem sufficient for free trees ([Read76b], [Scoins68]). Scoins noted, however, that central trees appear easy to generate from rooted trees and that bicentral trees are not significantly more difficult.

3.5.1 Central trees

For any tree of odd diameter, we can find a unique vertex (the centre) at which to root the tree. To generate all central trees, it then suffices to generate all rooted trees of odd diameter whose centre is the root. We can construct all rooted trees of given height by Read's algorithm. For each tree of given height h, we can construct all central trees of radius h+1 by simply appending a path of length h+2 to the root of the rooted tree, and then shifting the root of this tree to its centre. We constrain vertex addition so that no vertex is added at height greater than h+1. Read's method ensures that if an addition is made in the original tree, the result will be discarded since it will not be canonical. The rooted trees obtained by applying this modified algorithm are exactly the central trees desired. Note that no duplicates will be produced since Read's method verifies that each output graph is canonical.

3.5.2 Bicentral trees

We can generalise the above construction method to the bicentral case. Given a rooted tree t of height h, a path of length h+1 is appended to the root. The root of this tree is that vertex in the bicentre which is on the 'new' path. One then applies the modified rooted tree algorithm

to produce subtrees of height at most h, to produce all bicentral trees. We must ensure, in order to avoid isomorphic results, that the code of the subtree rooted at the root but excepting the subtree containing t is no larger than the code of t. (Otherwise, the tree will be produced with the other vertex of the bicentre as the root).

3.5.3 Some negative results concerning the automorphism partition

One wishes to consider an isomorph rejection algorithm to generalise the rooted tree methods without distinguishing the central and bicentral trees. The following inductive algorithm may be proposed. Let a (free) tree be in canonical form if the labelling of the vertices produces a maximal adjacency matrix. Let L(n-1) be the list of all nonisomorphic trees in canonical form on n-1 vertices.

```
Set L(n) = ø.
For each tree t in L(n-1) do
    Compute the automorphism partition of the tree.
    Let l be the highest numbered interior vertex.
    For k := l until n-1 do
        If k is the lowest numbered vertex in its
```

automorphism class, then let s = t; add the vertex n to the adjacencies of vertex k, and add s to L(n).

One duplicate produced by this method is

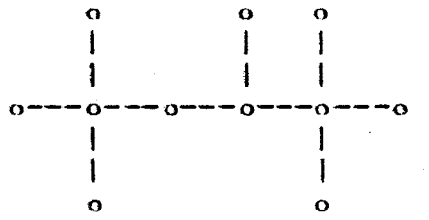


Figure 5

where the two numberings allowed by the algorithm are

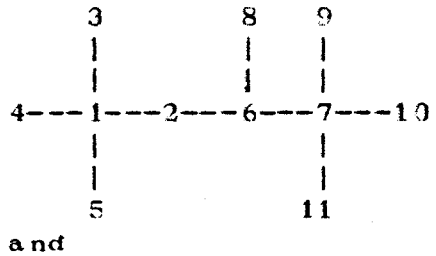
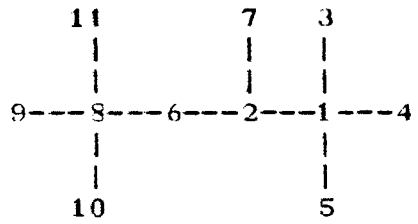


Figure 6

and



This in itself does not appear to be an insurmountable difficulty since it seems from experience that relatively few noncanonical trees are produced. The major drawback of this algorithm is that no linear procedure is known for placing a tree in this canonical form; the best known upper bound is $O(n^2)$.

3.5.4 Locally restricted free trees

James and Riha have presented an algorithm for the generation of trees with a given degree sequence (i.e., locally restricted trees) ([Riha74a], [Riha74b], [James74]). An algorithm for producing all partitions of the integer 2(n-1) into n parts must be supplied, such as that in [Riha76].

All such partitions are the degree sequence of some graph ([Menon64]), and those that are uniquely realisable as trees are characterised in [Hakimi63].

The method presented in [Riha74a] is applicable to the more general problem for graphs. The modification for trees will be discussed after a brief overview of the algorithm. James and Riha define an m-selection from a partition as a selection of m elements from the integers in the partition (this is more often called a subcomposition, e.g. [Farrell71]). In assigning the edges from a vertex of degree d, d-selections of the remaining vertices in the partition are chosen as the candidate adjacencies, and the chosen entries are decremented. This is done until the partition is all zero. This selection of edges is a graph. An isomorph elimination algorithm is then executed to remove duplicates from the output list.

The modification for trees is straightforward; one simply ensures that no cycles are produced; the result is certainly a tree.

Although James and Riha state "in this case (trees), the output is irredundant" ([Riha74a]), they note in [Riha74b] that an elimination algorithm is required. Computational experience shows that this elimination phase is mandatory.

In summary, observe that the algorithm will produce (possibly many) infeasible m-selections, and further will produce isomorphic output trees requiring elimination. The elimination method is a simple extension (see [Riha74b]) of the automorphism partition algorithm discussed earlier. The elimination step is nonlinear, and therefore the method does not appear to be as effective as the alternate methods.

Chapter 4: Existential Generation

One might with justification ask why existential generation is relevant in a study of exhaustive techniques. One motivation is that existential methods may suggest a fruitful area of research. Another is that, since existential methods solve part of their exhaustive counterpart, we may be able to extend them to the exhaustive case, by a 'divide and conquer' approach.

We propose an interesting relation between these two fields, which we call completeness. This opens a new area of study for exhaustive methods.

4.1 Some definitions

We review the basic terminology of existential generation methods ([Boesch76]). Given a sequence of integers, it is graphical iff it is the degree sequence of some graph; multigraphical iff it is the degree sequence of some multigraph. A graphical sequence is uniquely realisable if there is exactly one graph (up to isomorphism) with that degree sequence. The uniquely

realisable sequences are characterised in [Hakimi63]; more easily tested necessary and sufficient conditions for unique realisability have since appeared ([Li75], [Koren76]).

A vertex removal construction of a graph from a sequence is one in which some entry, x, in the sequence, is deleted, and x other entries are decremented by 1. This corresponds to the assignment of a set of x adjacencies for the deleted vertex. The remaining sequence is dealt with inductively. An edge removal algorithm is one in which two entries in the sequence are decremented by one at each step; these methods properly include the vertex removal methods.

Most of the problems discussed here were first introduced in the classic paper [Senior51].

4.2 Graphical sequences

Two characterisations of graphical sequences appear in the literature. The first is a nonconstructive characterisation due to Erdos and Gallai ([Erdos60]). We consider the second characterisation in more detail.

4.2.1 The characterisation of Havel-Hakimi

A constructive theorem published in [Havel55], and

independently in [Hakimi62], supplies the fundamental algorithm of existential generation. We let d = (d(1), ..., d(n)) be an integer sequence in nonascending order. Let d' be a sequence of length n-1, obtained from d by deleting the first entry, and then decrementing by 1 the first d(1) entries in the resulting sequence. Hakimi proves that d is graphical iff d' is. The sequence d' must be reordered into nonascending order to inductively test if it is graphical. The production of a realisation if the sequence is graphical is done in the following way. Each vertex has a label associated with it, so that in reordering the sequence, the vertex labels remain with the correct entry. We proceed as follows given a sequence d. Order d in nonascending order. Produce the sequence d' from d, and produce the edges between the label of the vertex in position 1 with the vertices in positions 2 through d(1)+1. Repeat the algorithm on d' until |d'| = 0. If at any stage a negative entry appears in the sequence, then the original sequence is not graphical.

4.2.2 A generalisation

One is led to inquire whether an effective generalisation of the Havel-Hakimi algorithm exists which produces other (nonisomorphic) realisations. The answer is affirmative ([Wang73]). The Havel-Hakimi algorithm

performs vertex removal on the entry which appears first in the sequence. Wang and Kleitman have shown that at any step, any vertex can be removed in the following way. Let j be an integer between 1 and n, and let d be an integer sequence. Create d' by first removing the j'th entry from d, then decrementing the first d(j) elements in the resulting sequence, and finally reordering the result.

The generalisation of Hakimi's result is that d is graphical iff d' is. In this way, nonisomorphic realisations may be created.

4.2.3 A clue for exhaustive techniques

One may inquire whether whether every graph can be produced by Wang and Kleitman's algorithm by judicious choice of the vertex to remove at each step.

The solution to this inquiry, which does not appear to have been considered, is negative. One graph which cannot be constructed by the algorithm is drawn here:

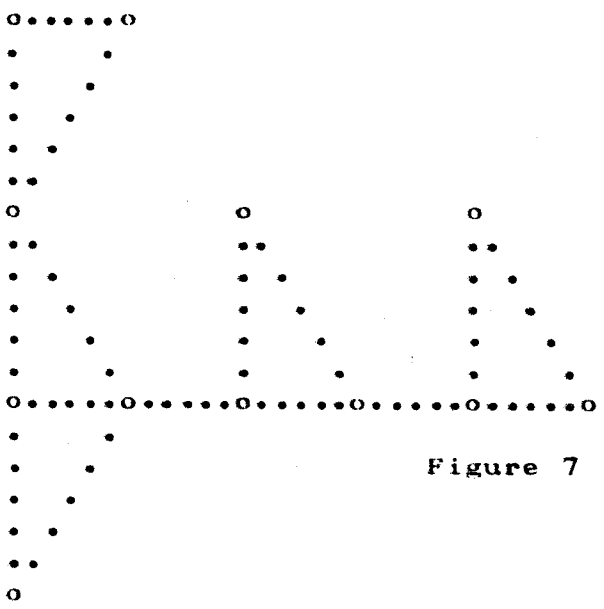


Figure 7

This strong negative result lends credibility to the belief that the algorithms for locally restricted graphs presented in Chapter Six, especially that of [Farrell71], are in some sense the best that we know how to do.

4.3 Connectivity and reliable networks

The problem which we consider in this section is one of prime importance; it is without question the most compelling motivation for the study of the generation of graphs. The aim is to produce reliable networks, or in graph theoretical terms, highly connected graphs. In an article on the construction of reliable networks ([Hakimi73]), Hakimi and Amin discuss the intimate relation

between graph theoretic formulations and network considerations for the problem.

4.3.1 The basic problem

Given a number of vertices n, and a number of edges m, what is the maximum (minimum) connectivity of any graph with n vertices and m edges? These important questions were first solved in [Harary62] by the following two theorems.

Theorem 1:

The maximum connectivity of a graph with n vertices and m edges is 0 if $m < n-1$, $\lfloor 2m/n \rfloor$ if $m \geq n-1$. □

Theorem 2:

The minimum connectivity of a graph with n vertices and m edges is $n(n-1)/2 - m$, or 0, whichever is larger. □

The importance of these results to network design is immediately apparent. They provide bounds on the reliability (unreliability) of the networks which can be constructed from given resources.

4.3.2 Connected graphs

In the version of the problem usually stated, one restricts the class of graphs to a given degree sequence. The Havel-Hakimi algorithm specifies no constraint on connectivity. In the construction of networks, however, it is mandatory that a network be connected.

To construct a connected network on a given degree sequence, one can employ Wang and Kleitman's algorithm with the stipulation that the j chosen be one corresponding to an vertex which has not yet been selected as the end of an edge, if there is one. In this way, vertices not yet in the component being constructed will be added to it. This will produce a connected network if one exists.

4.3.3 Biconnected graphs

In the design of networks, one major criterion is reliability, or redundancy. If one path fails, it is desirable to have an alternate path. Thus, network designers have attempted over the years to produce algorithms which construct highly connected networks. A minimum requirement for most network applications is that the network be 2-connected. We consider that case here.

The problem of generating biconnected realisations of a sequence is discussed in [Boesch74]. The existence problem was solved by the following theorem due to Hakimi

([Hakimi62]), and independently by Boesch and McHugh.

Theorem: An integer sequence d in nonascending order has a biconnected realisation iff

(1) d is graphical, and

(2) d(n) ≥ 2, and

(3) the number of edges in the resulting graph is no less than n-2+d(1). □

4.3.4 3-connected graphs

The existence problem for 3-connected realisations was later solved in [Rao70]. Rao and Rao's characterisation of those sequences having 3-connected realisations is given by the following theorem.

Theorem: Let d be an integer sequence in nonascending order. Then d has a 3-connected realisation iff

(1) d(n) ≥ 3, and

(2) d is graphical, and

(3) the number of edges is at least n-4 + d(1) + d(2).

□

4.3.5 k-connected graphs

The thrust of the research in this area is to produce conditions for the existence of k-connected realisations for arbitrary k. This is important to the network designer

in that it answers a question of paramount importance: what reliability can one expect to attain given the available degree sequence?

The characterisation of k-connected graphs was discovered independently by Hakimi ([Hakimi74]), and Wang and Kleitman ([Wang73]). The results are of great interest here since they are constructive. The characterisation, as formulated by Hakimi, is as follows.

Theorem:

Let d be an integer sequence in nonascending order. The sequence d is realisable as a k-connected graph (for $k \geq 3$) iff

(1) d is graphical, and

(2) $k \leq n-1$, and

(3) $d(n) \geq k$, and

(4) the number of edges is at least $((k-1)(n-1))/2 + 2d(1)$. □

We will discuss the construction method in [Hakimi74], rather than the construction of [Wang73]. The algorithm proceeds as follows. The vertex set for the graph to be constructed is $\{v(i) \mid 0 \leq i \leq n-1\}$. A graph $G_1=(V,E_1)$ is constructed by the following three rules.

1. If $i-j = 1,2, \ldots, \lfloor k/2 \rfloor \bmod n$, and $0 \leq i,j \leq n-1$, the

edge $(v(i),v(j)) \in E_1$.

2. If k is odd, and $j = i + \lfloor (n+1)/2 \rfloor \bmod n$ and $0 \leq i \leq \lfloor n/2 \rfloor - 1$, $(v(i),v(j)) \in E_1$.

3. If k is odd, and n is odd, $(v(\lfloor n/2 \rfloor), v(n-1)) \in E_1$.

□

The graph constructed in this way is k-connected ([Hakimi69]). The sequence obtained by subtracting the degrees in G_1 from those in G is graphical ([Hakimi74]). The construction of the remainder of the graph can be done by the Havel-Hakimi algorithm.

4.3.6 Maximally connected graphs

The network designer is often concerned not simply with attaining some connectivity k, but rather with attaining the maximum connectivity achievable with the given resources. This problem was considered in [Hakimi69], and later in [Hakimi74].

