UCRL-51953

PQ-TREE ALGORITHMS

Kellogg Speed Booth
(Ph. D. Thesis)

November 14, 1975

Prepared for U.S. Energy Research & Development Administration under contract No. W-7405-Eng-48

LAWRENCE LIVERMORE LABORATORY
University of California/Livermore

Printed in the United States of America
Available from
National Technical Information Service
U. S. Department of Commerce
5285 Port Royal Road
Springfield, Virginia 22151
Price: Printed Copy $___*; Microfiche $2.25

*Pages	NTIS Selling Price
1-50	$4.00
51-150	$5.45
151-325	$7.60
326-500	$10.60
501-1000	$13.60

LAWRENCE LIVERMORE LABORATORY
University of California / Livermore, California / 94550

UCRL-51953

PQ-TREE ALGORITHMS

Kellogg Speed Booth

(Ph. D. Thesis)

November 14, 1975

PQ-Tree Algorithms

Kellogg S. Booth

Abstract

A new data structure called a PQ-tree is introduced. PQ-trees can be used to implement linear-time algorithms which test a graph for either the property of being an interval graph or the property of being a planar graph. The interval graph test is based on a theorem of Fulkerson and Gross which defines the consecutive ones property for matrices. The planarity test implements an algorithm of Lempel, Even, and Cederbaum. Using the techniques developed here, both tests can be performed in a number of steps which is proportional to the size of the graph being tested.

A representation of PQ-trees as formulas is provided by the class of PQ-formulas. It is shown that PQ-formulas form a lattice under two fairly natural operations, both related to the idea of grouping various parts of a formula into consecutive substrings. A special case of the lattice meet operation is called reduction. Reduction is used when testing a matrix for the consecutive ones property. It is also used to arrange the vertices of a planar graph. A linear-time algorithm for reduction is given which makes use of the PQ-tree data structure. The basic reduction algorithm is then extended, using a linear-time test for graph chordality, to an algorithm for testing interval graphs.

Properties related to the consecutive ones property and to interval graphs are examined. Some have linear-time algorithms, but others are shown to be NP-complete problems. Various practical applications are suggested, including efficient storage techniques for Gaussian elimination and information retrieval.

PQ-Tree Algorithms

Kellogg S. Booth

November 14, 1975

Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley CA 94720

and

Computer Systems Division
Lawrence Livermore Laboratory
University of California, Livermore CA 94550

Acknowledgment

Much of the material in chapter 2 was developed independently by George Lueker. The PQ-tree data structure and the tree version of the reduction algorithm are almost identical to the ideas he used in an interval graph test. His results are cited in the bibliography and mentioned in the text where appropriate. A number of discussions we have had contributed to my choice of notation and style of presentation, but this account is my version of the story. A preliminary report of our joint work was presented at the 1975 SIGACT Conference in Albuquerque.

The problems treated here were originally suggested to me by my advisor, Dick Karp. He has been a continuing source of ideas, constructive criticism and encouragement, and has never failed to push me in the right direction, even though there has sometimes been no little amount of resistance. Both he and Bob Tarjan have contributed much to this research.

This document was prepared using the TRIX/RED report-generating system, for which almost all credit is due Hank Moll. Somehow, despite all complaints, he has managed to remain his own cheerful self. The printing used the REDPP program of John Beatty, who spent many late-nite

hours reading rough drafts, listening, and playing carroms.

The work reported here was performed under the auspices of the Energy Resources Development Administration while on the staff of the Computer Systems Division at Lawrence Livermore Laboratory. My supervisor, Bob Lee, was more than understanding during this period. Along with the rest of the Computer Graphics Section, he provides a unique environment in which to work.

This thesis is dedicated to my wife, Laurie.

Contents

Symbols

Σ	The alphabet.
δ_S	The number of unary nodes deleted during S-reduction.
φ	A PQ-formula.
φ_S	The pertinent subformula of φ with respect to the set S.
$\mathcal{A}_\varphi$	The collection of underlying arrangements for the equivalence class of φ.
$\mathcal{P}_\Sigma$	The lattice of all PQ-formulas over the alphabet Σ.
$\mathcal{S}_\varphi$	The family of normal sets for the equivalence class of φ.
$\mathfrak{T}$	A PQ-tree.
$\mathfrak{T}_S$	The pruned pertinent subtree of $\mathfrak{T}$ with respect to the set S.
l^*	The maximum root-to-leaf distance in $\mathfrak{T}_S$.

0 Introduction

Using a new data structure called PQ-trees, two existing algorithms have been improved to run in linear time. Fulkerson and Gross devised an $O(n^3)$ test to recognize interval graphs[Ful1] and Lempel, Even, and Cederbaum used an $O(n^2)$ procedure to test graph planarity[Lem1]. Both of these algorithms reduce to the problem of placing objects from a set into a linear arrangement, subject to the constraint that certain subsets of the objects be consecutive. PQ-trees are an efficient way to represent the classes of linear arrangements which can arise under these circumstances.

Using the techniques developed here the interval graph test can be implemented so as to run in O(n+e) steps, for graphs with n vertices and e edges, while the planarity test requires only O(n) steps. Graph planarity has been treated extensively in the literature. Interval graphs are less frequently studied, but a number of applications related to Gaussian elimination[Ros1-2], information retrieval systems[Gho1-4], archeology[Ken1-2], psychology[Rob1-3], genetics[Ben1-2], ecology[Coh1], and the study of traffic lights[Sto1] have appeared in the literature. Many of these are mentioned in the text. All are referenced in the bibliography.

0 Introduction

PQ-trees were invented to test matrices for the consecutive ones property. The consecutive ones property was itself invented as an aid in testing for interval graphs, and is part of the Fulkerson-Gross interval graph algorithm[Ful1]. PQ-trees were recognized as a general data structuring technique when the consecutive ones test was posed as a problem of formula manipulation. This approach had already been used by Lempel, Even, and Cederbaum in testing for graph planarity[Lem1]. It happens that the class of formulas is identical for the two problems! This is a bit remarkable since the questions do not superficially appear to have much in common.

The unifying theme is the search, given a set, for all of the linear arrangements in which various subsets of that set are consecutive. This is a question of interest in its own right. Numerous formulations, all equivalent, have appeared in the literature. These include the consecutive ones property for matrices[Ful1], the consecutive retrieval property for information systems[Gho1-4], sequence dating in archeology[Ken1-2], representing sets by intervals[Esw2], and the intervalization of parsing tables[Jen1]. This thesis reports on a formalism for discussing all of these problems and on linear-time algorithms for solving them. The basic algorithms are then used to build linear-time recognition tests for both planar and interval graphs.

The first chapter is a development of the mathematical theory of PQ-formulas. They represent a formalism for describing the ways in which objects are grouped consecutively within linear arrangements. It is shown that PQ-formulas form an algebraic lattice and an algorithm is presented for reducing formulas with respect to a set. The operation of reduction is the mechanism by which objects are forced to remain consecutive within an arrangement.

The second chapter presents PQ-trees, a data structure for representing PQ-formulas. Efficient methods are developed for manipulating PQ-trees and the operation of reduction is extended to PQ-trees. An algorithm is exhibited which will perform a sequence of reductions on a tree in a number of steps which is linear in the size of the tree plus the sum of the sizes of the sets. The existence of this algorithm is the building block for most of the material in later chapters.

Planarity is the topic of the third chapter. The reduction algorithm is extended to a planarity test. This results in an O(n) version of the method used by Lempel, Even, and Cederbaum[Lem1]. The new implementation compares favorably with the linear algorithm of Hopcroft and Tarjan[Hop1].

Chapter 4 is devoted to the consecutive ones property and its variants, the circular ones property[Rys1], the circularly compatibles ones property, and the quasi-circular ones property[Tuc2-3]. Using the reduction algorithm, linear-time solutions are produced for all but the last of these. It is conjectured here that this final problem is NP-complete[Aho1, Gar1, Kar1-2].

A number of generalizations for the consecutive ones property are all shown to be NP-complete. Exact algorithms would be useful in the applications to information retrieval, sequence dating, or intervalization, but NP-completeness implies that practical solution methods should probably be based on heuristic techniques, unless special properties of the data can be used to obtain efficient solutions.

Chapter 5 contains a discussion of graph models for families of sets. The consecutive ones property was first defined for the purpose of testing for interval graphs, a proper subclass of the chordal graphs[Lek1]. Utilizing an O(n+e) test for chordality, the reduction algorithm can be used as a linear-time test for interval graphs. Both chordal and interval graphs are examples of intersection graphs for appropriate families of sets. Restricting the intersection models to be specific types of families yields different classes of graphs.

A review of of these classes is provided, including new proofs for some old theorems. The four-class hierarchy which begins with interval graphs and culminates in the chordal graphs is of particular interest because a by-product of the O(n+e) interval graph recognition algorithm is a linear-time isomorphism test for interval graphs[Bth1]. The corresponding test for chordal graphs would be extremely useful because it has been shown that isomorphism for chordal graphs is equivalent to the isomorphism question for arbitrary graphs[Bth1].