The problem of maximum connectivity with a given degree sequence can be solved simply by finding the largest k for which (by the theorem given) we can construct a k-connected realisation. Hakimi's algorithm is then applied to give us such a realisation.

Yap states that, in network design, a more restricted

class is usually required. Often, all nodes must be similar; thus he has compiled a list of vertex transitive graphs for $n \leq 13$ ([Yap73]).

4.3.7 k-edge connected graphs

A characterisation of degree sequences with k-edge connected realisations is given in [Edmonds64]. For $k>1$, the only criterion is that the lowest degree be at least k. For k=1, the further stipulation required is that there be at least n-1 edges.

4.4 Edge removal algorithms

The methods discussed to this point characterise the degree sequence based on the effect of the removal of a vertex on the sequence. Boesch and Harary, in [Boesch76], examine edge removal methods. The main theorem of their paper is as follows.

Theorem: Let $d = (d(1), \ldots, d(n))$ be an integer sequence in nonascending order. Let j be an integer ($1 \leq j \leq n$). Assume $0 < d(j) < n$. Let d' be the sequence obtained by decrementing the j'th element and the $d(j)$'th element not counting the j'th.

Then d is graphical iff d' is. □

Boesch and Harary show that some (seemingly intuitive)

edge removal algorithms fail. The motivation for examining edge removal methods is again the anticipation of finding exhaustive methods. If one could characterise which edges may be deleted while still maintaining the graphical status of the sequence (and which cannot), one could exhaustively generate all locally restricted graphs by simply deleting all edges, in turn, which leave the sequence graphical. One would proceed inductively.

4.5 Other parameter restrictions

The emphasis both in this thesis and in the literature has been on connectivity considerations. This is understandable due to the practical applications.

Other parameter restrictions have been investigated to a minor extent, notably the work of [Kleitman73] on characterising sequences which admit a realisation with a k-factor. Another characterisation is that of planar graphical sequences, sequences which can be realised as a planar graph ([Chvatal69], [Schmeichel77]). Further examples are the characterisation of sequences which admit only realisations which are line graphs ([Rao77]), and the partial characterisation of sequences for which all realisations are hamiltonian ([Nash-Williams70]). The discussion of these is beyond the scope of this work, as

the work inherent in discussing such a comprehensive subject would truly require a volume of its own.

4.6 Switching and completeness

We next introduce a novel and largely unexplored connection between the problems of existential and exhaustive generation. One anticipates being able to employ the powerful and well charted existential results, introduced briefly here, in solving their exhaustive counterparts.

4.6.1 Switching

The concept of a switching in a graph has been introduced in [Eggleton75], as follows. Let a, b, c, d $\in$ V; (a,b), (c,d) $\in$ E; (a,c), (b,d) do not belong to E. A switching of G is defined as deleting the edges (a,b) and (c,d) and adding the edges (a,c) and (b,d) to obtain G'. The degree sequence of G' and that of G are the same.

The concept of switching was earlier introduced in a different setting in [Shaver73]. Shaver introduces the concept for matrices. Let M be a {0,1} matrix. An interchange on M is defined to be the transformation of any 2 by 2 submatrix

i j i j

$$\begin{array}{c} \\ k \\ l \end{array}\begin{array}{c} \begin{array}{cc} \ulcorner & \urcorner \end{array} \\ \begin{array}{cc} 1 & 0 \end{array} \\ \begin{array}{cc} 0 & 1 \end{array} \\ \begin{array}{cc} \llcorner & \lrcorner \end{array} \end{array} \quad \text{into} \quad \begin{array}{c} \\ k \\ l \end{array}\begin{array}{c} \begin{array}{cc} \ulcorner & \urcorner \end{array} \\ \begin{array}{cc} 0 & 1 \end{array} \\ \begin{array}{cc} 1 & 0 \end{array} \\ \begin{array}{cc} \llcorner & \lrcorner \end{array} \end{array},$$

or vice versa (where $i < j$; $k < l$). The interchange as a switching on graphs is applied to the adjacency matrix of the graph, with the caveat that i,j,k,l be distinct integers. If this constraint is not specified, the (more general) operation will be referred to as matrix switching. Shaver employs this method of matrix switching in the generation of block designs. The method is examined further in [Tompa75].

We introduce the notion of the <u>completeness</u> of an operation. Let r,c be integer vectors of length n; let X(r,c) be the set of matrices over {0,1} for which the row sum of row i is r(i) and the column sum of column j is c(j). Further, let MS(M) be the closure of the matrix M under the operation of matrix switching (the set producible from M by matrix switching). Finally, let R(M) be the vector of row sums, and C(M) the vector of column sums in M.

The operation MS is <u>complete</u> if for any M, MS(M) = X(R(M),C(M)). The operation S of switching is complete if the closure of M under S is equal to that subset of

X(R(M),C(M)) which are adjacency matrices.

If the operation S of switching is complete, one can exhaustively generate all locally restricted graphs in two steps. One employs an existential generation algorithm to create one graph G. The closure under S of G is computed, and this comprises the set of all graphs with the same degree sequence as G. Ryser has shown that the operation of matrix switching is complete by the following theorem.

Theorem: Given two matrices M_1 and M_2, if $R(M_1) = R(M_2)$ and $C(M_1) = C(M_2)$ then there is a finite sequence of matrix switchings which produce M_2 from M_1.

Proof: ([Ryser57])

The extension of this proof to (simple) switching on graphs is immediate.

4.6.2 Completeness

Ryser's proof is the extent of the known relation between existential and exhaustive generation. The resulting algorithm is not efficient, since expensive isomorph elimination is required. We introduce another operation, constrained switching, on graphs. Let P be any property which a graph may exhibit (k-connectivity, k-colourability, k-particity, et cetera). The operation

CS(G,P) produces ϕ if G does not have property P; otherwise it produces those graphs which have property P obtainable from G by a switching. The set $\gamma(d,P)$ is the set of graphs with degree sequence d and property P.

The operation CS is <u>complete</u> for property P iff for any graph G (having degree sequence d(G)), the closure of G under CS is equal to $\gamma(d(G),P)$. Property P is <u>complete</u> if CS is complete for P.

The importance of completeness is this: if property P is complete, one can generate all locally restricted graphs with property P. If $|\gamma(d(G),P)|$ is small, the generation algorithm will be practical.

As a concrete example, consider the problem of generating all 15-connected graphs on 20 vertices. There are relatively few such graphs. If CS were complete for 15-connectivity, we could curtail the computation so as to generate (by constrained switching) and thus store only 15-connected graphs.

With these powerful incentives in mind, we are led to search for properties which are complete. It comes as no surprise, given the paucity of previous investigation, that for for most properties, we do not know whether they are complete or not.

4.6.3 Negative results

We now examine some properties known not to be complete.

One property which comes to mind is the property of exact k-connectivity (G is exactly k-connected iff it it is k-connected, but not (k+1)-connected). The first property we examine is exact 0-connectivity. The following two graphs are both disconnected, but switching constrained to exact 0-connectivity cannot produce one from the other:

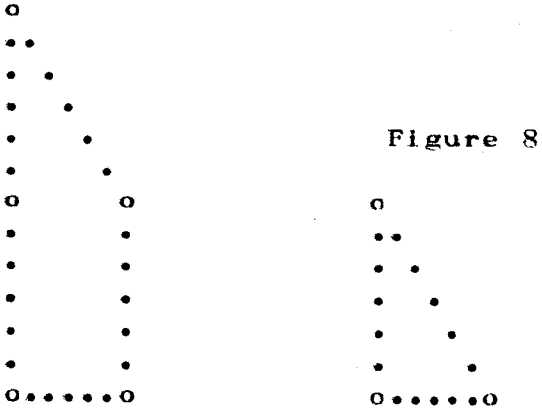
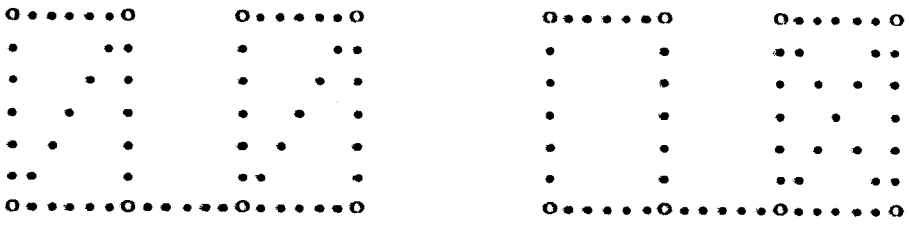


Figure 8

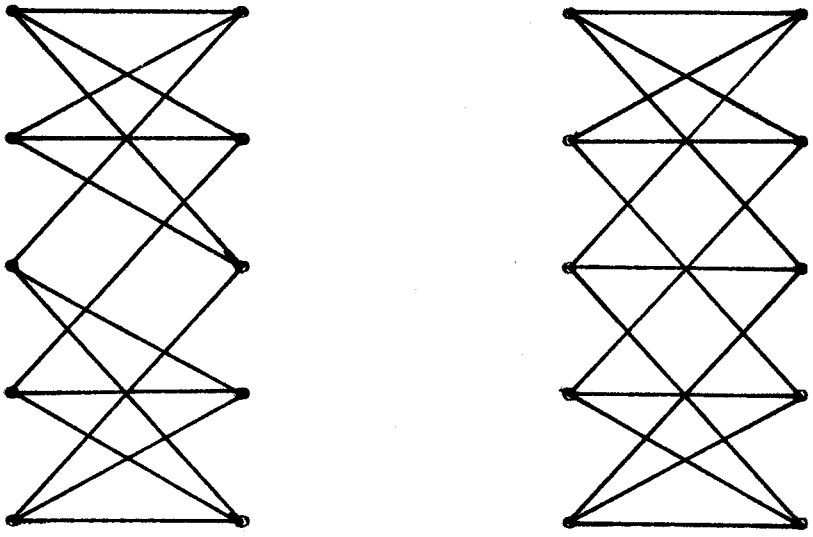
Exact 1-connectivity provides another false instance: Figure 9:



A construction of counteramples for the general case of exact k-connectivity has not been given.

Another property for which completeness fails is 2-colourability (or, equivalently, biparticity).

Figure 10:



One can easily verify that any switch produces an odd cycle in the graph.

The example furnished for exact 0-connectivity also constitutes a counterexample to the instance in which P is the property of having 2 components. This can be generalised in the obvious way to the property of having k components.

4.6.4 Positive results, conjectures, and suggestions for further research

The negative results, once obtained, do not show the non-existence of good algorithms; rather they show that this method does not produce them. A positive result, on the other hand, reveals an algorithm which may compare favourably to known methods. We discuss the few positive results known, and settle on some conjectures for future research.

Many graph theoretic properties of a trivial nature come to mind as candidates for completeness such as regularity, and other properties determined by the degree sequence.

One dismisses such instances immediately since they are in truth simply Ryser's theorem. One should not overlook the fact, however, that eulerian graphs are characterised by degree sequence and hence constitute an important family on which the algorithm may be effective.

Similarly, many degree sequences have only planar realisations ([Chvatal69]), and thus there is a restriction to planarity which is complete.

These trivial instances are insufficient to warrant investigation of completeness. A nontrivial example is the class of trees. The property of 'being a tree' is complete, as shown by the following theorem and observation.

Theorem: Let T_1 and T_2 be two trees, rooted at vertices of the same degree, and which have the same degree sequence. T_1 can be produced from T_2 by a finite number of switches.

Proof: Let r_1 be the root of T_1, r_2 the root of T_2. Let $\alpha(r)$ be the list of vertices adjacent to r, listed according to degree. Let $\alpha(i,r)$ be the list of vertices at height i from r.

Let $k = |\alpha(r_1)| = |\alpha(r_2)|$. For $1 \leq i \leq k$, we do the following. Let v be the i'th member of $\alpha(r_1)$, and let d = deg(v). If the i'th member of $\alpha(r_2)$ has degree d, then do nothing this iteration; otherwise, select some vertex, z, of degree d, at height no less than the height of v, and if at the same height, after the i'th position in $\alpha(r_2)$. Switch the edges (v,r_2) and the edge from z to its father.

At the end of the iteration, the degrees of the vertices in $\alpha(r_1)$ and $\alpha(r_2)$ agree in the first i positions.

The next step proceeds simply by replacing $\alpha(2,r_2)$ for $\alpha(r_2)$ and $\alpha(2,r_1)$ for $\alpha(r_1)$. Thus in turn the vertices at distance i are modified so that the j'th member of $\alpha(i,r_1)$ has the same degree as the j'th member of $\alpha(i,r_2)$. Upon completion, the tree T_2 will have been modified into T_1 by a finite number of switches. It is crucial to note that the vertex of degree d does exist, since the degree sequences are the same. □

This can be extended to show that switching is complete for the property of "being a tree". To produce a finite set of switches for two arbitrary free trees t_1, t_2 with the same degree sequence, one simply chooses any vertex v_1 from t_1, and then selects a vertex v_2 from t_2 such that $deg(v_1) = deg(v_2)$. One roots t_1 at v_1, and t_2 at v_2 and performs the algorithm above to produce t_1 from t_2.

The extension of this result to 1-connectivity remains an open question. The extension to k-connectivity is also open. We have verified by hand that any minimal block on fewer than eleven vertices (as listed in [Hobbs73]) produces all blocks with the same degree sequence by 2-connectivity constrained switching.

Other properties for which the question of completeness remains open are:

(i) planarity

(ii) hamiltonicity

(iii) pancyclicity

(iv) strong regularity

(v) k-colourability ($k \geq 3$).

These are of interest to both the graph theorist and the theoretician in algorithmic complexity. Although there is some evidence that k-connectivity and planarity are complete, there is scant evidence for the others. In the case of k-colourability, in fact, it seems likely that it is false, since there is a counterexample for 2-colourability.

An inquiry into these questions is desirable, since in many cases the closure of a small set under constrained switching is feasible, whereas a brute force backtrack approach will prove intractable. This inquiry is hampered for two reasons. Firstly, Ryser's proof, although constructive, makes no statements about graph properties. Therefore, unless the property depends on degree sequence alone, the switchings produced may <u>not</u> be legitimate under constrained switching. This, unfortunately, does <u>not</u> show that no set of constrained switchings exists, but rather

that this set is not legitimately constrained; there may be a set which is.

Secondly, although experimental verification can be done, one must for each property give a constructive method of producing a set of legitimate constrained switches from one arbitrary graph to another, as was done with trees.

It is our sentiment that if further work is done, algorithms based on completeness will be widely used. We do not mean to suggest that there is no use for switching methods at present. One feasible technique is to generate the closure of a single graph with the property required. If, by enumeration of the class, this closure set is known to be an exhaustive listing, terminate successfully. Otherwise, find another graph with the desired property which is not in the closure set, and generate its closure. Take the union of the two sets and retest. Eventually, the algorithm must terminate in success. This type of algorithm has been employed with success in dealing with block designs ([Gibbons76], [Mathon77b]). If the first graph does not generate the exhaustive list, we are assured that the graph property is not complete.

Chapter 5: Graph generation

In this chapter we consider one problem specifically and in detail. The problem is a request for a list of all nonisomorphic graphs on n vertices. Some generalisations of tree methods will be shown. It was intimated in the fourth chapter that switching appears as the basis of an inefficient exhaustive method. The impact of this statement will be fully realised upon the introduction of alternate techniques.

This chapter shall be concerned also with the relation between generation and enumeration. This close correspondence was noted *a posteriori* with trees in some sense; the correspondence was noted after the generation method was discovered. Here, we shall employ enumeration results *a priori* in producing generation methods.

The results of this examination shed new light on the problem of graph generation; nevertheless, the improvement over existing algorithms is not asymptotically significant.

5.1 Enumeration

Perhaps the most fundamental question in graph theory is: how many graphs are there? It is conceivable that the second question might be: what do they all look like? In light of the answer to the first question, only those with great audacity and a computer persist in asking the second.

Considering the difficulty of the second question leads one to ask how we can determine the number of graphs without producing a list. The techniques involved have been called "counting without counting"; a less confusing term might be "enumerating without generating". Two primary techniques employed in graph enumeration are the method of generating functions, and the Polya theory. Both approaches are employed in a fundamentally nonconstructive manner. The generating function approach often produces a constructive counterpart. This relation is immediate in the case of binary trees. The recurrence observation for the Catalan numbers was used in producing an algorithm for generation.

The Polya theory, on the other hand, has not been adapted to successful use as the foundation of generative approaches. In preparation for possible attacks on the problem via the Polya theory, we present a brief solution to the first question; for the details, the reader is

advised to see [Harary73].

The vertex set V of the graphs to be enumerated has cardinality n. The domain in Polya's theorem is defined to be the set of 2-subsets of V. The range, or set of colours, is {0,1}. The interpretation of the mappings as graphs is as follows: if (x,y) is coloured 1 under the map, then (x,y) is an edge; if 0, then it is a nonedge.

It remains to specify a group acting on the domain. We first consider a trivial application. If every vertex is distinguishable, we will produce labelled graphs. The group on V in this case is the identity group; so, therefore, is the group on the domain D. The cycle index of this group is x_1**C(n,2). Substituting |R| = 2 for x_1, we are informed that there are 2**C(n,2) labelled graphs on n vertices. This concurs with the simpler observation that there are 2**C(n,2) nonidentical {0,1} symmetric, zero diagonal n by n matrices.

The case of unlabelled graphs is equally straightforward. The group acting on V is the symmetric group of order n, since each vertex is indistinguishable from all others. The group on V induces a group acting on D, which Harary and Palmer call the pair group. The symmetric pair groups can be computed in a reasonable

length of time; tables exist for n ≤ 10, and formulae are known for larger n ([Harary73]). Upon substitution of |R| for each x(i), the number of graphs is found; if, instead, the symbolic polynomial (1+x)**i is substituted for x(i), the resulting polynomial g(x) can be interpreted in the following way: the number of graphs with n vertices and m edges is the coefficient of x**m in g(x). The symbolic polynomial (1+x)**i is obtained by assigning weights to the elements of the range. Nonedges are given weight 1; edges weight x. Thus a graph with m edges is counted in the coefficient of x**m.

We next trace the history of direct approaches to the graph generation problem; the Polya theory will then be consided as an approach to isomorph rejection.

5.2 The classical algorithm

In combinatorial computing, generally the initial algorithm investigated is brute force and employs little knowledge of the theoretical foundations for the problem. Often the algorithms are considered to be inefficient; nevertheless, there are many NP-complete combinatorial problems ([Cook71], [Karp72]) which one suspects are not solvable by good (polynomial) algorithms. One generally then adopts heuristic approaches. In this context, we

intend by the term heuristic that the methods avail themselves of some knowledge concerning the problem to reduce the size, if not the asymptotic complexity of the problem.

Read has called these brute force solution methods the BFI (Brute Force and Ignorance) algorithms. He has graphically described them in the following quote: "These are algorithms, devoid of any subtlety whatever, which simply keep thumping the problem on the back until it disgorges an answer" ([Read70]).

The classical, brute force method remains asymptotically the best known method of solution. Graph generation falls into this class of problems; although powerful heuristic methods are known, no algorithm to date is asymptotically better than the brute force one.

5.2.1 The algorithm

The classical algorithm is simply to generate all nonidentical adjacency matrices (labelled graphs) and then perform an isomorph elimination step, retaining only a canonical example from each isomorphism class. The coding for this elimination step is an expensive task; no known algorithm exhibits a worst case of better than exponential time complexity ([Read76a], [Overton75], [Proskurowski74]).

Moreover, the output list of exponential length must be sorted prior to elimination, or a search must be performed for each graph in the list. This further computation will involve significant overhead in the implementation. The full importance of this ordering (search) step is realised when one considers that the lists are of such a magnitude that auxiliary computer storage devices must be used. On the largest machines available today, for a moderate number of vertices (say 10 or 11), the sort or search with these output lists will be the dominant cost.

5.2.2 Heuristic Improvements

An inordinate amount of work is expended in the performance of isomorphic computations with the classical algorithm. One simple restatement of the method reduces duplication substantially. The algorithm is restated as an inductive algorithm. Every graph with q+1 edges can be produced from a canonical graph with q edges by a process of changing some nonedge to an edge. This operation, augmentation, can be applied in general in many ways to a given graph on q edges. The resulting graphs may or may not be canonical, and may or may not be duplicates of graphs already generated.

The inductive implementation is then:

```
Let L(0) = {the void graph on n vertices}.
for v := 1 until C(n,2) do
    set L(v) = ø.
    for each G in L(v-1) do
        for each nonedge (x,y) of G do
            set H = G.
            add the edge (x,y) to H.
            add H to L(v).
    perform isomorph elimination on L(v).
```

In some sense, this implementation is performing isomorph rejection rather than elimination, since it is discarding isomorphs during the computation. This the initial brute force algorithm did not do. Nevertheless, one correctly views this method as C(n,2)-1 invocations of an algorithm employing isomorph elimination. Isomorph rejection entails the noncreation of duplicates.

One need only create the lists on at most $\lceil C(n,2)/2 \rceil$ vertices. The remainder can be obtained by complementing lists; the graphs in the list indexed by C(n,2)-i are obtained by complementing the graphs in the list indexed by i, for $0 \leq i \leq C(n,2)$.

The classical method is the appropriate generalisation of the classical method for trees. The edges added in the

tree algorithm required the addition of a vertex as well; here that is not the case. The parallel excepting this one fact is accurate, as one notes by examining the structure of the algorithms. The analysis of the complexity does not 'carry over', however. We have generalised a tractable method for trees to an intractable method for graphs.

5.2.3 Heap's method

Historically, the next development was to discard temporarily the classical algorithm and employ another method which is also brute force, but, according to [Heap72], is more efficient for practical cases. The fundamental observation in the algorithm is that each graph with n vertices and m edges can be produced from some graph with n-1 vertices and m-j edges by the addition of a vertex of degree j adjacent to some j of the original vertices. Given a list of all graphs on n-1 vertices, one produces all graphs on n vertices by the following algorithm. The list LAST(i) is the list of all graphs with n-1 vertices and i edges.

```
for e := 1 until C(n,2) do
    set L(e) = ø.
    for j := 0 until min(e,n-1) do
        for each graph G in LAST(e-j) do
            for each selection of j vertices from G do
```

```
                    let H = G.
                    connect a new vertex to this selection
                    of j vertices in H.
                    add H to L(e).
            perform an isomorph elimination on L(e).
```

The algorithm employs no intelligent heuristic, but sufficed to generate the eight vertex graphs ([Heap72]).

5.2.4 An improvement on Heap's method

Heap, in his pioneering paper, overlooked an important method of improving his algorithm. Consider, for a moment, a sequence of graphs G(1), G(2), ..., G(n)=G in which G(i+1) can be produced from G(i) by Heap's method. In general, there will be many such sequences of graphs producing G which are valid. At least one of these sequences must satisfy the property that for each production of G(i) from G(i-1), a vertex is added in the production of G(i) which has minimal degree in G(i). We know, then, that it suffices to examine only production sequences in which the vertex added has minimal degree in the result. We therefore modify Heap's algorithm to select only graphs from LAST(e-j) whose minimum degree is at least j-1. We further require that the added vertex of degree j be connected to all vertices of degree j-1 in the original

graph.

The modified version can be no worse than the original since it performs no computation that the original did not; moreover, it does not perform much of the computation from the original, and constitutes therefore an improvement. Asymptotically, the ratio of the number of duplicates produced by the modified version to that produced by Heap's method tends to zero.

5.2.5 Practical improvements

For the sake of completeness we include the algorithm of [Baker74] employed in the generation of the nine vertex graphs. This implementation employed the classical algorithm. Baker, Dewdney, and Szilard introduced two practical improvements which made the production of a catalogue possible.

Both techniques are of minor importance here, since they are intended to be employed only in the case of the 9 vertex graphs. The first was the use of scatter storage techniques to search the list in the elimination phase. The second technique was the implementation of special purpose coding algorithms.

5.3 Read's algorithm

The first major breakthrough in the generation problem occurred with the investigation of a new class of algorithms, the orderly algorithms, in [Read76b]. The structure of the classical algorithm involves an augmenting operation which produces graphs in L(i+1) from those in L(i), and a canonicity definition which enables one to discard all but the canonical representative for an isomorphism class. The implementation requires a coding routine to place graphs in canonical form, and a routine to search the output list to determine if this is the first production of the graph. In the event that the graph is found on the output list, one copy must be discarded. This graph is called a loser.

5.3.1 The general method

Read's method adds one ingredient to those required by the classical algorithm. A list order is imposed. This order is a total order on canonical forms. The algorithm is defined to operate on vectors with a given definition of an augmenting operation, a canonicity definition, and a list order; the form of the algorithm is as follows:

1. Let L(i) be a list of canonical vectors which is in list order. Let L(i+1) be initially empty.

2. For each vector in L(i) in turn, apply the augmenting operation to produce a set of vectors which

are candidates for inclusion in the list L(i+1). Each vector produced is added to L(i+1) iff it is in canonical form and its addition to L(i+1) would leave the list in order.

There exist definitions of augmenting and canonicity for which this algorithm will produce incomplete lists. Constraints must be stated to ensure that the algorithms operate as desired. Read has stated three conditions and proved their sufficiency for correct operation of the method; they are:

(1) Each canonical vector in L(i+1) can be produced from some canonical vector in L(i).

(2) An operation f is defined on canonical vectors. The function f applied to a canonical vector of L(i+1) has as result the first graph in L(i) from which it can be produced by the augmenting operation. The function f must be weakly monotonic (if x precedes y on L(i+1), then f(x) does not follow f(y) in L(i)).

(3) When the augmenting operation is applied to a single vector in L(i), the many vectors produced are given in list order.

If the conditions are satisfied, the search through the output list is unnecessary.

5.3.2 The application to graphs

The application of orderly algorithms to the generation of graphs requires the existence of suitable definitions of augmenting, canonicity, and list order; furthermore, one is required to supply a vector representation of graphs. Consider a graph G. We will produce a vector with C(n,2) elements which represents the graph. One simply lists row by row, commencing at the top, the rows of the upper triangle of the adjacency matrix of G. In other words, one lists in order the elements above the principal diagonal. This vector formulation can be generalised. One can list all the elements in the upper triangle in any order, with the stated provision that the order selected is used for all graphs. The row by row formulation is called the standard vector form.

Our canonicity definition is that the lexicographically largest vector in a given isomorphism class is canonical. The list order chosen is the vector relation 'greater than'. The augmenting operation on a vector v produces many vectors which are the result of changing some 0 to a 1 in the vector. This change is performed from left to right, starting at the 0 which immediately succeeds the rightmost 1 in the vector. This last stipulation in the definition is a major improvement over the augmenting operation in the classical algorithm.

It can be enforced since if a zero preceding the rightmost 1 is changed, the result cannot satisfy the list order requirement for L(i+1). These definitions satisfy the three conditions ([Read76b]).

The effectiveness of Read's algorithm in a practical sense is clear; the search through the output lists is removed, and fewer graphs are produced. One must be careful to avoid unreasonable conjectures here. Placing a vector representation of a graph in canonical form is, in general, NP-hard ([Hirschberg73]). Verification of canonicity is not known to be easier than this.

5.3.3 Heap's method revisited

One wishes to improve other methods by the intelligent use of similar observations. Heap's method is fundamentally different from Read's in one important sense -- it is a vertex addition method rather than an edge addition method. In Heap's method, one adds a vertex to create an n vertex graph. We assume that in an implementation, the input graph will have its vertices labelled with the integers {1,2, ..., n-1}. The added vertex will be labelled n.

We will attempt to revise Heap's method so that, like orderly methods, it must simply check a graph produced for

canonicity. Consider the vector form of a graph obtained by listing the entries of the upper triangle of the adjacency matrix of the graph column by column starting at the left. We call this the transpose standard form. Consider a graph G which is canonical in transpose standard form. The deletion of the vertex labelled n and its incident edges will produce an n-1 vertex graph G'; we contend that the labelling of G' induced by its labelling in G is canonical in transpose standard form. Assume to the contrary that this labelling is not canonical; let the transpose standard code of G' be c. The vertex labelled n in G can be placed adjacent to the vertices of G' to which it was adjacent in G, but which have subsequently been renumbered in placing G' in canonical form. The labelling of G induced by this process produces a larger code for G than the vector form produced by the original labelling of G, since G' is not canonical and c is larger than the vector form for G' as it was originally labelled. This contradicts our assumption that G is in canonical form. Therefore, G-{n} is canonical if G is.