Intersection models are also of interest to numerical analysis; if graphs are viewed as matrices, the classes are a way of categorizing matrices according to their sparseness. The recognition algorithms can be applied to the zero / nonzero structure of sparse, symmetric, positive definite matrices to find efficient storage schemes for Gaussian elimination[Ros1-2].

The bibliography contains an annotated list of all of the references. An effort has been made to trace the historical development of the various concepts discussed in this thesis. It is hoped that this will serve as a convenient guide for further research into the many problems left unanswered here.

1 Formulas

This chapter introduces PQ-formulas, a formalism for describing certain linear arrangements for a set of objects. Equivalence transformations which induce a partitioning of PQ-formulas into equivalence classes are used to relate PQ-formulas both to strings and to sets. The notion of reducing a formula with respect to a set is defined and then generalized to a pair of operations, meet and join, under which the equivalence classes have the algebraic structure of a lattice. A preliminary algorithm for reducing formulas completes the material in this chapter; an efficient implementation is deferred until the next chapter where data structures for PQ-formulas are examined.

I Formulas

Before describing the techniques which have been developed it is necessary to present some definitions. The notation has been chosen to conform closely with the ideas in [Lem1], although some simplifications have been made. Let $\Sigma = \{a_1, \cdots, a_m\}$ be any set of atomic symbols or atoms. Atoms need have no properties other than that they be mutually distinguishable. A class of formulas can be defined which has the atoms as basic symbols. Each application interprets the formulas differently, so only structural properties will be analyzed in this chapter.

1 Formulas

The following rules define the class of PQ–formulas over an alphabet Σ. An equivalent definition could be made using BNF notation since the formulas form a context–free language over the terminal symbols $\Sigma \cup \{(,),[,]\}$. This is left as an exercise.

PQ–Formulas

[1] Each atom $a \in \Sigma$ is an elementary formula.

[2] The product $(\varphi_1 \cdots \varphi_k)$ is a P–formula whose factors are the PQ–formulas $\varphi_1, \cdots, \varphi_k$.

[3] The concatenate $[\varphi_1 \cdots \varphi_k]$ is a Q–formula whose components are the PQ–formulas $\varphi_1, \cdots, \varphi_k$.

[4] A PQ–formula over Σ is any elementary formula, P–formula, or Q–formula built up by the recursive use of these rules.

The usual notion of a subformula exists. Any substring of a PQ–formula φ which also satisfies the definition of a PQ–formula is called a subformula of φ. Every PQ–formula is a subformula of itself. Products and concatenates are called compound formulas; their factors or components are collectively referred to as constituents. In both of the applications which follow, it turns out that there is no need to distinguish between the product of two constituents and their concatenate. It is convenient to eliminate this redundancy and to also rule out any extraneous nesting of brackets. A product is regarded as proper only if it has at least two factors. A concatenate is proper only if it has at least three components. Extending this notion further, the entire formula is regarded as proper when every product and concatenate within it is proper. Only proper formulas will be considered here.

Example 1.1:

Suppose that the alphabet is given as the set

$$\Sigma = \{a,b,c,d,e,f,g,h,i,j,k\}.$$

The following are proper formulas. The last one will be used as an illustration throughout the chapter.

(abcdefghijk)	[(abcd)(efgh)(ijk)]
a	(aaaaaaaaaaaaaa)
(((ab)c)d)	[[[abc][def][ghi]]jk]
((abc)(abc)(abc)d)	(abc[d(ef)[(gh)i(jk)]])

These are improper formulas.

(a)	[ab]
[(abcdef)(ghijk)]	(((((((abc)))))))
([abababababa])	((a)[bb])

These are not formulas at all.

abcdefghijk	(((((())))))
((ab)[cde]]	(ab)[cde]
ab()dec	[dc(b])

Constituents are grouped by brackets much as operands within arithmetic expressions are grouped by parentheses. The definitions for PQ-formulas do nothing more than specify the legal ways in which brackets can be interspersed with atoms. The bracketing is very important and carries much, if not all, of the formula's meaning. The analogy with arithmetic expressions can be continued in this respect.

The associative nature of arithmetic allows more than two operands to be considered without specifying whether operations are performed left-to-right or right-to-left. When the operations are commutative, the operands can even be permuted. An expression will frequently have a number of equivalent forms because it is legal to rearrange the operands within a pair of parentheses.

There are usually a number of ways to write a PQ-formula, too. The rules are different than those for arithmetic expressions, however, because the interpretation of PQ-formulas is different. A product is both "associative" and "commutative," but a concatenate is only "associative." Within a compound formula the constituents can sometimes be reordered without changing the formula's meaning. Which reorderings are allowed depends upon the type of the formula.

Equivalence Transformations

[1] A P-formula $(\varphi_1 \cdots \varphi_k)$ can be replaced by any permutation $(\varphi_{\pi(1)} \cdots \varphi_{\pi(k)})$ of its factors.

[2] A Q-formula $[\varphi_1 \cdots \varphi_k]$ can be replaced by the unique reversal $[\varphi_k \cdots \varphi_1]$ of its components.

Two formulas φ and ψ are said to be equivalent if there exists a sequence of permutations and reversals which transform φ into ψ. These transformations define an equivalence relation which partitions the PQ-formulas into equivalence classes. It is these classes which turn out to be of interest. The classes correspond in a very natural way to sets of orderings for Σ.

Example 1.2:

The following formulas are all equivalent. They constitute an entire equivalence class.

(a[bcd]e)	([bcd]ae)
([bcd]ea)	(e[bcd]a)
(ea[bcd])	(ae[bcd])
(a[dcb]e)	([dcb]ae)
([dcb]ea)	(e[dcb]a)
(ea[dcb])	(ae[dcb])

Both of the formulas below are also in a common equivalence class, along with 766 other formulas.

(abc[d(ef)[(gh)i(jk)]]) (a[[(jk)i(hg)](ef)d]cb)

An arrangement of Σ is any linear ordering of the elements of Σ. The set of all arrangements for Σ is the same as the set of permutations on Σ, so the two terms are interchangeable. (This use of the word permutation should not be confused, however, with the permutation of a P-formula.) It is again convenient to restrict the type of formula to fit the applications; the permutations which will be useful are those on sets, not those on multisets. Attention will be focused on formulas in which each atom occurs exactly once. This means that each formula is merely a bracketed arrangement of Σ, an interpretation which will be convenient later.

Counting the number of formulas in an equivalence class is easy under these restrictions. Suppose that φ is a proper PQ-formula. Let

p = the number of subformulas of φ which are products,

and

q = the number of subformulas of φ which are concatenates.

Note that φ itself is included in either p or q, since it is a subformula of itself and must also be counted. The internal structure of the concatenates doesn't affect anything, but something has to be known about the products. Let

f_i = the number of factors in the i^{th} product.

where the indexing of the products is completely arbitrary. There is a simple expression for the size of a class.

Lemma 1.3:

The number of distinct PQ-formulas equivalent to a formula φ is exactly

$$n(\varphi) = 2^q \cdot \prod_{i=1}^{p} (f_i!)$$

Proof:

The proof is an easy induction on the number of atoms in a formula. More precisely, it is an induction on the number of atoms in Σ, the alphabet over which formulas are defined.

Assertion:
The above expression correctly computes the number of formulas in the equivalence class of φ when φ is a proper PQ-formula over the alphabet $\Sigma = \{a_1, \cdots, a_m\}$.

Basis:
When m = 1 every formula is just a single letter and only one arrangement is possible. There are no compound subformulas, so both p and q are zero. The expression evaluates to one, which is the correct value in this case.

Induction:
Let m > 1. If φ is a formula over Σ then each of the subformulas of φ occurs exactly once in φ. Each such occurrence is within

some constituent. The assignment of subformulas to constituents is not affected by an equivalence transformation. The left-to-right arrangement of constituents within φ is thus independent of the the left-to-right arrangements in which atoms appear within constituents. The lemma then follows from the induction hypothesis by noticing that the number of equivalent forms for each of the p products and q concatenates, except for φ itself, is already known to be counted correctly. The rest is easy.

Every formula within the equivalence class of φ can be constructed by first arranging the immediate constituents of φ and then arranging all of the subformulas within each constituent. The previous argument shows that these two steps are independent. If φ is a product, without loss of generality assume that it is the first one. The total number of equivalent formulas is then

$$n(\varphi) = (f_1!) \cdot \prod_{j=1}^{f_1} n(\varphi_j)$$

where the φ_j are the factors of φ. If φ is a concatenate the number of equivalent formulas is

$$n(\varphi) = 2 \cdot \prod_{j=1}^{g} n(\varphi_j)$$

where the φ_j are the components of φ and g is the number of components. In either case, product or concatenate, the result is merely a rearrangement of the numeric factors of the expression which was asserted to be correct.