The implication of this observation is that all canonical graphs on n vertices can be produced from some canonical graph on n-1 vertices. We therefore modify Heap's algorithm to verify as a graph is produced that it

is (transpose standard) canonical; we write it onto the output list iff it is.

It is a depressing side effect that the suggested minimal degree modifications are no longer valid here. This is one instance where one as yet cannot have the best of both worlds. One can either reduce the number of output graphs dramatically, or reduce the time investment per graph. We have failed, however, to produce a method which benefits from both techniques.

5.4 Possible approaches via the Polya theory

The breakthrough supplied by Read's method motivates one to design an efficient algorithm for non-canonicity. The penultimate goal, however, is to avoid the production of non-canonical graphs altogether.

5.4.1 Isomorph rejection

We direct our attention to the problem which we suggest is the final aim of this work, that of never producing a non-canonical output graph. Prior to this point, the methods discussed have been essentially two step methods, one of production, and one of elimination. The methods sought for here are isomorph rejection techniques.

The only mention of efficient isomorph rejection in

the literature appears in enumeration. Although some formulations of the Polya theoretic enumeration of graphs do not employ true isomorph rejection, the enumeration can be stated in such a way that rejection, rather than elimination, is being performed. We consider a formulation of the Polya theoretic enumeration of graphs as a starting point for the examination of isomorph rejection in graph generation.

The enumeration of graphs on n vertices can be phrased as follows. Let D be the set of unordered pairs of elements from the set {1,2,...,n}. The range R is just the set {0,1}. The group A is the pair group of the symmetric group of order n. The isomorph rejection is implicitly performed by the actions of the group A on D.

What is the group A with reference to D? We let the elements of D be all coloured (mapped to) 0. The group A is the automorphism group of the result. A constructive implementation of the Polya theoretic formulation exists. One initially flags all possible {0,1} vectors of length C(n,2) 'unseen'. One repeatedly selects some unseen graph, G, and outputs it. Each permutation of A is applied to G, and the resulting labelled graph is marked seen. The actions of A are employed to "reject" all possible isomorphs of a graph when it is output. This process of

selection, output, and "rejection" is repeated until all graphs are marked seen. Upon completion of this process, one graph from each isomorphism class will appear on the output list. This procedure can be easily modified so that the output graphs satisfy a given canonicity definition. The major drawback of this method is, surprisingly, not one of time. It is, instead, one of storage. The entire group A is stored, as are 2**C(n,2) flags. Some minor improvements can be effected (for example, storing just the generators of the group); nevertheless the practical use of this algorithm is small.

One might reasonably hope for a method which enjoys the best of both worlds, a refinement of Read's algorithm employing isomorph rejection.

5.4.2 Automorphism partitions

In the algorithm from the Polya theory, isomorph "rejection" is performed by remembering the actions of an automorphism group. One might question our use of "rejection", since the method is so inefficient; nevertheless, it never produces a graph which must be tested for inclusion in the output list.

In the quest for an efficient algorithm, we must necessarily avoid use of such extravagant quantities of

storage. One solution to this dilemma is to perform isomorph rejection piecewise. The following technique appears to be a reasonable approximation to the original method; this modification is storage efficient, however. One computes the automorphism partition of each input code, by any standard method (for example, see [Corneil72], [Mathon77a], and [Dymond76]). One then performs the augmenting operation iff for a given zero in the code, it is leftmost in its similarity class as well as satisfying the conditions in Read's method. The justification for this rejection technique is that if two augmentations are done in similar positions, the results will be isomorphic.

One is led to inquire whether, for general vector forms of graphs, this rejection is sufficient to bypass elimination of duplicates. The answer to this is negative, as shown by the following graph. The nonedges e_1 and e_2 are not similar, but the effect of adding e_1 is isomorphic to the effect of adding e_2:

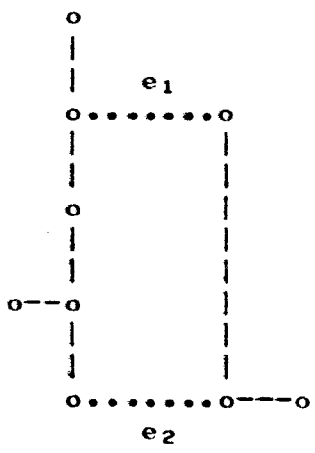


Figure 11

We therefore restrict our attention to the standard vector form for graphs. Once again one inquires whether the rejection is sufficient with the standard vector form. It is unfortunate that the answer is again negative. The duplicates arise precisely from not remembering enough information to perform the required rejection.

Say x, z are indices of two zero positions in the same similarity class in G, and further say y is the index of a zero in a different similarity class such that $x<y<z$. Suppose that an augmenting operation is performed in position y to produce a new graph H. In H, the nonedges indexed by x and z are not necessarily similar. This would allow the selection of position z as a candidate for an augmenting position. The result of augmenting in position z would be isomorphic to the result of augmenting in G in position x and then in a position similar to y.

5.4.3 Efficient isomorph rejection

One must recall some information from previous automorphism partition computations. This can be done as follows. One selects a position for augmentation as before. A rejection vector is output as well which has a 1 in those positions similar to a zero with smaller index in the vector than the current augmenting index. This vector is output with each graph. Having selected an augmenting position, one employs it to augment iff the rejection vector does not have a 1 in the corresponding position.

5.4.4 Guidelines for further work

The extent of the isomorph rejection is massive: is it sufficient? The answer is once again negative. Nevertheless, the duplicates produced by this method have been found experimentally to violate simple canonicity requirements. The non-canonicity algorithm of 5.5, which is of polynomial time complexity, suffices to eliminate all known duplicates produced by this algorithm. This observation is significant since it implies that a polynomial elimination method may suffice to replace the (currently used) exponential routines. One duplicate produced by the method is:

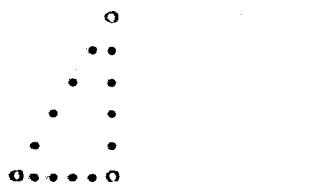


Figure 12

where two numberings produced by the algorithm are:

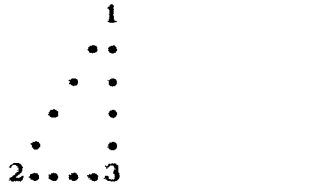


Figure 13a

and

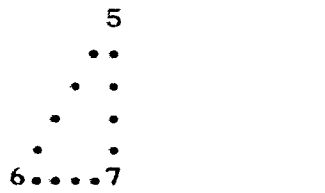


Figure 13b

The algorithm may produce a duplicate which fails the non-canonicity test. We know of no such graph. The polynomial elimination method can be replaced by a degree constraining algorithm as was used for rooted trees; such an algorithm would preprocess the graph.

Further research is required to determine whether duplicates are produced by this algorithm. If no duplicates are produced, the method is the first which employs isomorph rejection solely in the generation of graphs. In practice, computing automorphism partitions is approximately as difficult as verifying canonicity. The number of duplicates requiring elimination is drastically

reduced. The net result is an improved method, whether or not elimination is required.

5.4.5 Isomorph rejection and Heap's method

We turn next to Heap's method to examine the feasibility of isomorph rejection. The application of isomorph rejection in vertex addition methods such as Heap's has been considered by McKay ([McKay77]). He modifies Heap's method to produce graphs with maximum degree d. This is done by adding a vertex of degree d in all possible ways to (n-1)-vertex graphs such that the largest degree in the result is d.

Isomorph rejection is performed by finding the automorphism partition of the (n-1)-vertex graph, and only performing vertex addition in nonsimilar ways. McKay shows that this does not suffice to prevent production of duplicates. He therefore computes the automorphism partition of each output graph as well, and writes it on the output list iff the added vertex is in the canonically first automorphism class. In this way, no duplicates are produced.

The elimination step is remarkably like the verification of canonicity in Read's method. Once again, the rejection is powerful, but does not make the

elimination unnecessary.

5.5 Notes concerning implementation

The routines required are standard, available programs, except for some heuristic improvements to quickly verify non-canonicity "most of the time". We once again restrict ourselves to standard canonical forms, and examine rules which, in polynomial time, will test a code for canonicity, and will usually succeed in showing the code to be non-canonical if it is.

The first test proposed is simply to ensure that the vertex labelled 1 has maximal degree k. If, to the contrary, vertex 1 has degree $l < k$, then there is a larger code prefixed by k 1's (which is larger than the current code prefixed by l 1's). Thus the code is canonical only if the vertex numbered 1 has maximal degree.

We next define the priority of a vertex i with respect to another vertex j as follows. Let p(i,j) = (2**(n-1))*a(1) + ... + (2**(n-j))*a(j) where a(k) is 1 if the edge (i,k) is present, 0 otherwise. The code is canonical only if for all $2 \leq j < v1 < v2 \leq n$, $p(v1,j) \geq p(v2,j)$. If this were not the case, then there would exist v1,v2 whose interchange would produce a larger code.

These necessary conditions are polynomial tests; they do not suffice to always show non-canonicity, but do exclude the vast majority of non-canonical output graphs.

Chapter 6: Graphs restricted to given parameters

In applications requiring catalogues, the user is often interested only in graphs with given parameter values. The network analyst may be interested only in k-connected graphs, and the chemist only in graphs with a given degree sequence. The electronics engineer may be interested in planar graphs as layouts for a circuit. In the timetabling of lectures and examinations, the university administrator will be concerned with k-colourable graphs. Even if the particular restricted class of concern is not studied here, it is likely to fall into a similar class with one of the methods introduced. In preference to presenting a complete study of restricted classes, we present a potpourri of problems with dissimilar solutions.

There is a further importance to this investigation. We earlier observed that the generalisation of techniques for restricted classes furnish valuable clues to more general methods. There is a dual observation to this: often a general method can be specialised to the problem at

hand.

The generation of related combinatorial structures will be considered. A few of the examples mentioned will be digraphs, multigraphs, set systems, hypergraphs, set packings, and block designs. Read's method is shown to be sufficiently general to deal with most of these configurations. Each application of Read's method can be improved via isomorph rejection.

6.1 Locally restricted graphs

In preparation for a presentation of methods for locally restricted graphs, we make note of an obvious 'generalisation' of these techniques to the problem of listing all n vertex graphs. The production of possible degree sequences of graphs with n vertices and m edges is performed by listing all partitions of the integer 2m into n parts, with no part larger than n-1. We may require that the components of each partition are listed in nonascending order. An algorithm for this appears in [Riha76]. Each partition thus generated is tested by the Havel-Hakimi algorithm. If it is not graphical, it is discarded from the list. Each remaining partition has some realisation as a graph. An algorithm to generate locally restricted graphs is then invoked with each remaining partition as the

degree sequence. In this way, all graphs with n vertices will be generated.

6.1.1 Farrell's method

The first method known to us which attempted to solve the problem of generation of locally restricted graphs appeared in [Farrell71]. The method is in some ways similar to the revised version of Heap's method. The main difference is that, at each step, rather than adding vertices from a range of degrees, a vertex of specified degree is added. This is implemented as follows. Let d be the degree of the vertex numbered 1. For each d-subcomposition of the remaining vertex partition in turn, connect vertex 1 to the d vertices corresponding to the lowest numbered vertices in the classes of the partition represented in the subcomposition. The remainder of the graph is constructed recursively. This selection of subcompositions operates with a partitioning imposed on the vertices. Two vertices are in the same class iff they have the same degree and are connected to the same set of vertices already placed in the graph.

6.1.2 Farrell's method -- an example

Farrell's algorithm is more complex than earlier methods. We therefore supply illustrative example.

Consider the degree sequence (3,3,2,2,1,1). This sequence is graphical. At the first step, find the 3-subcompositions of (3,2,2,1,1). They are:

(3,2,2)

(3,2,1)

(3,1,1)

(2,2,1)

(2,1,1)

Each of these will be selected in turn as candidate adjacencies for the first vertex. We consider here the case when the first subcomposition (3,2,2) is selected. The remaining sequence after adding these three edges will be (2,1',1',1,1). The vertices marked 1' are distinct from those marked 1 since they are connected to the vertex already added. We now find the 2-subcompositions of (1',1',1,1). They are:

(1',1')

(1',1)

(1,1)

Selecting each of these in turn, we construct three graphs with labels (2,1',1',1,1). They are:

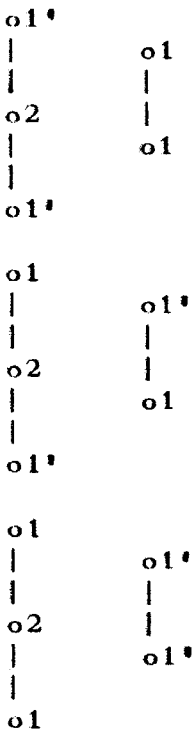


Figure 14

The addition of the vertex of degree 3 in the appropriate way produces 3 of the graphs with the sequence (3,3,2,2,1,1). The vertex yet to be added will be added to the graphs drawn so that its adjacencies will be the vertices labelled 2, 1', and 1'. When each subcomposition of the original sequence is processed in this way, all graphs on that degree sequence are produced.

6.1.3 Farrell's method -- improvements

We consider a canonical form for graphs which is

particularly suited to the generation of locally restricted graphs. Define a numbering of the vertices to be canonical iff the induced vector form is maximal subject to the constraint that vertices of higher degree are numbered before vertices of lower degree.

Examination of Farrell's algorithm shows that at least one production of a graph gives the graph in this canonical form. This theorem appears as a corollary to a more general result presented in the next section.

We can therefore modify the algorithm as follows. Verify as a graph is generated that it is canonical; write it on the output list iff it is. We again bypass the searches in output lists. We can, moreover, modify the method to avoid the production of clearly non-canonical graphs. We define a <u>signal</u> of non-canonicity as an instance of two vertices x, y with the same degree where x is numbered before y, but in the leftmost position of a vertex to which one of x, y is not adjacent, then y is adjacent, and x non-adjacent. In this event, if y were numbered before x, a larger code would result. We therefore ensure in the operation of the algorithm that when a signal is encountered, this branch of the computation is discarded.

Read's observations have been appropriately generalised to handle a new class of problems; verifying canonicity appears to be a general concept indeed.

6.1.4 A new method

Farrell's method has one important difference from the direct methods. The difference is that input lists are not supplied; all graphs are created from scratch. One important observation is that after i vertex additions, the partial graph created will, in succeeding vertex additions, spawn many graphs. Observe that if the partial graph created is not in canonical form, all graphs derived from it will fail to be canonical. One should therefore verify the canonicity of partial results as well.