QED

The preceding proof implicitly uses the fact that equivalence classes are nothing more than sets of bracketed arrangements. It is very convenient to be able to talk about equivalence classes in this manner.

Removing the brackets from a formula leaves a permutation of Σ. This is the underlying arrangement for the formula. The collection of underlying arrangements for an equivalence class consists of all underlying arrangements for formulas in the class, and is written as $\mathcal{A}_\varphi$. Frequently this will simply be called the collection of underlying arrangements for φ, the equivalence class being understood.

It will sometimes be convenient to refer to the empty collection of arrangements. The symbol 0 stands for the null formula. The null formula has no underlying arrangements. Its counterpart is the universal formula which is $(a_1 \cdots a_m)$ or any of its permutations. They are collectively abbreviated by the symbol 1. The underlying arrangements for the class include every possible ordering of Σ. Thus

$$\mathcal{A}_0 = \phi$$

and

$$\mathcal{A}_1 = \{ \pi \mid \pi \text{ is a permutation on } \Sigma \}.$$

The next lemma shows that the characterization of an equivalence class by its underlying arrangements is exact. The arrangements determine the formulas uniquely.

Lemma 1.4:

Two formulas φ and ψ are equivalent iff $\mathcal{A}_\varphi = \mathcal{A}_\psi$.

Proof =>:

Assume that φ is equivalent to ψ. The result is trivial since the equivalence classes have exactly the same formulas and thus the same set of underlying arrangements.

Proof <=:

Assume that $\mathcal{A}_\varphi = \mathcal{A}_\psi$. The conclusion follows from an induction on the number of atoms in the alphabet $\Sigma = \{a_1, \cdots, a_m\}$.

Assertion:

If φ and ψ each have m atoms and $\mathcal{A}_\varphi = \mathcal{A}_\psi$, then φ is equivalent to ψ.

Basis:

Let m = 1. There is only one PQ–formula!

Induction:

Let $m > 1$. Consider any two formulas φ and ψ each with m atoms. Both are necessarily compound so assume that

$$\varphi = (\varphi_1 \cdots \varphi_r) \quad \text{or} \quad [\varphi_1 \cdots \varphi_r]$$

and

$$\psi = (\psi_1 \cdots \psi_s) \quad \text{or} \quad [\psi_1 \cdots \psi_s].$$

Each formula is proper, hence r and s are each at least two.

Claim 1:

The two formulas have the same number of constituents: r = s.

Proof of claim 1:

Pick any constituent φ_p of φ. Its atoms must all appear somewhere in ψ. More importantly, they must be consecutive in ψ because they are consecutive in every underlying arrangement of φ's equivalence class and $\mathcal{A}_\varphi = \mathcal{A}_\psi$. Suppose they are in more than one constituent of ψ. The constituents must be consecutive in ψ.

If any one of these constituents is not composed entirely of atoms from φ_p then two adjacent constituents can be found, say ψ_q and ψ_{q+1}, such that one is not composed entirely of atoms in φ_p. But an equivalence transformation for ψ exists which simultaneously reverses the order of the atoms within ψ_q and the atoms within ψ_{q+1}, resulting in an underlying arrangement for ψ in which the atoms of φ_p are not consecutive. As noted before, this is a contradiction since the atoms are consecutive in every arrangement of φ's class and by assumption must also be consecutive in every arrangement of ψ's class.

The first step toward verifying the claim has been completed. No two constituents of ψ can contain atoms from φ_p unless each constituent is entirely composed of atoms from φ_p. In other words, each constituent of φ must either consist of atoms which are the exact union of the atoms in some set of the constituents of ψ or else all of its atoms must be contained within a single constituent of ψ. Suppose that the atoms of φ_p are in more than one constituent of ψ. There is at least one atom of φ not in φ_p because $r \geq 2$. This atom does not occur in any constituent of ψ in which atoms of φ_p occur. It follows that ψ is a concatenate since otherwise a permutation of its constituents could be made which would cause the atom not in φ_p to occur between two atoms belonging to φ_p.

But now, if the atoms of φ_p are in a concatenation of components of ψ, given two components of ψ, an atom which is not in φ_p always occurs in ψ closer to the atoms in one component than to those in the other and the closer component is always the same. This leads to a contradiction. Look at two atoms of φ_p which occur in distinct components of ψ. An atom not in φ_p is always closer, in ψ, to one than the other. This isn't true in φ since the entire constituent φ_p can be reversed, causing outside atoms to be closer to either of two specified atoms within φ_p.

This establishes the claim that $r = s$ since the atoms of a single φ_p are completely contained within some ψ_q and, by a symmetric argument, each ψ_q is completely contained within some φ_p. Moreover, this also shows that there is a 1-1 correspondence between the respective constituents such that paired constituents have exactly the same atoms. □

Claim 2:

The pairing between constituents of φ and ψ is an equivalence mapping.

Proof of claim 2:

It is easy to see that by restricting attention to just the atoms in a particular constituent the same arrangements will be obtained

in either φ or ψ. The induction hypothesis then applies and guarantees that corresponding constituents are equivalent. □

Claim 3:
φ and ψ are either both products or both concatenates.

Proof of claim 3:
The proof is completed by noticing that if r = 2 both formulas must be products because they are proper and if r > 2 and one were a product and the other a concatenate it would be possible to find three atoms x, y, and z from different constituents which always occur as x–y–z or z–y–x in the concatenate but occur as x–z–y in some permutation of the product. Once again this contradicts the assumption that $\mathcal{A}_\varphi = \mathcal{A}_\psi$. □

The two formulas are thus either both products or both concatenates and they have equivalent constituents. It follows that φ and ψ are equivalent.

QED

The key property used in the last proof is the way in which atoms appear together in adjacent groups. PQ–formulas are useful for precisely this reason. A subset $S \subset \Sigma$ is normalized in an arrangement of Σ if all the atoms of S form a consecutive subsequence in the arrangement. Specifically, no two atoms in S are separated by an atom not in S. The definition is easily extended to cover formulas. S is normalized in a formula φ if it is normalized in the underlying arrangement. An even stronger condition can be obtained by examining the entire equivalence class. S is a normal set for the formula φ if it is normalized in every member of the equivalence class of φ.

Let $\mathcal{S}_\varphi$ be the family of normal sets for φ. By definition these are exactly the sets which are consecutive in every formula of the equivalence class. Every set is considered normal for 0, the null formula, because there are no arrangements in which the sets must be normalized. The universal formula, with all of its freedom to rearrange, has hardly any normal sets — just Σ and its singletons.

$$\mathcal{S}_0 = 2^{\Sigma}$$

and

$$\mathcal{S}_1 = \{\Sigma\} \cup \left\{ \{a\} \mid a \in \Sigma \right\}.$$

The goal of this chapter is to characterize those formulas in which a given set is normalized. This requires a few more preliminaries. A formula equivalent to φ in which S is normalized is called an S-normalization of φ. Formulas having S-normalizations are said to be S-reducible. The choice of the term reducible may seem a bit confusing. It is used instead of normalizable for reasons which will become clear after more of the properties of PQ-formulas have been investigated. Testing reducibility will require examining the structure of a formula as it relates to the set S. To do this, it is convenient to have a way of classifying various parts of a formula.

An atom in φ is pertinent to S if it is a member of S; a product or concatenate is considered pertinent exactly when it contains one or more pertinent atoms. The pertinent subformula of φ, with respect to S, is the shortest subformula which contains all of S. It is unique and is denoted by φ_S. Notice that not every subformula of φ_S is pertinent. This will be a matter of concern when constructing efficient algorithms for handling formulas.

Example 1.5:

Consider one of the formulas from example 1.1.

(abc[d(ef)[(gh)i(jk)]]).

If the set in question is chosen to be

S = {e,i,j,k}

then the pertinent subformula is

[d(ef)[(gh)i(jk)]].

The pertinent atoms are e, i, j, and k; the nonpertinent atoms are a, b, c, d, f, g, and h. All of the other subformulas are pertinent except for (gh). This last subformula is an example in which a subformula of the pertinent subformula is not pertinent.

As the name suggests, the pertinent subformula is the important part of φ when dealing with S. Whether φ is S-reducible depends only upon the structure of the pertinent subformula. Intuitively, parts of the formula in which a set does not occur cannot affect the consecutiveness of its atoms. This is the next lemma.

Lemma 1.6:

A formula φ is S-reducible iff φ_S is S-reducible.

Proof =>:

If φ is S-reducible it has an S-normalization φ'. The subformula of φ' corresponding to φ_S will be S-normalized in φ' since any subformula of an S-normalized formula is automatically S-normalized. This is an S-normalization of φ_S, hence φ_S is S-reducible.