A modification of Farrell's method to deal with this additional observation has departed to such a great extent from the original, that it can rightfully be considered a new method.

The verification of partial canonicity has been employed in other generation problems, for example the generation of the six vertex digraphs ([Read77]).

We have shown that the standard canonical form, appropriately modified, allowed the implementation of an

orderly version of Farrell's method. We will generalise this theorem to a family of vector forms. Define a partial order on the set of all 2-subsets of {1,2, ..., n}, satisfying:

(1) (i,j) precedes (i,k) if j < k, and

(2) (i,k) precedes (j,k) if i < j.

We restrict ourselves to vector forms in which the entries of the upper triangle of the adjacency matrix are listed in the canonical form in an order which violates neither rule for the partial order. This relaxes the earlier restriction to specific vector forms. It includes both the standard and transpose standard form. Having selected an ordering, we define canonicity as maximality within isomorphism class subject to the degree conditions; to avoid ambiguity, we refer to vectors in this canonical form as b-canonical.

Theorem: Each b-canonical graph with degree sequence d is produced by the algorithm.

Proof:

Let G be a labelled graph with degree sequence d which the algorithm fails to produce. Then there exist i < j < k such that

(1) deg j = deg k, and

(2) In G, (i,k) is an edge but (i,j) is not, and

(3) When adding vertex i, the labels of the remaining vertices are l(i+1), ..., l(n). Then l(j) = l(k).

If no such i, j, and k exist, the graph is produced by the algorithm.

Since l(j) = l(k), they are adjacent to the same subset of {1, ..., i-1} in the partial graph created so far; they are <u>similar</u> as a result. If edge (i,j) were selected in preference to (i,k), then

(1) the partial graph created by the addition of (i,j) is isomorphic to the partial graph created by the addition of (i,k), since (i,j) and (i,k) are similar, and

(2) a larger code results since (i,j) precedes (i,k) in the partial order, and

(3) after addition of vertex i, the labelling l'(i+2), ..., l'(n) to be used in the addition of vertex i+1 is the same whether j or k is selected.

We have shown a larger code isomorphic to G, and thus G is not b-canonical. This establishes the theorem. □

One cannot anticipate a more general result, since the constraints on the partial order are required in the proof.

6.1.5 James and Riha's algorithm

An independent investigation of generation of locally restricted graphs was undertaken by James and Riha ([Riha74a]). An implementation of their method has appeared in the literature ([James76]). Their method was a rediscovery of Farrell's algorithm; thus we do not repeat the description.

6.1.6 DENDRAL

Generation of locally restricted graphs has also been considered by chemists for use in a heuristic program to aid the chemist in molecular analysis; this program is DENDRAL ([Feigenbaum71]). A veritable flood of papers concerning molecule generation has appeared relating to this application. Other molecule generation methods, besides DENDRAL, have also appeared, but they are not formulated by a graph theoretic approach (see [Lynch71]).

We present a quick index to the work on DENDRAL, and mention the design of the methods involved. The generation algorithms are discussed in [Brown74a], [Brown74b], [Carhart75], and [Sridharan73]; an excellent overview of them appears in [Smith74]. The work in [Buchanan74] is concerned with applying the results of generation methods

to a program for molecular analysis. Another paper, [Sridharan74], is a catalogue produced by the generation methods which extended earlier catalogues of cubic graphs in the literature ([Balaban66], [Balaban70a]). This catalogue has been superseded by the appearance of larger catalogues in [Bussemaker76], and [Faradzev76].

More work has appeared in the chemical literature. The work in [Sheikh70] and [Masinter74a] is concerned with generation of special subclasses of molecules. In [Masinter74b], the DENDRAL program has first been concerned with the exploitation of molecular symmetry. Other chemists have employed DENDRAL as the starting point for chemical based infromation systems ([Kudo75]).

The methods employed are extremely specialised. The fundamental structure of the algorithms derive from the tree generator of [Lederberg64]. This generator produced, via a technique requiring isomorph elimination, a list of the saturated hydrocarbons. The technique was generalised to simple cyclic graphs in [Lederberg65]. The current algorithm involves a two step generation phase. The underlying tree of the molecule is generated (in graph theoretical terms, this is the block-cutvertex tree of the molecule's underlying graph). A superatom, or block with vertices marked as having unused adjacencies, is then used

to replace a vertex of the tree. This is done in all possible ways; the result is a catalogue of graphs with given blocks and block-cutvertex tree. The generators make selections which guarantee that the results have a given degree sequence.

The methods involve many hidden assumptions which are true for molecules, but not true for graphs in general. Examples of this are restricted ring size and alternating bonds in rings.

6.2 Connected and k-connected graphs

A common misconception of the novice is to imagine that there are only connected graphs. There is some elegance in this view; in applications to networks and chemistry, the concern is clearly directed to connected graphs. Disconnected graphs can be viewed as a set of connected graphs. There is ample reason from an applications standpoint to exclude disconnected graphs from catalogues.

We therefore consider generation methods for connected graphs.

6.2.1 Algorithms based on the classical method

The generation of all connected graphs can be achieved

by applying the classical algorithm with as initial list the list L(n-1) of the trees on n vertices. This initial list is produced by the tree methods. The verification that all connected graphs are produced by this method can be seen by observing that every connected graph can be obtained by a finite number of augmentations from a spanning tree of the graph.

6.2.2 Algorithms based on Read's method

The simplicity of the extension of the classical method does not carry over to Read's method. Consider for example the graph:

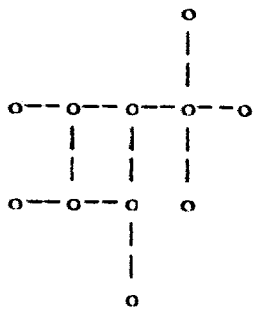


Figure 15

There is no (standard canonical) augmentation which is legitimate by Read's constraints which produces this graph from a tree.

6.2.3 Edge deletion

The inductive algorithms introduced have all employed edge or vertex augmentation. We consider here an inductive

edge deletion method.

Each connected graph can be produced from some connected graph with one more edge by the deletion of an edge. We can therefore start with the complete graph, and inductively delete edges to produce graphs. As a graph is produced, one checks that it is connected, and writes it on the output list iff it is.

Read's method can be employed as well. Instead of augmentation, we use deletion constrained so that only 1's to the left of the leftmost 0 in the code are changed to 0's. One again tests each output graph for connectedness, as before.

6.2.4 Algorithms based on locally restricted graphs

One simple modification of the locally restricted generators is tc first generate only sequences which have connected realisations (see [Hakimi63]), and then generate graphs with those sequences, discarding those which are disconnected. The method of James and Riha provides a component counting facility; therefore, the number of components in an output graph is known as it is produced, and very little extra labour is involved in removing disconnected results. The relative improvement is not as substantial as with the classical method.

6.2.5 k-connected graphs

No specialisation of the classical method or Read's method is known which generates only k-connected graphs. The classical algorithm can be used, given a list of minimally k-connected graphs. Although some small catalogues of minimally k-connected graphs are available (for example, [Hobbs73]), general techniques to produce these catalogues are not known. The edge deletion method is appropriate, however; every connectivity test is replaced by a k-connectivity test.

The locally restricted generators can be used to operate with degree sequences in which the minimal degree is k. Infeasible degree sequences will be rejected; on the other hand, graphs which are not k-connected will be produced from degree sequences which do have some k-connected realisation.

In an application to physical chemistry, the case of 2-connectivity has been considered. Many essentially brute force methods based on the classical algorithm have been discussed in the literature ([Sykes66], [Heap66], [Domb67]).

An improvement on these, which appeared in

[Proskurowski73], deserves a more detailed introduction. In addition to the usual operation of edge augmentation, Proskurowski introduces an operation called <u>vertex splitting.</u> Splitting a vertex v entails dividing the adjacencies of v into two nonempty classes a_1 and a_2. One replaces v with two adjacent vertices, v_1 and v_2; v_1 is then made adjacent to the vertices of a_1, and similarly v_2 with a_2. Proskurowski proves that any 2-connected graph can be produced by vertex splitting and edge augmentation from K_3. The more interesting result is that <u>all</u> graphs produced are 2-connected; unfortunately, an isomorph elimination step is still required.

Tutte has proposed a similar inductive method for 3-connected graphs which produces no graphs which are not 3-connected ([Tutte61]). He employs edge augmentation and vertex splitting. The vertex splitting operation is here constrained so that only vertices of degree at least four are split, and the two groups a_1 and a_2 both contain at least two vertices. The only 3-connected graphs not produced by these operations are <u>wheels;</u> these can be easily added to complete the lists.

In a discussion of Tutte's algorithm, Weinberg suggests that one may employ automorphism partition information to avoid the production of duplicates; he does

not propose a method for doing this, however ([Weinberg71]).

We recall for a moment the analysis in 4.6, in which we observed that completeness for k-connectivity is important. If k-connectivity is complete, one can generate k-connected graphs without generating lists of graphs which are not k-connected. This would offer a clear improvement over existing techniques.

6.3 Planar graphs

The structure of planar graphs was investigated by considering generation schema for their production ([Barnette73], [Barnette74], and references therein). We are concerned with these generation methods not in the context originally set forth, but rather to exhibit a class of methods for generation, and to introduce some topologic considerations into this study.

The topology of the generation method is fundamental: one deals with planar maps instead of the more abstract definition of graphs. The work is concerned with 3-connected graphs; as a result, the interpretation of a graph as a unique map is well defined ([Whitney33]). Barnette introduces three operations on planar maps. The three operations are splitting a face, subdividing a

hexagon, and double face splitting. We state here only the main result. Barnette shows that these operations suffice to generate the planar 4-connected graphs.

Our concern with this method is not with the practical details or the operations; rather we consider the format of the method. Many similarities exist with the method of constrained switching. The algorithm here employs an operation set which is constrained to the structures of interest.

6.4 Graphs with a given subgraph

When first considering a method for generation of graphs with given subgraphs, one does not fully comprehend the generality of such a method; this is usually due to one's predilection to fix the sort of subgraph of interest first. We examine instead a fully general method in which one states the subgraph of interest as an input parameter.

The method is a constrained version of Read's method. In the description of Read's method given earlier, we adopted a standard vector form for graphs. There are many different forms which we did not introduce at that point: we shall now introduce a comprehensive definition.

Let $EP(n) = \{(i,j) \mid 1 \leq i < j \leq n\}$ be the set of possible

edges of a graph whose vertex set is {1,2, ...,n}. A vector order is an arrangement VO of the set EP. A vector representation or form of a graph G is a binary vector, in which the i'th component is 1 iff the edge indexed by i in VO is present in G. There are C(n,2)! vector orders.

Read's method operates independently of the vector order chosen; we may therefore select an arbitrary vector order and apply Read's method. There arises no substantial problem in the canonicity definition: each vector which is maximal in its isomorphism class is canonical. The canonical labellings of the graph will differ for different vector orders. This will cause no difficulties as long as a given vector order is understood throughout.

These observations will be used to implement our general approach. We begin by examining an example. The subgraph which we choose is a Hamilton path. We consider a subset of the possible vector orders, in which every possible edge of the form (i,i+1) is listed before all possible edges not of this form. This vector order was introduced in [Hirschberg73]. The number of possible vector orders remains large, (n-1)!(C(n,2)-n+1)!. One supplies as initial list just the vector consisting of n-1 initial 1's, followed by all 0's. This vector is a Hamilton path in canonical form in the vector order. The

application of Read's method to this vector will produce all graphs with a Hamilton path. The fundamental observation is that no graph will be left out due to problems with list ordering, as would be the case with the standard vector order.

In spite of this important specialisation of Read's method, it is desirable to exhibit a specialisation which avoids the necessity of choosing appropriate vector orders for the problem at hand. We now propose a specialisation which operates in the correct manner independent of the subgraph selected.

We are given a subgraph S which must appear in each graph generated. Assume that S has n vertices, since if it has fewer, we can add isolated vertices to make up n. First list the edges of S, e(1),...,e(q), in any order. Define a vector order O(S) of EP induced by S in which the i'th element in O(S) is the possible edge e(i) for $1 \leq i \leq q$. The remaining elements of EP are ordered in any convenient fashion. Input as initial vector the vector prefixed by q 1's followed by 0's. The invocation of Read's method using this initial graph (which is, after all, just S) and the vector order O(S) will produce all graphs with subgraph S.

The method can be applied directly to the generation

of Hamiltonian and pancyclic graphs, graphs with a clique of given size, or graphs with a specified spanning tree.

One further problem included in this general class is to produce all graphs without a given subgraph. Consider the method for generating all graphs with a subgraph S. We again introduce the order O(S). Input an initial vector of all 0's. We apply Read's algorithm with one proviso: if the graph produced has q initial 1's, discard it without verifying canonicity. The justification for this is plain enough: a graph has S as a subgraph iff its representation in O(S) commences with q initial 1's.

One desirable extension would be to show how to generate graphs without any subgraph from a set of graphs, or without a homeomorph of a given graph as a subgraph. This appears to be a harder problem.

6.5 k-colourable graphs

We consider an example of a problem for which no efficient specialisation is known. A few aspects of the generation of k-colourable graphs are instructive. No degree sequence conditions are known to exist. A relation with the subgraph methods is infeasible; those methods are concerned with graphs with and without a given single subgraph, not those without a given set of subgraphs (or,

more precisely, homeomorphs of a given graph). The classical method and Read's method of graph generation do not at first appear to be relevant.

A trivial theorem concerning colourability can be employed to our benefit here. If a graph, G, is not k-colourable, no augmentation of G can be k-colourable. An inefficient method, but nevertheless the best available, can then be stated. One generates graphs in the usual way; as each graph is produced, one verifies that it is k-colourable prior to its acceptance.

The major stumbling block with this method is that the verification of k-colourability is NP-complete ([Cook71], [Karp72]). This will consume much time beyond the elimination to be performed. This test can be performed efficiently when k=2; the class generated will then be the bipartite graphs.

6.6 Other problems

We have just scratched the surface. The examples were selected as representative of problems in the area. A few additional problems are mentioned here, with appropriate notes for further study.