Proof <=:

If φ_S is S-reducible apply the equivalence transformations which S-normalize φ_S to all of φ. S occurs only in φ_S so φ is also normalized. This implies that φ is S-reducible.

QED

II Reduction

Testing a formula for reducibility can be accomplished by assigning labels to subformulas. The classification into pertinent and nonpertinent is not sufficient for this task. A finer distinction is

required. A subformula may receive one of four labels: empty, full, singly partial, or doubly partial. It may also remain unlabeled. The choice depends upon how many of its atoms are pertinent and the way they are distributed among constituents. The idea behind the labeling is to keep track of the freedom available for rearranging constituents within subformulas. Clearly all of the pertinent atoms have to be arranged consecutively, so keeping track of where they occur is a good strategy. The labels perform this bookkeeping.

No atom of an empty subformula is pertinent. Every atom of a full subformula is pertinent. Neither of these need be treated any differently than the way a single atom is handled. All the complications arise when some atoms are pertinent and some are not. A singly partial subformula is one to which pertinent atoms can be attached on only one end, the other end being unavailable. Doubly partial subformulas cannot be built upon at all. Both ends are unavailable. At a general step, having arranged some of the atoms, it will be possible to add pertinent atoms to both ends of the arrangement, to only one end, or to neither end. This is the point of the labels. Exact definitions for the terms are provided by step 3 of the next algorithm.

The algorithm assigns labels to the subformulas until it eventually labels φ_S, if it is S-reducible. The order in which subformulas are processed is governed entirely by the way they are placed on a labeling queue. Among other things, this guarantees that the subsubformulas of a subformula have been processed by the time the subformula itself is processed. Note that this algorithm does not actually perform the rearrangement. It only checks a formula φ to see if it can be S-normalized.

Normalization

[1] Assume that every subformula of φ is initially unlabeled and that the queue is empty. Place each atom on the queue to be labeled. Any order is acceptable at this point.

[2] Remove the formula ψ at the head of the queue. It is the next

candidate for labeling.

[3] Label ψ choosing the first applicable case below. Note that the order in which alternatives are considered is important. If none of these cases apply, halt with the answer "irreducible."

Case 1:

ψ = a, an atom. There are two subcases to handle.

a) If a $\notin$ S label ψ empty.

b) If a $\in$ S label ψ full.

Case 2:

$\psi = (\psi_1 \cdots \psi_k)$, a product. This time there are four subcases.

a) If all factors are empty label ψ empty.

b) If all factors are full label ψ full.

c) If at most one factor is singly partial, all others empty or full, label ψ singly partial.

d) If at most two factors are singly partial, all others empty or full, label ψ doubly partial.

Case 3:

$\psi = [\psi_1 \cdots \psi_k]$, a concatenate. There are again four subcases.

a) If all components are empty label ψ empty.

b) If all components are full label ψ full.

c) If at most one component is singly partial label ψ singly partial, all other components empty or full. Do this only if all full components occur first (or last) and the partial component (if present) separates the full components from the empty ones.

d) If at most two components are singly partial label ψ doubly partial, all other components empty or full. Do this only if all full components occur together and the partial components (if present) separate the full components from the empty ones, one on each side of the full components.

[4] If ψ was the last constituent without a label in some formula within φ_S, append the enclosing formula to the tail of the queue.

[5] Go back to step 2 if the queue is not empty, otherwise halt with the answer "reducible."

This first algorithm will be the basis for much of what follows. The scheme used to categorize subformulas will be repeated again and will govern the transformations applied to formulas. The astute reader may well question whether these steps provide an efficient test for S-reducibility as they are stated here. They do not. This will be attended to in due time. The goal at this point is to explain the basic mechanism by which formulas are analyzed. Subsequent versions will use more selective queueing criteria to eliminate unnecessary work, but the underlying inside-out organization will remain.

Example 1.7:

Using the same formula as Example 1.5, let the set again be

$S = \{e,i,j,k\}$.

The labels generated by the algorithm are the following.

·····(ef)·····	Singly partial
·····(gh)·····	Empty
·····(jk)·····	Full
·····[(gh)i(jk)]··	Singly partial
····[d(ef)[(gh)i(jk)]]·	Doubly partial
(abc[d(ef)[(gh)i(jk)]])	Unlabeled

The formula itself does not receive a label because one of its constituents (φ_S) is doubly partial. The algorithm answers "reducible," however, because φ_S receives a label.

The overall flow of the algorithm is quite simple. Beginning with the atoms, each subformula is processed using the results already obtained for its constituents. This continues until all of φ_S has been labeled or until some part cannot be processed. In either case the queue eventually empties and the algorithm terminates. In general, the queueing process does not label everything in a formula. If φ cannot be

S-normalized at least one of the subformulas of φ_S will not be labeled. This is an immediate corollary of the next lemma and is the justification for calling the algorithm a test for S-normalization.

Lemma 1.8:

A formula φ is S-reducible iff the normalization algorithm labels φ_S.

Proof =>:

The labeling of a subformula of φ depends only on the labels of its constituents. It is invariant under an equivalence transformation. Thus it is sufficient to consider a single member for each equivalence class. Which member is arbitrary, so it pays to choose a convenient one. Since φ is S-reducible, assume that it is S-normalized. An induction on the number of times step 2 is executed will establish that every subformula of φ_S is labeled by the algorithm and hence φ_S itself is eventually labeled.

Assertion:
Every subformula of φ_S selected by step 2 during the first m iterations is labeled at step 3.

Basis:
Let m = 0. The claim holds vacuously since no formulas have been selected.

Induction:
Let m > 0 and assume that the subformula ψ is selected during the m^{th} iteration. Because of the way in which subformulas are placed on the queue every one of ψ's constituents must already have been selected as a candidate for labeling. The induction hypothesis guarantees that every constituent has received a label. There are a number of cases to consider for ψ, corresponding to the different situations which can be encountered in step 3.

Case 1:

ψ is an atom. Every atom gets labeled by step 3 because the two subcases are exhaustive. It is either empty or full.

Case 2:

ψ is a P-formula. If at most two factors are labeled singly partial (none being doubly partial) one of the four subcases occurs. Any other possibility contradicts the assumption that φ was S-normalized, since at least two pertinent atoms would be separated by a nonpertinent atom.

Case 3:

ψ is a Q-formula. Again the assumed S-normalization of φ rules out anything other than the four subcases recognized by the algorithm.

Each subformula of φ_S selected by step 2 is labeled at step 3, the immediately enclosing formula being appended to the queue if it is not outside of φ_S. This insures that sooner or later φ_S is also labeled.

Proof <=:

Assume that the normalization algorithm labels φ_S. A similar induction on the number of times the loop is executed shows that there exists an equivalence transformation for φ which S-normalizes φ_S. By lemma 1.6 this is sufficient.

Assertion:

There is an equivalence transformation for φ such that every subformula of φ_S selected during the first m iterations is S-normalized.

Basis:

Let m = 0. The assertion is again vacuously true, no formulas having been selected.

Induction:

Let $m > 0$ and assume that ψ is chosen at the m^{th} iteration. If ψ is not in φ_S there is nothing to prove. Otherwise, one of the cases of step 3 applies to ψ since it receives a label. A case-by-case analysis shows that the current transformation can be converted to one which also S-normalizes ψ.

Case 1:

ψ is an atom and hence trivially S-normalized by any transformation.

Case 2:

ψ is a P-formula. The cases in which it is empty or full are easy since ψ is already S-normalized. If ψ is singly partial permute the factors so that all of the empty factors occur before the full ones and the partial factor (if present) lies in between. This is an S-normalization of ψ.

The case of two partial factors is similar, except that the full factors are permuted so as to lie between the partial ones. The partial factors must be next to the full factors but can be reversed, any partial subformulas also being reversed, so that all of the pertinent atoms are consecutive. These are all equivalence transformations and will not affect the S-normalization of any previously considered subformula.

Case 3:

ψ is a Q-formula. Again the empty and full cases are already handled. It is easy to verify that except for its partial components ψ is already S-normalized. Just as in case 2 the partial components can be reversed, if necessary, to yield an equivalence transformation which is compatible with previous S-normalizations and which also S-normalizes ψ.

QED

This last lemma does more than just assert the validity of the labeling. It gives an explicit method for S-normalizing an S-reducible formula. Applying the permutations and reversals at the same time each subformula

is labeled produces an S-normalized version of φ after φ_S is labeled in step 3. Thus a set S can be made to occur consecutively in φ using a simple set of rewriting rules and applying them inside-out as in the algorithm. This rearrangement is the point of normalization. Note carefully, however, the difference between an S-normalization of φ and the case in which S is a normal set. In the former case, the consecutive arrangement is not permanent. The normalization can be undone. Forcing S to remain consecutive, no matter what equivalence transformation is applied, is the real goal.

Example 1.9:

Continuing example 1.7, the normalizing transformations are given below.