One generation problem is of special concern to an

open problem in graph theory, the Reconstruction Conjecture ([Bondy76]). The generation problem is the production of graphs with a given spectrum (set of eigenvalues of the adjacency matrix). The spectrum of a graph is reconstructible ([Tutte76]). If a generation procedure for graphs with given spectrum exists, one could generate relatively small sets and verify that their subgraphs are nonisomorphic, to verify that there is no counterexample with that spectrum. A useful theoretical byproduct of this would be a characterisation of which sequences are the spectrum of some graph, and which are the spectrum of exactly one (for lack of a previous definition, we term these spectral and unispectral sequences, respectively).

Another class which has recently caused much interest (and dismay) for the designers of graph isomorphism algorithms is the class of strongly regular graphs ([Corneil76]). These graphs have proven in the past to be difficult cases for known isomorphism algorithms ([Corneil72], [Mathon77a], [Dymond76]). There are existential methods for strongly regular graphs. Efficient exhaustive generation methods have eluded detection to this date; a recent survey of results appears in [Weisfeiler76].

The flavour of the work done cannot be adequately

depicted without mentioning some related graph generation methods. Some extremely specialised classes of graphs are mentioned to allow the reader to infer the scope and specialisation in the study of generation methods. One such method is employed to generate the concave vertex weighted trees, a special type of labelled tree which finds application in circuit design ([Tapia67]); another is the generation of minimal triangle graphs, graphs which when drawn in the plane form a diagram of superimposed diagrams of triangles ([Bowen67]).

One final graph generation method in the literature is an excursion into recreational mathematics. In [Teh65], Teh and Jha describe many techniques which, given one or two regular graphs, produce some larger graph. Their methods are existential only; moreover, their criterion for construction is not that the graphs be in any way useful in an applications sense. Their aim, in fact, is just the opposite: they generate graphs for their aesthetic value only.

We conclude our examination of graphs on this light note.

6.7 Other graphical structures

In this section, we mention extensions of Read's

method to digraphs and multigraphs; these extensions have been ignored until this point so as to maintain a manageable scope for this study.

6.7.1 Directed graphs I

An oriented graph is a directed graph in which the presence of an edge (x,y) implies the absence of the edge (y,x). We mention oriented graphs as an instructive example of the generation method for digraphs with a given subgraph. One simply generates all digraphs without the (unique) two vertex digraph with two arcs as a subgraph.

6.7.2 Directed graphs II

The generation of digraphs, unlike oriented graphs, has appeared in the literature ([Read76b]). The problem has attracted slight practical attention owing primarily to the astronomic growth of the number of digraphs. As an illustration of this, the generation of the six vertex digraphs, which number approximately 1.5×10^6, was recently undertaken ([Read77]). The extension of this to the seven vertex digraphs would entail the production of a list of 8.8×10^8 digraphs ([Harary73]).

The methods from the literature mentioned follow a similar pattern to the undirected counterpart. Read's

method can be employed, and again isomorph rejection via automorphisms may be used.

6.7.3 Tournaments

A tournament is an oriented complete graph. The generation of tournaments can be performed more efficiently than the generation of oriented graphs in general; this is once again due to an appropriate application of Read's method. It suffices to use the standard canonical form for graphs rather than digraphs since the adjacency matrix of a tournament is skew symmetric.

6.7.4 Posets, semilattices, and lattices

A subclass of the digraphs are posets. A poset is a pair $\langle V,R \rangle$ where V is a finite set of elements $\{v(1), \ldots, v(n)\}$, and R is a binary relation on V which is reflexive, transitive, and antisymmetric. The relation R induces a another natural relation on V, the covering or dominating relation. An element x is said to cover an element $y \neq x$ if xRy and there is no $z \neq x, y$ such that xRz and zRy. A graph is constructed from the covering relation of a poset as follows. The vertex set of the graph is V, and for $x, y \in V$, (x,y) is an arc of the arc set iff x covers y. This is just the Hasse diagram of the poset. Graphs of posets are acyclic by the anticircularity lemma ([Birkhoff67]).

Birkhoff alsc gives a proof that if there is a sequence of elements x(1), ..., x(k) such that x(i) covers x(i-1), no element from this sequence can cover x(1) except x(2). Otherwise, the covering relation is violated.

Further, if neither of these conditions is violated for some digraph D, D is the graph of some poset. If a digraph D violates either of these conditions, any graph obtainable by augmenting D must also violate the condition. We can therefore employ a method similar to that used for generation of graphs with a given subgraph. One applies the digraph generation method with the following modification. On input of a digraph, one finds, for each vertex v, the set R(v) of vertices reachable from v. Then for each v, and for each x $\in$ R(v), one disallows the addition of the arc (x,v) and also the arc (v,x). The first restraint ensures that the acyclic character of the graph is maintained. The second ensures that the covering relation is preserved. One thus performs only "allowable" augmentations, thereby producing a list of nonisomorphic posets.

The generation of posets is of interest since effective enumeration formulae are not known ([Jackson77]). Semilattices and lattices are subsets of the class of posets, and lie in a fascinating correspondence with this

class. A semilattice is a poset with a greatest lower bound. A lattice is a semilattice with a least upper bound.

The relation among lattices, semilattices, and posets is also described in [Birkhoff67]. The number of posets on n elements is the number of semilattices on n+1 elements; this number is also the number of lattices on n+2 elements. Recall from 3.1 that when structures have the same enumerator, it is reasonable to anticipate the existence of a generative correspondence. We are therefore not surprised to find one here.

To generate a list of semilattices on n elements, one proceeds as follows. Generate a list of posets on n-1 elements. For each poset produced, add an element which covers no element, and let it be covered by all previous lower bounds of the poset (the sinks of the digraph). One can readily convince oneself that if the posets produced are nonisomorphic, so are the semilattices. This process is repeated to produce lattices. To generate lattices on n elements, proceed as follows. Generate semilattices on n-1 elements. For each semilattice, add an element covered by nothing, and let it cover all previous upper bounds of the semilattice (sources of the digraph). Each lattice produced is unique up to isomorphism.

The generation of restricted classes of lattices can often be accomodated by the subgraph methods.

A related problem is the generation of quasi-orders. This class includes the posets; a quasi-order is reflexive and transitive, but not necessarily antisymmetric. We will not attempt the requisite definitions here, but rather give a quick survey for the reader interested in pursuing the topic. Quasi-orders are in one-one correspondence with finite topologies ([Anderson70]), whose generation has been considered in [Evans67]. Topologies are also in one-one correspondence with transitive digraphs. A subclass of the class of topologies, T(0)-topologies, were also generated in [Evans67], by exploiting a one-one correspondence with acyclic digraphs.

6.7.5 Rooted graphs

Rooted graphs do not seem to have been studied in the literature. Read's method is again applicable; one changes the group acting on the edge set to a group which **fixes** the root.

6.7.6 Multigraphs

Read's method applies equally well to multigraphs.

The autcmorphism partition of a multigraph can be computed by, for example, Mathon's algorithm ([Mathon77a], [Dymond76]).

Catalogues of multigraphs have appeared in the chemical literature; cubic muligraphs on $n \leq 10$ are listed in [Balaban70b], and multigraphs of degree at most four on $n \leq 5$ in [Balaban73].

6.8 Combinatorial configurations

In a work of full generality on this subject, one would ideally survey and examine generation methods for general combinatorial structures. An effort of this magnitude and complexity would prove fruitful, in that many sufficiently general methods could be proposed and specialised. It is unfortunate that the breadth of such a study precludes the possibility of supplying sufficient detail.

We present a brief overview in the hope of conveying two observations. The first is that many graph techniques are appropriate for other structures. The second is that some generation techniques employed successfully with other structures may prove practical for the generation of graphs.

6.8.1 Set systems

A k-set system is a generalisation of a graph, defined as follows. Let P(S) be the powerset of a set S. Let P be a set of sets. Let EP(k,P) be the set of k-subsets of P. A k-set system is a pair $\langle V,E \rangle$ where V is a vertex set (usually $\{1,2,\ldots,n\}$), and E, the edge set, is a subset of EP(k,P(V)). The nomenclature is natural, since the simple graphs are exactly the 2-set systems. The standard representation scheme for set systems is via incidence matrices. One can also represent a set system as a k-dimensional adjacency matrix. Let i(1), i(2), ..., i(k) be indices into this matrix. We concern ourselves only with index vectors into the matrix in which $i(j)<i(j+1)$ for $1 \leq j < k$, due to the symmetries of the matrix. One then forms a vector order for all index vectors which satisfy this restriction, as was done for graphs. This gives a vector form for set systems; we define canonicity as maximality within isomorphism class. Read's method can again be applied. The question of isomorph rejection here is a little more complex; an automorphism partition algorithm for set systems has not to our knowledge appeared in the literature.

6.8.2 Hypergraphs

A further generalisation of set systems and graphs appears in the literature; this is a structure known as a hypergraph. In a hypergraph, an edge can be incident with any positive number of vertices; in a set system this integer must be k. A formal definition of a hypergraph is a pair $\langle V,E\rangle$ where V is the vertex set, and E is a subset of $P(V)-\{\emptyset\}$. A k-hypergraph is a hypergraph in which no edge is incident with more than k vertices.

The generation of k-set systems can be restated, after dealing with a few minor details, for k-hypergraphs. In the construction of the allowable index sets into the adjacency matrix, we enforced the rule that the entries in the indices be strictly increasing. This corresponds when k=2 to considering only the upper triangle of the adjacency matrix. Consider the following example: we are to encode a 5-hypergraph. Since this is a hypergraph, and not necessarily a set system, we may have edges of cardinality less than 5. Say we have an edge (1,2,3). We can here represent this edge by the index 5-tuple (1,1,1,2,3). We therefore consider index tuples of the form $i(1), \ldots, i(k)$ where for $1\leq j<k$, either $i(j)<i(j+1)$ or $i(j)=i(j+1)=i(1)$. We again impose a vector order on this set of index tuples. Read's algorithm is applied using the vector forms induced. It is interesting to note that when k=2, k-hypergraphs are

just simple graphs with loops (that is, an edge from a vertex to itself).

The generation of hypergraphs is performed by this method; it is, after all, just the case k=n.

6.8.3 Set packings

Pursuant to successes with Read's method and the feasibility of isomorph rejection, we may anticipate success in coping with other combinatorial generation problems. Our anticipation has a fallacious assumption implicit in it, however; not all structures are fundamentally like graphs. We will consider two structures on which much research has been done, block designs, and a generalisation, set packings.

A (v,k,lambda) set packing is a set of subsets b(i) of EP(k,P({1,2,...,v})), for which the intersection of b(i) and b(j) contains lambda elements for i≠j. A recent comprehensive survey of generation methods for set packings appears in [Colbourn77]. The application of a method which operates well for graphs does not appear to be appropriate for one primary, reason: there does not appear to be a good vector representation for set packings. The straightforward representation is the same as that for set systems. It does not seem easy to enforce the intersection

condition with this representation without excessive loss of efficiency.

6.8.4 Block designs

We will not attempt to give more than a short introduction to the generation of block designs. An investigation has recently appeared which is devoted to that purpose ([Gibbons76]). The methods for block designs are based on the fact that one cannot guarantee to build up large designs from smaller ones. All designs on a given parameter set, unlike packings, have the same number of blocks. The methods given for graphs were inductive, and thus cannot apply here. The main emphasis with designs is on hill-climbing ([Shaver73], [Tompa75]) and backtracking ([Golomb65]). These techniques seem appropriate for restricted structures such as designs.

6.9 Conclusions

We have suggested a small portion of possible techniques for graph generation. We conclude that an efficient method to cope both with graphs and vastly different structures will not be found. A general method was recently proposed in [Faradzev76], which deals with all the structures mentioned; however, the time it saves in the generation of highly structured configurations, it

loses many times over when one turns to less structured classes.

The algorithm employs a branch-and-bound method ([Lawler66]), accepting the output structures for inclusion on the output list based on the value of a canonicity predicate, and on the value of a membership (in the class of interest) predicate. The implementation of these predicates is very general; as a result the testing is not as efficient as specialised routines for the given problem. In addition, the backtracking involved in branch and bound is manifestly a poor method for graphs.

We must attempt to employ methods which are as general as possible without sacrificing efficiency to too great an extent. We contend that Read's method is one such: as well as its many graph applications, it is sufficiently general to generate set systems and hypergraphs as well.

Chapter 7: Conclusions

We present a final analysis to formulate a concise description of current day technology, and to suggest fruitful lines of research.

7.1 The variety of methods

The most surprising observation is that there exists a great variety of algorithms. We have attempted to subdivide them into three classes: direct methods, methods based on existential methods, and specialisation and generalisation techniques. In the past, these methods have been examined as independent techniques. The methods involved have been largely treated independently as a result of the lack of an adequate survey. The aim of further research, we feel, should not require the examination of more and more methods, but should rather examine unifying features underlying many methods. The expansion of the class of orderly algorithms is such a goal, examined briefly with the revised versions of Heap's method and Farrell's method. The recent publication of

Read's method shows that such general methods exist; it remains to present further general techniques to cope with the great diversity of problems.

7.2 Graphs and restricted classes

In both Chapter 3 and 6, we were largely motivated by the investigation of generalisation and specialisation. The examination of trees in the third chapter concerned us primarily as a means to produce efficient general methods. Although generalisations were noted, our avowed aim remains unfulfilled. Producing a desirable generalisation remains an open problem.

In the investigation of other restricted classes in the sixth chapter, we focused primarily on the dual process of specialisation. This effort met with more success; the use of some slight knowledge of the structure of canonical forms allowed us to specialise some methods efficiently. The main credit for this success should properly be bestowed on Read's method, for its generality.

The primary aim must be the examination of efficient specialised methods with the goal of producing useful general methods. The secondary aim is precisely the dual. General methods must be examined to characterise the restricted problems for which they are appropriate.

7.3 Exhaustive and existential generation

In the generation of graphs, an area of study has been introduced which is novel; the foundation of exhaustive graph generation techniques on existential methods has not appeared in the literature. This lack is peculiar when one considers the investigations of this relation with other combinatorial generation problems. The algorithms we investigated are incremental. Some local change in a structure is performed to produce another structure. We have investigated the use of such algorithms in graph generation. In the fourth chapter, we emphasised that the existence of such methods provided a critical correspondence between existential and exhaustive techniques. This correspondence is critical for two reasons. On the one hand, the exhibition of incremental methods opens a new field of investigation; moreover, the incremental methods proposed may prove more efficient than previously known methods. On the other hand, this class of methods provides the opportunity to employ the well understood solutions to problems of existential generation as a partial solution in exhaustive generation; this is a new use for these techniques.