·····(fe)·····	Permuted factors
·····(gh)·····	No change
·····(jk)·····	No change
·····[(gh)i(jk)]··	No change
····[d(fe)[(kj)i(hg)]]·	Reversed component
(abc[d(fe)[(kj)i(hg)]])	Normalized

In this example, there are a number of other admissible normalizations. Any permutation of a, b, c, and the outermost concatenate will work. There are also others.

Lemma 1.8 shows that in an S-reducible formula φ every subformula of φ_S must satisfy one of the cases of step 3. This classification into empty, full, and singly or doubly partial is very important and will be used in all subsequent versions of the algorithm. It is convenient to have a notation which describes the constituents of a formula ψ in this context. Let the empty constituents be denoted by $\psi_1^e, \cdots, \psi_r^e$ and the full constituents by $\psi_1^f, \cdots, \psi_t^f$. There can be at most two partial constituents, both singly partial, if the subformula is to receive a label. Let these be φ^p and φ^q. It turns out that these are unimportant

themselves, but their subconstituents are important. Denote them by $\psi_1^p,\cdots,\psi_s^p$ and $\psi_1^q,\cdots,\psi_u^q$.

Using all of these pieces it is possible to build a new formula, closely related to φ, in which S is normal. Once a set is normal in the formula, no equivalence transformation can force the atoms of S to be anything except consecutive. Building the new formula is accomplished by the next algorithm which is a modification of the normalization algorithm. It uses the labeling generated there to control the way in which subformulas are processed. Such a transformation is called reduction.

Reduction

[1] Let φ be an S-normalized formula such that every subformula of φ_S has been labeled by the normalization algorithm [page 18]. If φ can't be normalized, set $\varphi = 0$, the null formula, and halt immediately with the answer "irreducible." Start with an empty queue. Place each atom on the queue to be reduced.

[2] Remove the formula ψ at the front of the queue. It is the next candidate for reduction.

[3] Reduce ψ, choosing the first applicable case below. As before, the order in which alternatives are considered is important.

Case 1:

ψ is an atom. No changes ever have to be made. Leave it alone.

Case 2:

ψ is a P-formula. There are four subcases, one for each possible label from the normalization algorithm.

a) Leave it alone if ψ is empty. It is already reduced.

b) Leave it alone if ψ is full. This is reduced too.

c) Replace ψ with the concatenate

$$[(\psi_1^e\cdots\psi_r^e)\psi_1^p\cdots\psi_s^p(\psi_1^f\cdots\psi_t^f)]$$

if ψ is singly partial, (Throughout it is assumed that no improper subformulas are introduced. This can be

accomplished by removing extraneous brackets when they occur and by replacing concatenates of two components by their product.) Use the product

$$(\psi_1^e \cdots \psi_r^e[\psi_1^p \cdots \psi_s^p(\psi_1^f \cdots \psi_t^f)])$$

instead for the special case in which $\psi = \varphi_S$.

d) Replace ψ with the product

$$(\psi_1^e \cdots \psi_r^e[\psi_1^p \cdots \psi_s^p(\psi_1^f \cdots \psi_t^f)\psi_u^q \cdots \psi_l^q])$$

if ψ is doubly partial.

Case 3:

ψ is a Q-formula.

a) Leave it alone if ψ is empty. It's already reduced.

b) Leave it alone if ψ is full. This is reduced too.

c) Replace ψ with the concatenate

$$[\psi_1^e \cdots \psi_r^e \psi_1^p \cdots \psi_s^p \psi_1^f \cdots \psi_t^f]$$

if ψ is singly partial.

d) Replace ψ with the concatenate

$$[\psi_1^e \cdots \psi_v^e \psi_1^p \cdots \psi_s^p \psi_1^f \cdots \psi_t^f \psi_u^q \cdots \psi_l^q \psi_{v+1}^e \cdots \psi_r^e]$$

if ψ is doubly partial. (Note that the empty components may be split with full and partial components in between. This is indicated by the fact that the first v empty components, $1 \leq v \leq r$, appear on the left and the remaining ones on the right.)

[4] If ψ was the last unreduced constituent of some compound formula within φ_S append the enclosing formula to the end of the queue.

[5] Go back to step 2 if the queue is not empty, otherwise halt with the anser "reducible."

When φ is S-reducible the reduction algorithm computes a new formula called the S-reduction of φ. It is easy to see that the result of the algorithm is a formula in which S is normal.

Example 1.10:

Finishing example 1.9, the reductions to be performed are listed below for all the subformulas except the atoms, which remain unchanged.

·····[fe]·····	Changed to concatenate
·····(kj)·····	No change
·····(hg)····	No change
·····[(kj)i(hg)]··	No change
····[dfe(kj)i(hg)]·	New concatenate built
(abc[dfe(kj)i(hg)])	Reduced

Note that [fe] is not proper after it is reduced. This is all right since it will become part of a proper concatenate later. This will always be the case any time a P-formula is changed to a Q-formula; it eventually becomes part of a larger Q-formula.

Any subformula which is partial will be replaced by a Q-formula except for φ_S, which is a special case. After all of the replacements, every subformula is either empty or full; there are no partial subformulas within φ_S after reduction. A precise statement of this fact is the aim of the next lemma.

Lemma 1.11:

Let φ be an S-reduced formula. Every subformula of φ_S is either empty or full, except possibly φ_S. It may be partial if it is a concatenate, but all of its full components must be consecutive.

Proof:

This is immediate after examining the result for each case in step 3 of the reduction algorithm.

QED

The transformations applied in the normalization algorithm usually admit more than one S-normalization for a formula. Each yields a different reduced form when the reduction algorithm is applied. The various S-reductions will not differ significantly, however. The next lemma compares the formulas which can be achieved via the reduction process.

Lemma 1.12:

Any two S-reductions of φ are equivalent.

Proof:

Given any two S-normalizations φ_1 and φ_2, there is an obvious equivalence mapping between subformulas. The claim is that this equivalence is preserved after reduction and hence that the two S-reductions are themselves in the same equivalence class. The proof is an induction on the number of times step 2 is executed while processing φ_1.

Assertion:
Assume that the S-reduced forms of the subformulas of φ_1 which are reduced during the first m iterations of step 2 are equivalent to the S-reduced forms of their corresponding subformulas in φ_2.

Basis:
Let m = 0. Any mapping works since no subformulas have been reduced.

Induction:
Let m > 0 and assume that ψ_1 is selected for reduction in φ_1 at the m^{th} iteration. There is a 1-1 equivalence between

constituents of ψ_1 and those of ψ_2. its image in φ_2. By the induction assumption these constituents are pairwise equivalent after reduction.

The labels assigned during normalization will be the same in ψ_1 and ψ_2 so the only difference is the exact order in which the constituents are reassembled. This is always an equivalence transformation, hence the reduced φ_1 and φ_2 retain their prior equivalence.

QED

The normalization and reduction algorithms can be combined into a single algorithm which, given a formula φ and a set S, computes a new formula in which S is normal. The fact that this works for any S-reducible formula prompted the use of the term reducible in the first place. This is the point of the next lemma, which directly characterizes the S-reduction of a formula in terms of its underlying arrangements.

Lemma 1.13:

Given a formula φ and a set S there is a formula φS whose underlying arrangements are exactly the underlying arrangements of φ in which S occurs consecutively. Furthermore φS is unique up to an equivalence transformation.

Proof:

There are two cases to consider depending upon whether or not φ is S-reducible.

Case 1:
If S cannot be normalized in φ there are no underlying arrangements in which S occurs consecutively. The null formula is the unique PQ-formula which has no underlying arrangements. Let $\varphi S = 0$.

Case 2:

Assume that φ is S-reducible. The underlying arrangements of φ in which S is consecutive correspond 1-1 with the S-normalizations of φ. This is because each S-reduction of φ is a rebracketing of exactly one of the S-normalizations and hence collectively they have precisely the desired underlying arrangements. Let φ_S be any S-reduction of φ. Lemma 1.4 insures uniqueness to within an equivalence transformation.

QED

Note that in this interpretation every formula has an S-reduction. If φ is not S-reducible the S-reduction is 0, the null formula. In all cases the reduction process leads to a new equivalence class with a smaller (reduced) set of underlying arrangements. It throws out exactly those arrangments in which S does not occur consecutively. The structure of PQ-formulas is thus able to capture the essence of consecutive occurrence. This point is reinforced by looking at the normal sets for a formula.

Corollary 1.14:

For any formula φ and any two subsets S_1 and S_2 of Σ, $\varphi S_1 S_2$ is equivalent to $\varphi S_2 S_1$.

Proof:

From the lemma both double reductions have the same underlying arrangements. Again using lemma 1.4, the formulas must be unique up to an equivalence transformation.

QED

The normal sets can be used to obtain a characterization similar to that of lemma 1.4, which related a formula to its collection of underlying arrangements.

Lemma 1.15:

Two formulas φ_1 and φ_2 are equivalent iff $\mathcal{S}_{\varphi_1} = \mathcal{S}_{\varphi_2}$.

Proof =>:

This is immediate from the definition of normal sets. The equivalence classes are the same, so the normal sets are identical.

Proof <=:

Suppose that $\mathcal{S}_{\varphi_1} = \mathcal{S}_{\varphi_2}$. The proof is by induction on the number of atoms in a formula.

Assertion:
All subformulas in φ_1 having no more than m atoms have a corresponding equivalent subformula in φ_2.

Basis:
Let m = 1. The only such formulas are the atoms themselves. Elementary formulas correspond by the identity mapping and are obviously equivalent.

Induction:
Let ψ_1 be a subformula of φ with m > 1 atoms. Since each of its constituents has less than m atoms, they all satisfy the induction hypothesis and have equivalent images in φ_2. These images must be exactly the constituents of some ψ_2.

Suppose this were not the case. The atoms of ψ_1 would not be consecutive in φ_2 but would be consecutive in φ_1. This is a contradiction because they comprise a normal set for φ_1 and therefore must also be normal in φ_2. An argument similar to that of lemma 1.4 then establishes that ψ_2 must actually be equivalent to ψ_1. This completes the induction and the proof.

QED

III Characterizations

Equivalence classes can be thought of either as collections of arrangements or as families of subsets. Either notion characterizes a class completely. The normal sets are sometimes more convenient, but they require a knowledge of the inner structure of $\mathcal{S}_\varphi$. The first step toward this understanding is to examine the closure properties of such families.

A definition is handy. Two sets S and T overlap whenever they have a nonempty intersection and are incomparable with respect to set containment. An equivalent definition is to demand that each of $S \cap T$, $S - T$, and $T - S$ be non-empty. Any family of normal sets must be closed under a number of operations on overlapping sets.

Lemma 1.16:

If S and T are overlapping sets in $\mathcal{S}_\varphi$ then each of $S \cup T$, $S - T$, and $S \cap T$ is also in $\mathcal{S}_\varphi$.

Proof:

Suppose that S and T overlap and are both in $\mathcal{S}_\varphi$. Consider any representative from φ's equivalence class. S and T are normal so they occur consecutively. All three of $S \cup T$, $S - T$, and $S \cap T$ must also be consecutive. Because the representative was arbitrary this is true for every formula in the class, hence the sets are normal and members of $\mathcal{S}_\varphi$.

QED

Stated another way, the lemma says that the normal sets are always a family closed under operations on overlapping sets. The strict converse is not true, but the closure properties are almost all that is needed to determine $\mathcal{S}_\varphi$. A family closed under the previous operations need not correspond to the normal sets of any formula. $\mathcal{S}$ is consistent iff the universal formula can be simultaneously reduced by all of the sets in $\mathcal{S}$. This means that some nonnull formula contains $\mathcal{S}$ within its family of

normal sets.

Given a consistent family $\mathcal{S}$ which is closed under union, difference, and intersection of overlapping sets, a PQ-formula can be constructed which has $\mathcal{S}$ for its family of normal sets provided the family contains a minimal subfamily of required sets.

Lemma 1.17:

$\mathcal{S}$ is the family of normal sets for some PQ-formula over Σ iff either $\mathcal{S}$ is 2^{Σ} or else it is consistent, contains Σ and every singleton of Σ, and whenever any two overlapping sets S and T are in $\mathcal{S}$ all three of $S \cup T$, $S \cap T$, and $S - T$ are also in $\mathcal{S}$.

Proof =>:

Clearly every singleton of Σ as well as Σ itself is always consecutive in any arrangement, so each is a normal set and must be in $\mathcal{S}$. The closure properties of lemma 1.16 complete the proof.

Proof <=:

The first condition is easy to dispose of. It is just a technicality. If $\mathcal{S}$ is all of 2^{Σ} it is $\mathcal{S}_0$. This leaves the other alternative. So assume that $\mathcal{S}$ is consistent. Recall that 1 is the universal formula. The definition of consistency requires that the universal formula be simultaneously reducible by all of the sets.

Define a formula determined from the sets by the reductions

$$\varphi = 1S_1 \cdots S_n$$

where

$$\mathcal{S} = \{ S_1, \cdots, S_n \}.$$

From lemma 1.11 it is clear that each of the sets is normal in φ. Thus

$$\mathcal{S} \subset \mathcal{S}_{\varphi}.$$

To actually show equality the reverse containment must also be exhibited. Let

$$\varphi_0 = 1$$

and

$$\varphi_j = \varphi_{j-1} S_j$$

for $1 \leq j \leq n$. The proof is an induction on j.

Assertion:
For all $j \geq 0$, the containment

$$\mathcal{S}_{\varphi_j} \subset \mathcal{S}$$

is valid.

Basis:
Let $j = 0$. Then $\varphi_0 = 1$. The only normal sets for 1 are Σ and its singletons. This implies that

$$\mathcal{S}_{\varphi_0} \subset \mathcal{S}.$$

Induction:
Let $j > 0$. Consider the effect of the reduction algorithm on φ_{j-1}. The resulting formula φ_j has all of the previous normal sets plus some new ones. A new normal set will be introduced only when a change is made in the formula. This, in turn, happens only when a partial subformula is reduced, which itself happens only when S_j overlaps one of the existing normal sets. The added normal set is always either a union, intersection, or differece of S_j and this previous normal set. The induction hypothesis guarantees that the normal set is already in $\mathcal{S}$ and the definition of $\mathcal{S}$ requires S_j to be there. This completes the induction since the closure properties assumed for $\mathcal{S}$ imply that the containment still holds.

The proof is now trivial. From the induction it is true that

$$\mathcal{S}_{\varphi_n} \subset \mathcal{S}.$$

Obviously $\varphi = \varphi_n$. This implies that

$$\mathcal{S}_{\varphi} \subset \mathcal{S}$$

which is exactly the containment desired. Equality then holds between the two families.

QED

The notation φS for the S-reduction of φ was chosen to suggest that a set S can be regarded as an operator on formulas. It also implies that more than one set can be used. Thus $\varphi S_1 \cdots S_n$ stands for the formula produced by successive reductions of φ, as in the last lemma. Corollary 1.14 demonstrates that the order in which these reductions are made is unimportant. Combining this with lemma 1.15, each formula φ can be written as the reduction $1S_1 \cdots S_n$, where the sets are precisely the normal sets of φ.

In general it is not necessary to include every normal set. Any subfamily of $\mathcal{S}_{\varphi}$ which reduces the universal set to φ is a generating family for φ. Conversely, a family $\mathcal{S}$ generates the family $\mathcal{S}^*$, the closure of $\mathcal{S}$ with respect to operations on overlapping sets, where $\mathcal{S}^*$ is defined to be the smallest family containing all of $\mathcal{S}$ and closed under union, intersection, and difference of overlapping sets. Uniqueness of $\mathcal{S}_{\varphi}$ implies that $\mathcal{S}^* = \mathcal{S}_{\varphi}$ for any generating family $\mathcal{S}$ of φ.

The second characterization for PQ-formulas solves a problem partially posed by Jennings[Jen1]. Although he did not use the idea of PQ-formulas, he did compute all of the normal sets given a subfamily which generates them. The fact that any family closed under these operations determined a formula if it contained the singletons and Σ was not discussed. Listing the normal sets given a generating family is easy and can be done in a number of steps proportional to the total size of the sets, as will be seen in chapter 2.

For any set $S \in \mathcal{S}^*$, the height of S with respect to $\mathcal{S}$ is the minimum height over all fully parenthesized expressions which compute S using only the union, intersection, and difference of overlapping sets from $\mathcal{S}$. The height of an expression is just the maximum nesting level of its parentheses.

When a formula cannot be reduced it is of interest to characterize the failure in terms of a particular generating family, the obvious choice being the family of sets actually involved in the reductions. The characterization which follows is due to Tucker and appears in [Tuc2] in a slightly different form. The original version will be discussed in chapter 4.