The focus of further research must be on the

demonstration of classes for which incremental methods are appropriate, and the demonstration of applicable incremental methods. An introduction to this line of research was begun with the investigation of completeness; this appears to be a reasonable starting point for further work.

7.4 The state of the art

We traced the evolution of direct methods from the classical method through iterative refinement to Read's method, and beyond. A careful examination of Read's method supplemented with some observations on graph enumeration provided some clues to performing isomorph rejection in the problem. Once again, we have fallen short of attaining our expressed goal. The success was, nevertheless, substantial in reducing the complexity of the elimination step.

Further research is required to provide methods to remove the elimination step, if possible.

7.5 More related topics

The thesis is permeated by yet another technique which we chose not to emphasise. The technique is that of providing a one-one correspondence between the structures of interest and an easily listed class of structures. This

technique was employed with the classes of trees with some success. The application of this technique to general problems appears to be of a higher order of difficulty than that of direct solution. Despite this warning, the technique may prove successful; nevertheless, we do not advocate its investigation with the same force as other lines of research.

Another problem which we have avoided until this point is the probabilistic generation of graphs. Many applications of the methods proposed would be at least as well served by the invocation of probabilistic graph generators to produce a sample list. One may object that existential generators serve this function already. There is, however, a decided difference. Existential generators fail to produce some graphs independent of the number of times the method is employed.

One may further object that the random graph generators in the literature ([Kuhn71], for example) serve the function of probabilistic generators. We must carefully examine what is required of a probabilistic method. Let there be g nonisomorphic graphs in the class of interest. A probabilistic generator will produce one of the g graphs selected from the list where each graph is selected with equal probability, $1/g$. Random graph

generators do not truly satisfy this requirement; in fact, no probabilistic generation methods are known to us excepting the brute force solution. One 'probabilistic' method for rooted tree generation, in [Nijenhuis75], essentially performs a brute force search.

The probabilistic generation of simple combinatorial sets has recently been investigated ([Williamson77]); Williamson shows that any set which can be ranked (placed in 1-1 correspondence with the integers) efficiently, can be generated efficiently. This supplies efficient probabilistic methods for simple structures such as permutations, and binary trees.

It seems inevitable that probabilistic generation methods will evolve. In fact, Lovasz has posed the problem of the existence of probabilistic generation methods for cubic graphs ([Lovasz77]). The production of exhaustive lists will not be superseded, but rather complemented, by them.

7.6 An eye to the future

Our final conclusion is one which we arrived at through many conversations, many correspondences, and many papers. We reluctantly note our conviction that the breadth and depth of graph generation as a study is

invariably underestimated.

Let us never lose the lessons we have learned.

References

For quick access, the page numbers on which an item is cited are listed on the top line of the entry.

[Aho74] 17
Aho, A.V.; Hopcroft, J.E.; Ullman, J.D. The Design and Analysis of Computer Algorithms Addison-Wesley (1974)

[Anderson70] 130
Anderson, S.S. Graph theory and finite combinatorics Markham Publ. (1970)

[Baker74] 3 86
Baker, H.H.; Dewdney, A.K.; Szilard, A.L. "Generating the 9 point graphs"; Math. Comp. 28 (1974) 833-838

[Balaban66] 112
Balaban, A.T. "Valence isomerism of cyclopolyenes"; Rev. Roumaine Chim. 11 (1966) 1097-1116

[Balaban70a] 112
Balaban, A.T.; Davies, R.O.; Harary, F.; Hill, A.; Westwick, R. "Cubic identity graphs and planar graphs derived from trees"; J. Austral. Math. Soc. 11 (1970) 207-215

[Balaban70b] 131
Balaban, A.T. "Chemical graphs VIII. Valence isomerism and general cubic graphs"; Rev. Roumaine Chim. 15 (1970) 463-486

[Balaban73] 131
Balaban, A.T. "Chemical graphs XVIII. Graphs of degree four or less, isomers of annulenes, and nomenclature of bridged polycyclic structure"; Rev. Roumaine Chim. 18 (1973) 635-653

[Barnette73] 118
Barnette, D. "Generating planar 4-connected graphs"; Israel J. Math. 14

(1973) 1-13

[Barnette74] 118
Barnette, D. "On generating planar graphs"; Discrete Math. 7 (1974) 199-208

[Birkhoff65] 18
Birkhoff, G.; MacLane, S. A Survey Of Modern Algebra Macmillan (1965)

[Birkhoff67] 127 129
Birkhoff, G.D. Lattice Theory AMS Colloq. Publ. (1967)

[Boesch74] 60
Boesch, F.T.; McHugh, J.A.M. "Synthesis of biconnected graphs"; IEEE Circuits and Systems CAS-21 (1974) 330-334

[Boesch76] 54 64
Boesch, F.; Harary, F. "Line removal algorithms for graphs and their degree lists"; IEEE Trans. Circuits and Systems CAS-23,12 (1976) 778-782

[Bondy76] 3 123
Bondy, J.A.; Hemminger, R.L. "Graph reconstruction - a survey"; Research Report CORR 76-49, Univ. of Waterloo (1976)

[Bondy77]
Bondy, J.A. private communications.

[Booth76] 20
Booth, K.S.; Lueker, G.S. "Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms"; JCSS 13,3 (1976) 335-379

[Booth77] 47
Booth, K.S. private communications

[Bowen67] 125
Bowen, R. "The generation of minimal triangle graphs"; Math. Comp. 21 (1967) 248-250

[Brown74a] 111
Brown, H.; Masinter, L. "An algorithm

for the construction of the graphs of organic molecules"; Discrete Math. 8 (1974) 227-244

[Brown74b] 111
Brown, H. "Molecular structure elucidation III"; STAN-CS-74-469 Stanford Univ. (1974)

[Bruckner1900] 10
Bruckner, M. Vielecke und Vielflache Teubner (1900)

[Bussemaker76] 112
Bussemaker, F.C.; Cobeljic, S.; Cvetkovic, L.M.; Seidel, J.J. "Computer investigation of cubic graphs"; T.H.-Report 76-WSK-01, Tech. Univ. Eindhoven (1976)

[Carhart75] 111
Carhart, R.E.; Smith, D.E.; Brown, H.; Sridharan, N.S. "Applications of artificial intelligence for chemical inference XVI. Computer generation of vertex graphs and ring systems"; J. Chem. Inf. Comput. Sci. 15,2 (1975) 124-130

[Cayley1857] 10
Cayley, A. "On the analytical form called trees"; Phil. Mag. 13 (1857) 172-176

[Cayley1859] 10
Cayley, A. "On the analytical form called trees. part two"; Phil. Mag. 18 (1859) 374-378

[Cayley1875] 10
Cayley, A. "On the analytical form called trees, with application to the theory of chemical combinations"; Brit. Assoc. Rep. for the Advancement of Science (1875) 257-305

[Chvatal69] 65 73
Chvatal, V. "Planarity of graphs with given degrees of vertices"; Nieuw Arch. Wisk. 17 (1969) 47-60

[Colbourn77] 19 134

Colbourn, M.L.J. "Analytic and computer techniques for set packings"; M.Sc. thesis, Univ. of Toronto (1977)

[Cook71] 80 123
Cook, S.A. "The complexity of theorem-proving procedures"; Proc. Third Annual ACM Symp. on the Theory of Computing (1971) 151-158

[Corneil68] 43
Corneil, D.G. "Graph isomorphism"; Ph.D. thesis, Univ. of Toronto (1968)

[Corneil72] 43 95 124
Corneil, D.G. "An algorithm for determining the automorphism partition of an undirected graph"; BIT 12 (1972) 161-171

[Corneil74] 19
Corneil, D.G. "The analysis of graph theoretical algorithms"; Univ. of Toronto DCS TR65 (1974)

[Corneil76] 124
Corneil, D.G.; Mathon, R. "Algorithmic techniques for the generation and analysis of strongly regular graphs and other combinatorial configurations"; Algebraic Aspects of Combinatorics (Qualicum Beach, B.C.) proceedings to appear (1976)

[deBruijn64] 17
deBruijn, N.G. "Polya's theory of counting"; in: Applied Combinatorial Mathematics (Beckenbach, E.F.; ed) Wiley (1964) 144-184

[deBruijn67] 21 22 31
deBruijn, N.G.; Morselt, B.J.M. "A note on plane trees"; JCT 2 (1967) 27-34

[deBruijn71] 17
deBruijn, N.G. "A survey of generalisations of Polya's enumeration theorem"; Nieuw Arch. Wisk. 2,19 (1971) 89-112

[Domb67] 5 116
Domb, C.; Heap, B.R. "The

classification and enumeration of multiply connected graphs"; Proc. Phys. Soc. London 90 (1967) 985-1001

[Dymond76] 95 124 131
Dymond, P.W.; Colbourn, C.J.; Zlatin, D. "An implementation of an algorithm of R. Mathon for the computation of automorphism groups"; unpublished manuscript (1976)

[Dyson47] 5
Dyson, G.M. A new notation for organic chemistry and its application to library and indexing problems Royal Inst. of Chemistry (1947)

[Edmonds64] 64
Edmonds, J. "Existence of k-edge connected ordinary graphs with prescribed degrees"; J. Res. Nat. Bur. Stand. 68B (1964) 73-74

[Eggleton75] 66
Eggleton, R.B. "Graphic sequences and polynomials : a report"; in: Infinite and Finite Sets (Hajnal, A.; Rado, R.; Sos, V.T.; eds) North-Holland (1975) 385-392

[Erdos60] 55
Erdos, P.; Gallai, T. "Graphs with prescribed degrees of vertices"; Mat. Lapok 11 (1960) 264-274

[Erdos74] 13
Erdos, P.; Spencer, J. Probabilistic methods in combinatorics Academic Press (1974)

[Evans67] 13 130 130
Evans, J.W.; Harary, F.; Lynn, M.S. "On the computer enumeration of finite topologies"; CACM 10 (1967) 295-298

[Faradzev76] 3 13 112 135
Faradzev, I.A. "Constructive enumeration of combinatorial objects"; Paris Conference (1976)

[Farrell71] 52 58 104

Farrell, E.J. "Computer implementations of an algorithm to generate subcompositions and applications to problems in graph theory"; Univ. of Waterloo M.Math. thesis (1971)

[Feigenbaum71] 5 111
Feigenbaum, E.A.; Buchanan, B.G.; Lederberg, J. "On generality and problem solving; a case study using the DENDRAL program"; in: Machine Intelligence 6 (Meltzer,B.;Michie, D.; eds) Edinburgh Univ. Press (1971) 165-190

[Gibbons76] 19 76 135
Gibbons, P.B. "Computing techniques for the construction and analysis of block designs"; Ph.D. thesis, Univ. of Toronto (1976)

[Golomb60] 14
Golomb, S.W. "A mathematical theory of discrete classification"; Information Theory, Fourth London Symposium (1960) 404-425

[Golomb65] 135
Golomb, S.W.; Baumert, L.D. "Backtrack programming"; JACM 12,4 (1965) 516-524

[Gordon48] 5 16
Gordon, M.; Kendall, C.E.; Dawson, W.H.T. Chemical ciphering Royal Inst. of Chemistry (1948)

[Grace65] 10
Grace, D.W. "Computer search for nonisomorphic convex polyhedra"; Stanford Computation Center Tech. Report No. 15 (1965)

[Hakimi62] 55 60
Hakimi, S.L. "On the realisability of a set of integers as degrees of the vertices of a linear graph"; J. SIAM 10 (1962) 496-506

[Hakimi63] 52 54 115
Hakimi, S.L. "On realizability of a set of integers as degrees of the vertices of a

linear graph II. Uniqueness"; J. SIAM 11 (1963) 135-147

[Hakimi69] 63 63
Hakimi, S.L. "An algorithm for construction of the least vulnerable communication network or the graph with the maximum connectivity"; IEEE Trans. CT-16 (1969) 229-230

[Hakimi73] 58
Hakimi, S.L.; Amin, A.T. "On the design of reliable networks"; Networks 3 (1973) 241-260

[Hakimi74] 62 62 63 63
Hakimi, S.L. "On the existence of graphs with prescribed degrees and connectivity"; SIAM J. Appl. Math. 26 (1974) 154-164

[Hall67] 17
Hall, M. Jr. Combinatorial Theory Wiley (1967)

[Harary62] 59
Harary, F. "The maximum connectivity of a graph"; Proc. Nat. Acad. Sci. USA 48 (1962) 1142-1145

[Harary64] 21 22
Harary, F.; Prins, G.; Tutte, W.T. "The number of plane trees"; Indag. Math. 26 (1964) 319-329

[Harary69] 17
Harary, F. Graph theory Addison-Wesley (1969)

[Harary73] 17 21 78 80 126
Harary, F.; Palmer, E.M. Graphical Enumeration Academic Press (1973)

[Harary77] 3
Harary, F. private communications.

[Havel55] 55
Havel, V. "A remark on the existence of finite graphs"; Casopes Pert-Mat. 80 (1955)

477-480

[Heap66] 116
Heap, B.R. "The enumeration of homeomorphically irreducible star graphs"; J. Math. Phys. 7 (1966) 1582-1587

[Heap72] 10 84 85
Heap, B.R. "The production of graphs by computer"; in: Graph Theory and Computing (Read, R.C.; ed) Academic Press (1972) 47-62

[Hirschberg73] 90 120
Hirschberg, D.; Edelberg, M. "On the complexity of computing graph isomorphism"; Tech. Rep. TR130, Dept. Elect. Eng., Princeton Univ. (1973)

[Hobbs73] 74 116
Hobbs, A.M. "A catalog of minimal blocks"; J. Res. Nat. Bur. Stan. 77B (1973) 53-60

[Hopcroft72] 20 33 44 45
Hopcroft, J.E.; Tarjan, R.E. "Isomorphism of planar graphs (working paper)"; in: Complexity of Computer Computations (Miller, R.E.; Thatcher, J.W.; eds) Plenum (1972) 131-152

[Hopcroft74] 47
Hopcroft, J.E.; Wong, J.K. "Linear time algorithm for isomorphism of planar graphs"; Proc. Sixth Annual ACM Symp. on the Theory of Computing (1974) 172-184

[Izbicki67] 11
Izbicki, H. "On the electronic generation of regular graphs"; in: Theory of Graphs Dunod (1967) 178-182

[Jackson77] 128
Jackson, D.M. private communications

[James74] 52
James, K.R.; Riha, W. "The production of trees and rooted trees of order $\leq$ 15, classified according to partition"; Tech. Report 47, Univ. of Leeds (1974)

[James76] 111
James, K.R.; Riha, W. "Algorithm 28 : algorithm for generating graphs of a given partition"; Computing 16 (1976) 153-161

[Kagno46] 10
Kagno, I. "Linear graphs of degree less than 7 and their groups"; Amer. J. Math. 68 (1946) 505-529

[Karp72] 80 123
Karp, R.M. "Reducibility among combinatorial problems"; in: Complexity of computer computations (Miller, R.E.; Thatcher, J.W. ; eds) Plenum (1972) 85-103

[Klarner69] 22
Klarner, D.A. "A correspondence between two sets of trees"; Indag. Math. 31 (1969) 292-296

[Kleitman73] 65
Kleitman, D.J.; Wang, D.L. "Algorithms for constructing graphs and digraphs with given valences and factors"; Discrete Math. 6 (1973) 79-88

[Knott77] 22 25 26
Knott, G.D. "A numbering system for binary trees"; CACM 20,2 (1977) 113-115

[Koren76] 55
Koren, M. "Sequences with a unique realisation by simple graphs"; JCT (B) 21 (1976) 235-244

[Kudo75] 112
Kudo, Y.; Sasaki, S-I. "Principle for exhaustive enumeration of unique structures consistent with structural information"; J. Chem. Inf. Comput. Sci. 16,1 (1975) 43-49

[Kuhn71] 141
Kuhn, W.W. "A random graph generator"; Proc. Third S.-E. Conf. on Combinatorics, Graph Theory, and Computing (1971) 311-313

[Lawler66] 136

Lawler, E.L.; Wood, D.E. "Branch and bound methods - a survey"; Operations Res. 14 (1966) 699-719

[Lawson70] 5
Lawson, J.D. "A note on Runge-Kutta processes for quadratic derivative functions"; Proc. Louisiana Conf. Comb., Graph Th., and Comput. (1970) 189-198

[Lederberg64] 4 44 112
Lederberg, J. "Notational algorithms for tree structures"; NASA Tech. Report CR 57029 (1964)

[Lederberg65] 4 112
Lederberg, J. "Topology of cyclic graphs"; NASA Tech. Report CR 68898 (1965)

[Li75] 55
Li, S-Y.R. "Graphic sequences with unique realisation"; JCT (B) 19 (1975) 42-68

[Lipson75] 18
Lipson, J.D. "Notes on applied algebra"; unpublished.

[Liu68] 17
Liu, C.L. Introduction to Combinatorial Theory McGraw-Hill (1968)

[Lovasz77] 142
Lovasz, L. the problem session, Waterloo Conf. on Graph Theory (1977)

[Lynch71] 5 111
Lynch, M.F.; Harrison, J.M.; Town, W.G.; Ash, J.E. Computer Handling of Chemical Structure Information American Elsevier (1971)

[Masinter74a] 112
Masinter, L.M.; Sridharan, N.S.; Lederberg, J.; Smith, D. "Applications of artificial intelligence for chemical inference XII Exhaustive generation of cyclic and acyclic isomers"; J. Amer. Chem. Soc. 96 (1974) 7702-7713

[Masinter74b] 112
Masinter, L.M.; Sridharan, N.S.; Carhart, R.E.; Smith, D.E. "Applications of artificial intelligence for chemical inference XIII Labelling of objects having symmetry"; J. Amer. Chem. Soc. 96 (1974) 7714-7723

[Mathon77a] 95 124 131
Mathon, R. "Sample graphs for graph isomorphism testing"; to appear as a Univ. of Toronto DCS technical report

[Mathon77b] 76
Mathon, R. private communications.

[McKay76] 16 16
McKay, B.D. "Backtrack programming and the graph isomorphism problem"; M.Sc. thesis, Univ. of Melbourne (1976)

[McKay77] 99
McKay, B.D. communications with R.C.Read (1977)

[Menon64] 52
Menon, V.V. "On the existence of trees with given degrees"; Sankhya Ser. A 26,1 (1964) 63-68

[Nash-Williams70] 65
Nash-Williams, C.St.J.A. "Valency sequences which force graphs to have Hamiltonian circuits"; Univ. of Waterloo tech. report (1970)

[Nijenhuis75] 142
Nijenhuis, A.; Wilf, H.S. Combinatorial Algorithms Academic Press (1975)

[Overton75] 16 81
Overton, M.; Proskurowski, A. "Finding the maximal incidence matrix of a large graph"; STAN-CS-75-509 Stanford Univ. (1975)

[Plemmons67] 19
Plemmons, R.J. "On computing

non-equivalent finite algebraic systems"; Math. Algorithms 2 (1967) 80-83

[Polya37] 17 36
Polya, G. "Kombinatorische Anzahlbestimmungen fur Gruppen, Graphen, und Chemische Verbindungen"; Acta Math. 68 (1937) 145-254

[Proskurowski73] 116
Proskurowski, A. "Automatic generation of graphs"; Ericsson Technics 29,2 (1973) 65-105

[Proskurowski74] 16 81
Proskurowski, A. "Search for a unique incidence matrix of a graph"; BIT 14 (1974) 209-226

[Rao70] 61
Rao, S.B.; Rao, A.R. "Existence of triconnected graphs with prescribed degrees"; Pacific J. Math. 33,1 (1970) 203-207

[Rao77] 65
Rao, S.B. "Characterization of forcibly line-graphic degree sequences"; Utilitas Math. 11 (1977) 357-366

[Read66] 10
Read, R.C. "The production of a catalogue of digraphs on 5 nodes"; Report UWI/CC1, Computing Centre, University of the West Indies (1966)

[Read69] 33
Read, R.C. "How to grow trees"; in: Combinatorial Structures and Their Applications (Guy, R.; Hanani, H.; Sauer, N.; Schonheim, J.; eds) Gordon and Breach (1969) 343-347

[Read70] 81
Read, R.C. "Graph theory algorithms"; in: Graph Theory and Its Applications (Harris, B.; ed) Academic Press (1970) 51-78

[Read72] 33
Read, R.C. "The coding of various kinds

of unlabelled trees"; in: Graph Theory and Computing (Read, R.C.; ed) Academic Press (1972) 153-183

[Read73] 18
Read, R.C.; Tarjan, R.E. "Bounds on backtrack algorithms for listing cycles, paths, and spanning trees"; Memo ERL-M433, Univ. of California at Berkeley (1973)

[Read76a] 6 15 17 81
Read, R.C.; Corneil, D.G. "The graph isomorphism disease"; J. Graph Th., to appear

[Read76b] 3 17 33 33 47 48 87 90 126
Read, R.C. "Every one a winner"; Algebraic Aspects of Combinatorics (Qualicum Beach, B.C.), proceedings to appear (1976)

[Read77] 34 108 126
Read, R.C. private communications.

[Riha74a] 52 52 53 111
Riha, W.; James, K.R. "The production of graphs realising a given partition"; Tech. Report 43, Univ. of Leeds (1974)

[Riha74b] 52 53 53
Riha, W.; James, K.R. "Algorithm for description and ordering of trees"; Tech. Report 54, Univ. of Leeds (1974)

[Riha76] 52 103
Riha, W.; James, K.R. "Algorithm 29: efficient algorithms for doubly and multiply restricted partitions"; Computing 16 (1976) 163-168

[Rotem75] 22 23
Rotem, D. "On a correspondence between binary trees and a certain kind of permutation"; Inf. Proc. Lett. 4,3 (1975) 58-61

[Rotem77] 22 24 24 26
Rotem, D.; Varol, Y. "Generation of binary trees from ballot sequences"; Research Report CS-77-29, University of

Waterloo (1977)

[Ruskey77a] 25
Ruskey, F.; Hu, T.C. "Generating binary trees lexicographically"; SIAM J. Comput., to appear.

[Ruskey77b] 29
Ruskey, F. "Generating t-ary trees lexicographically"; to appear.

[Ryser57] 68
Ryser, H.J. "Combinatorial properties of matrices of zeroes and ones"; Canad. J. Math. 9 (1957) 371-377

[Schmeichel77] 65
Schmeichel, E.F.; Hakimi, S.L. "On planar graphical degree sequences"; SIAM J. Appl. Math. 32,3 (1977) 598-609

[Scoins67] 20
Scoins, H.I. "Linear graphs and trees"; in: Machine Intelligence 1 (Collins, N.L.; Michie, D.; eds) American Elsevier (1967) 3-15

[Scoins68] 20 31 34 34 44 48
Scoins, H.I. "Placing trees in lexicographic order"; in: Machine Intelligence 3 (Michie, D.; ed) Edinburgh Univ. Press (1968) 43-60

[Senior51] 55
Senior, J.K. "Partitions and their representative graphs"; Amer. J. Math. 73 (1951) 663-689

[Shaver73] 19 66 135
Shaver, D.P. "Construction of (v,k,lambda) designs using a non enumerative search technique"; Ph.D. thesis, Syracuse Univ. (1973)

[Sheikh70] 112
Sheikh, Y.M.; Buchs, A.; Delfino, A.B.; Schroll, G.; Duffield, A.M.; Djerassi, C.; Buchanan, B.G.; Sutherland, G.L.; Feigenbaum, E.A.; Lederberg, J.

"Applications of artificial intelligence for chemical inference V Computer generation of cyclic structures"; Org. Mass Spect. 4 (1970) 493-501

[Smith74] 111
Smith, D.H.; Masinter, L.N.; Sridharan, N.S. "Heuristic DENDRAL: Analysis of molecular structure"; in: Computer Representation and Manipulation of Chemical Information (Wipke, W.T.; Heller, S.R.; Feldmann, R.J.; Hyde, E.; eds) Wiley-Interscience (1974) 287-316

[Sridharan73] 111
Sridharan, N.S. "Computer generation of vertex graphs"; STAN-CS-73-381 Stanford Univ. (1973)

[Sridharan74] 111
Sridharan, N.S. "A catalog of quadri/trivalent graphs"; STAN-CS-74-404 Stanford Univ. (1974)

[Swift60] 16 37
Swift, J.D. "Isomorph rejection in exhaustive search techniques"; Proc Symp. Appl. Math. X AMS (1960) 195-200

[Sykes66] 5 116
Sykes, M.F.; Essam, J.W.; Heap, B.R.; Hiley, B.J. "Lattice constant systems and graph theory"; J. Math. Phys. 7 (1966) 1557-1572

[Szilard77]
Szilard, A.L. private communications.

[Tapia67] 125
Tapia, M.A.; Myers, B.R. "Generation of concave node-weighted trees"; IEEE Trans. CT-14 (1967) 229-230

[Tarjan72] 18
Tarjan, R.E. "Depth first search and linear graph algorithms"; SIAM J. Comput. 1,2 (1972) 146-160

[Teh65] 125

Teh, H.H.; Jha, P. "Various constructions of regular graphs"; Bull. Math. Soc. Nanyang Univ. (1965) 31-44

[Tompa75] 19 67 135
Tompa, M. "Hill-climbing, a feasible search technique for the construction of combinatorial configurations"; M.Sc. thesis, Univ. of Toronto (1975)

[Trojanowski77a] 29
Trojanowski, A.E. "On the ordering, enumeration, and ranking of k-ary trees"; Univ. of Illinois Tech. Rep. UIUCDCS-R-77-850 (1977)

[Trojanowski77b] 29
Trojanowski, A. "Ranking and listing algorithms for k-ary trees"; submitted to SIAM J. Computing (1977)

[Tutte61] 117
Tutte, W.T. "A theory of 3-connected graphs"; Indag. Math. 23,4 (1961) 441-455

[Tutte76] 124
Tutte, W.T. "All the king's horses: a guide to reconstruction"; Research Report CCRR 76/37, Univ. of Waterloo (1976)

[Walker60] 16
Walker, R.J. "An enumerative technique for a class of combinatorial problems"; Proc. Symp. Appl. Math. 10 AMS (1960) 91-94

[Wang73] 56 62 62
Wang, D.L.; Kleitman, D.J. "On the existence of n-connected graphs with prescribed degrees ($n \geq 2$)"; Networks 3 (1970) 225-240

[Weinberg71] 4 4 117
Weinberg, L. "Linear graphs - theorems, algorithms, and applications"; in: Aspects of Network and System Theory (Kalman, R.E.; DeClaris, N.; eds) Holt, Rinehart, Winston (1971) 61-162

[Weisfeiler76] 124
Weisfeiler, B. On contruction and identification of graphs Springer- Verlag (1976)

[Wells71] 16 21 22 26 27
Wells, M.B. Elements of Combinatorial Computing Pergamon Press (1971)

[Whitehead73] 16
Whitehead, E.G. Combinatorial Algorithms Courant Inst. of Math. Sciences (1973)

[Whitney33] 118
Whitney, H. "A set of topological invariants for graphs"; Amer. J. Math. 55 (1933) 321-325

[Williamson77] 142
Williamson, S.G. "On the ordering, ranking, and random generation of basic combinatorial sets"; in: Combinatoire et representation du groupe symetrique (Foata, D.; ed) Springer-Verlag (1977) 309-339

[Wilf77]
Wilf, H.S. "A unified setting for sequencing, ranking, and selection algorithms for combinatorial objects"; Advan. Math. 24,3 (1977) 281-291

[Yap73] 64
Yap, H.P. "Point symmetric graphs with $p \leq 13$ points"; Nanta Math. 6,1 (1973) 8-20

[Zaks77]
Zaks, S. "Generating binary trees lexicographically"; Tech. Rept. UIUCDCS-R-77-888, Univ. of Illinois Urbana-Champaign (1977)

Index of symbols

Symbol	Meaning
$\in$	is a member of
$\lfloor x \rfloor$	floor of x.
$\lceil x \rceil$	ceiling of x.
$\|x\|$	absolute value of x
|	such that
P(v)	the powerset of a set v.
x*y or xy	the product of x and y
x/y	x divided by y
x(i)	the i'th component of a vector x.
x**y	x raised to the power y
$\square$	end of proof

Index of standard notation:

G, H	graphs
t	a tree
n	the number of vertices in the graph.
m	the number of edges in the graph
V, V(G)	the vertex set of a graph
E, E(G)	the edge set of a graph
L(i)	the list of the structures

	of interest of content i
A	a permutation group
D	the domain in the Polya theory
R	the range in the Polya theory