Theorem 1.18 (Tucker):

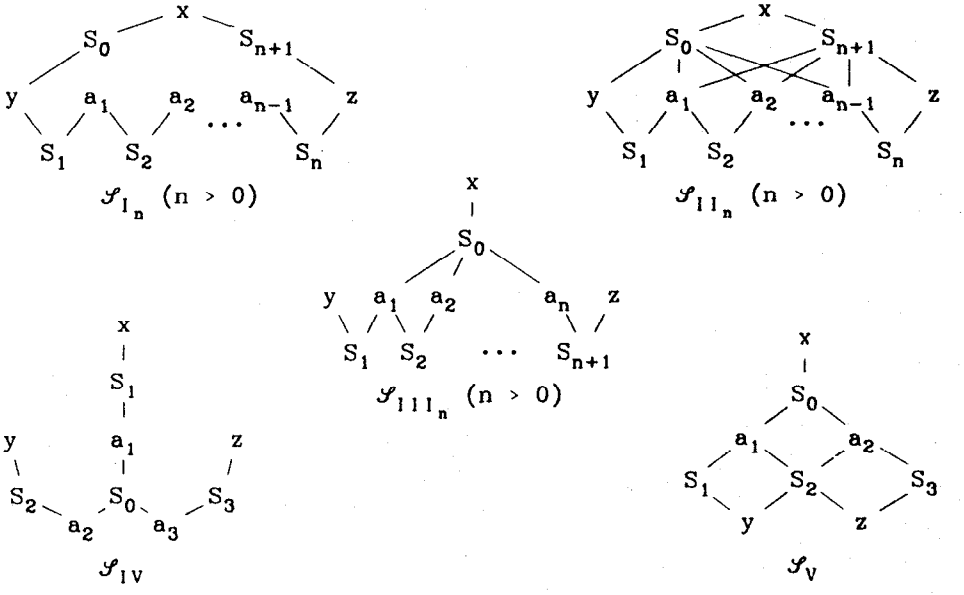
For any family of sets

$$\mathcal{S} = \{ X_1, \cdots, X_k \},$$

the formula

$$\varphi = 1X_1 \cdots X_k$$

is X-reducible iff $\mathcal{S} \cup \{X\}$ contains no subfamily which is one of the following types of families:



(These are the forbidden subfamilies. Sets are denoted by upper case letters and atoms by lower case letters. A line connecting a set with an atom indicates membership. Missing lines imply nonmembership. The atoms designated by x, y, and z are singled out for special attention. They are asteroidal triples. Between any two such atoms there is an alternating sequence of atoms and sets to which the third atom is not adjacent. These will be seen again in later chapters. Note that the first three types are actually family schemata, having a parameter n > 0, while the last two are simple families.)

Proof =>:

By inspection none of the five types of subfamilies is consistent. If the universal formula is reduced by all of the sets within a family except one (let it be X) the resulting formula will not be X-reducible. This can be verified by simply running the reduction algorithm [page 25].

Proof <=:

Suppose that the reduction algorithm fails to X-reduce φ. There are only three ways that this can happen. Each gives rise to an instance of one of the forbidden subfamilies. This will be verified in two stages. The first step will verify that a forbidden subfamily exists within the closure of the family $\mathcal{S} \cup \{X\}$. After this fact has been established, it will be shown that the forbidden subfamily can actually be chosen from the smaller family $\mathcal{S} \cup \{X\}$.

Claim 1:
The reduction algorithm fails to X-reduce φ iff there is an instance of the forbidden subfamilies within the family $(\mathcal{S} \cup \{X\})^*$. This is a weaker condition than the conclusion of the theorem. The forbidden subfamily is being drawn from a much larger family than just the sets used for reduction.

Proof of claim 1:
The claim will be verified by examining the three ways in which the reduction algorithm may fail to X-reduce φ. Since the reduction algorithm itself has no failure outcome, the only failure which can occur is when the formula cannot be labeled by the normalization algorithm [page 18]. This, in turn, only fails if a formula selected at step 2 does not receive a label in step 3. Every atom gets a label, but some compound formula may not receive a label. The following cases account for all of the possibilities which can occur.

Case 1:

A subformula does not get labeled because one of its subformulas has already been labeled as doubly partial. Let the doubly partial subformula be ψ. Define

$$X_\psi = \{ a \mid a \text{ occurs within } \psi \}.$$

Then X and X_ψ overlap because not every pertinent atom is in ψ and not every atom in ψ is pertinent. Let

$$X' = X \cap X_\psi.$$

The atoms of ψ are obviously normal in φ; they constitute a complete subformula. Closure then implies that

$$X_\psi \in (\mathcal{S} \cup \{X\})^*$$

and hence

$$X' \in (\mathcal{S} \cup \{X\})^*.$$

Consider the effect of X'-reducing φ. Because $\varphi_{X'}$ is a subformula of ψ and ψ receives a label, $\varphi_{X'}$ receives a label. This means that φ is X'-reducible. Consider what has happened to ψ after X'-reduction of φ. It is a doubly partial Q-formula and has the form

$$[\psi_1^e \cdots \psi_v^e \psi_1^f \cdots \psi_s^f \psi_{v+1}^e \cdots \psi_r^e]$$

where $1 < v < r$.

Define the following sets, using the atoms from the indicated pieces of ψ.

$$S_0 = X$$

$$S_1 = \{ a \mid a \text{ occurs in } \psi_1^e \cdots \psi_v^e \text{ or } \psi_1^f \cdots \psi_s^f \}$$

$$S_2 = \{ a \mid a \text{ occurs in } \psi_1^f \cdots \psi_s^f \text{ or } \psi_{v+1}^e \cdots \psi_r^e \}$$

The latter two are normal sets for the X′-reduced version of φ. Closure implies that

$$S_1 \in (\mathcal{S} \cup \{X'\})^* \quad \text{and} \quad S_2 \in (\mathcal{S} \cup \{X'\})^*$$

Clearly

$$(\mathcal{S} \cup \{X'\})^* \subset (\mathcal{S} \cup \{X\})^*$$

because X′ is already in the family on the right-hand side. All three sets (S_0, S_1, and S_2) are thus within the closure. A forbidden subfamily can be constructed simply by choosing the appropriate atoms.

Choose x to be any pertinent atom not occuring in ψ, a_1 a pertinent atoms which does occur in ψ, and let y and z be nonpertinent atoms of ψ, one from ψ_l^e and the other from ψ_r^e. This is an instance of $\mathcal{S}_{III_1}$, one of the forbidden subfamilies.

Case 2:

Some P-formula ψ does not get labeled because it has three or more singly partial subformulas. The details are similar to the first case, the result being that for a suitable choice of X′ ⊂ X, the formula ψ looks like

$$(\cdots\psi_1^p\cdots\psi_2^p\cdots\psi_3^p\cdots)$$

after X′-reducing its factors, the indicated subformulas being all singly partial Q-formulas. Define sets S_1, S_2, and S_3 to be the atoms occuring in each of the three Q-formulas and choose x, y, and z to be nonpertinent atoms from the sets, with a_1, a_2, and a_3 pertinent atoms from the sets. S_0 is again X itself. This is an instance of $\mathcal{S}_{IV}$, and as before, all four of the sets are in the closure.

Case 3:

Some Q-formula does not get labeled because all of its pertinent atoms are not consecutive. The same type of argument applies. A set X′ ⊂ X can be chosen. X′-reducing ψ's components gives a Q-formula. It must look like

$$[\cdots\psi_1^f\cdots\psi_1^e\cdots\psi_2^f\cdots]$$

The components superscripted with f are full and the component superscripted with e is empty. Let S_0 be the atoms occurring from ψ_1^f through ψ_1^e and S_1 the atoms occuring from ψ_1^e through ψ_2^f. Picking x, y, and z from the three subformulas gives an instance of $\mathcal{S}_{I_1}$ when S_2 is chosen to be X. As before, all of the sets are contained within the closure. □

Notice that the forbidden subfamilies which were produced were, in each case, quite simple. In the first and third cases they were the smallest family within a schemata of families. The second was one of the simple families. Two of the types did not occur at all! This will be rectified in the next part of the proof. All of the indicated families <u>are</u> actually necessary in order to guarantee that the forbidden subfamily occurs completely within the sets involved in the reductions. This will be the second part of the proof; it is considerably more complicated than the first part because of the large case analysis involved.

<u>Claim 2:</u>
A forbidden subfamily exists within $(\mathcal{S} \cup \{X\})^*$ only if a forbidden subfamily exists within $\mathcal{S} \cup \{X\}$. This will complete the theorem, since it has already been demonstrated that a forbidden subfamily exists within the second family iff the reduction algorithm fails to X-reduce φ.

<u>Proof of claim 2:</u>
The proof is by induction on h, the maximum height of a set within an instance of the forbidden subfamilies.

<u>Assertion:</u>
If $(\mathcal{S} \cup \{X\})^*$ has a forbidden subfamily of maximum height h, then $\mathcal{S} \cup \{X\}$ has a forbidden subfamily.

<u>Basis:</u>
Let h = 0. Every set of the subfamily is actually in $\mathcal{S} \cup \{X\}$.

This is the desired state of affairs.

Induction: Let $h > 0$. The proof is a case-by-case analysis. Most of these are quite similar, so only one will be done in detail. Assume, then, that the forbidden subfamily is an instance of $\mathcal{I}_{I_n}$. Referring back to the diagram in the statement of the theorem, without loss of generality, suppose that S_1 is one of the sets of maximum height within the forbidden subfamily. Its height is h. There are a number of cases to consider.

Case 1:
$S_1 = S \cup T$, where S and T are overlapping sets of height no more than h-1. If $\{y,a_1\} \subset S$ or $\{y,a_1\} \subset T$ the appropriate set can be substituted for S_1 in the forbidden family. This yields another instance of $\mathcal{I}_{I_n}$.

Otherwise choose a new element w which is in both S and T. If it is in none of the other S_j, add both S and T plus w to the forbidden family in place of S_1. This will be an instance of $\mathcal{I}_{I_{n+1}}$.

If w is an atom from one of the sets other than S or T, there are still three more possibilities to explore. If w is not in one of the other sets, say S_j, then an instance of $\mathcal{I}_{I_p}$ is formed (for some $p > 0$) by considering the two closest sets on either side of S_j of which w is a member (they must exist because S and T can fill this role) plus all of the sets which occur on the path connecting the two and passing through S_j.

The last two possibilities occur when w is in every one of the sets. A special case is for $n = 1$. This gives rise to an instance of $\mathcal{I}_{II_1}$ when all of the atoms and sets except S_1 are used. If $n > 1$ then an instance of $\mathcal{I}_{IV}$ is easily constructed using S_0, S_{n+1}, and T with the atoms x, z, w, and a.

Case 2:
$S_1 = S \cap T$, where S and T are overlapping sets of height no more than h-1. Again there are a number of subcases, but each can be

easily verified. If S or T cannot be directly substituted for S_1 then an instance of some other forbidden family may occur.

Case 3:

$S_1 = S - T$, where S and T are overlapping sets of height no more than h-1. Again either a direct substitution of S for S_1 or else some similar modification will result in a forbidden subfamily.

Obviously these first three cases cover only some of the situations. They are in fact the simplest ones. The complete symmetry of $\mathcal{S}_{I_n}$ means that only S_1 has to be considered. The other families have many more cases, each with three subcases, depending upon whether the selected set is a union, intersection, or difference. But in every case a set of maximum height can be eliminated from the forbidden family while still leaving another forbidden family. This process can be repeated until the maximum height is less than h, at which point the induction hypothesis applies. □

QED

IV A Lattice

There are two different ways to describe a formula: its underlying arrangements and its normal sets. The next lemmas use the two characterizations to generate new formulas from old ones.

Lemma 1.19:

If φ_1 and φ_2 are two formulas, then $\mathcal{A}_{\varphi_1} \cap \mathcal{A}_{\varphi_2}$ is the collection of underlying arrangements for some PQ-formula.

Proof:

Assuming that $S_1, \cdots, S_r$ are the normal sets for φ_1 and that $T_1, \cdots, T_s$ are the normal sets for φ_2, the previous discussion asserts that

$$\varphi_1 = S_1 \cdots S_r \quad \text{and} \quad \varphi_2 = T_1 \cdots T_s.$$

Define

$$\varphi_1 \wedge \varphi_2 = S_1 \cdots S_r T_1 \cdots T_s .$$

By lemma 1.13 this has precisely the set of underlying arrangements in which all of the sets are consecutive so

$$\mathcal{A}_{\varphi_1} \cap \mathcal{A}_{\varphi_2} \subset \mathcal{A}_{\varphi_1 \wedge \varphi_2} .$$

The opposite containment also holds since every underlying arrangement for $\varphi_1 \wedge \varphi_2$ is valid for both φ_1 and φ_2. Thus

$$\mathcal{A}_{\varphi_1} \supset \mathcal{A}_{\varphi_1 \wedge \varphi_2} \quad \text{and} \quad \mathcal{A}_{\varphi_2} \supset \mathcal{A}_{\varphi_1 \wedge \varphi_2} .$$

The containments carry over to the intersection of the collections, giving

$$\mathcal{A}_{\varphi_1} \cap \mathcal{A}_{\varphi_2} \supset \mathcal{A}_{\varphi_1 \wedge \varphi_2}$$

and thus the two collections must be equal.

QED

The formula $\varphi_1 \wedge \varphi_2$ is called the meet of the formulas φ_1 and φ_2. The meet is a generalization of the reduction operation. It is the mutual reduction of two formulas. Another binary operator can be defined in an analogous manner using the characterization by normal sets.

Lemma 1.20:

If φ_1 and φ_2 are two formulas, then $\mathcal{S}_{\varphi_1} \cap \mathcal{S}_{\varphi_2}$ is the family of normal sets for some PQ-formula.

Proof:

The closure properties carry over to the intersection of the two families. Both Σ and all of its singletons are still in the intersection. Lemma 1.17 then applies, guaranteeing that a unique PQ-formula is defined.

QED

Let $\varphi_1 \vee \varphi_2$ be the formula which corresponds to the intersection of two families. The formula $\varphi_1 \vee \varphi_2$ is called the join of the two formulas φ_1 and φ_2. Together with the meet it forms an algebraic system over the equivalence classes. The meet is a direct extension of reduction. The resulting collection of arrangements is always at least as small as either of the original two. The join, on the other hand, acts to increase the number of underlying arrangements. The two operators are thus somewhat opposites. They are not inverses because

$$(\varphi_1 \wedge \varphi_2) \vee \varphi_2 \neq \varphi_1$$

for arbitrary φ_1 and φ_2, but it is true that

$$(\varphi_1 \wedge \varphi_2) \vee \varphi_2 = \varphi_2.$$

With this final hint, the algebraic properties should not be hard to guess, although the names meet and join have surely spoiled the suspense anyway. Given the alphabet Σ, let $\mathcal{P}_\Sigma$ denote the collection of equivalence classes of PQ-formulas over Σ. As the names of the operators suggest, the PQ-formulas form a lattice. The proof is a simple, although somewhat lengthy exercise in verifying the algebraic definitions.

The proof which follows is not the only possible way to achieve this result. The duality between underlying arrangements and normal sets implies that either can be used to obtain isomorphic lattices. Using just the normal sets and the usual notion of algebraic closure for the operations of lemma 1.16, an alternate proof can be given. The corresponding proof using arrangements is somewhat trickier, since the operations under which closure is to be obtained are a little obscure. The proof here uses a hybrid approach, in order to exercise some of the machinery developed so far.

Theorem 1.21:

The algebra $(\mathcal{P}_\Sigma, \wedge, \vee)$ is a lattice with maximum element 1 and minimum element 0.

Proof:

Consider the relation $\prec$ on PQ-formulas defined as

$$\varphi_1 \prec \varphi_2$$

iff both

$$\varphi_1 \wedge \varphi_2 = \varphi_1 \qquad \text{and} \qquad \varphi_1 \vee \varphi_2 = \varphi_2.$$

It must be shown that this is a partial order on $\mathcal{P}_\Sigma$ and that the meet and join are, respectively, a greatest lower bound and a least upper bound.

Claim 1:
The relation $\prec$ is a partial ordering on the equivalence classes $\mathcal{P}_\Sigma$.

Proof of claim 1:
The definitions of $\wedge$ and $\vee$ can be used to define $\prec$ directly in terms of underlying arrangements and normal sets. Simple substitution shows that

$$\varphi_1 \wedge \varphi_2 = \varphi_1 \qquad \text{and} \qquad \varphi_1 \vee \varphi_2 = \varphi_2$$

iff

$$\mathcal{A}_{\varphi_1} \cap \mathcal{A}_{\varphi_2} = \mathcal{A}_{\varphi_1} \qquad \text{and} \qquad \mathcal{S}_{\varphi_1} \cap \mathcal{S}_{\varphi_2} = \mathcal{S}_{\varphi_2}$$

iff

$$\mathcal{A}_{\varphi_1} \subset \mathcal{A}_{\varphi_2} \qquad \text{and} \qquad \mathcal{S}_{\varphi_1} \supset \mathcal{S}_{\varphi_2}.$$

From this it is clear that the partial ordering of set containment on each of the classes

$$\left\{ \mathcal{A}_\varphi \mid \varphi \in \mathcal{P}_\Sigma \right\} \qquad \text{and} \qquad \left\{ \mathcal{S}_\varphi \mid \varphi \in \mathcal{P}_\Sigma \right\}$$

induces a partial ordering on equivalence classes and that $\prec$ is the intersection of these two relations. This is sufficient to guarantee that $\prec$ is a partial order. □

Claim 2:
The meet is a greatest lower bound. (The proof that the join is a least upper bound is similar and is omitted.)

Proof of claim 2:
From the definition of the meet

$$\mathcal{A}_{\varphi_1 \wedge \varphi_2} = \mathcal{A}_{\varphi_1} \cap \mathcal{A}_{\varphi_2}$$

so both

$$\mathcal{A}_{\varphi_1 \wedge \varphi_2} \subset \mathcal{A}_{\varphi_1} \quad \text{and} \quad \mathcal{A}_{\varphi_1 \wedge \varphi_2} \subset \mathcal{A}_{\varphi_2}.$$

Lemma 1.19 says that the normal sets of φ_1 and φ_2, taken together, generate the normal sets for the meet. Using the closure properties given in lemma 1.9

$$\mathcal{S}_{\varphi_1 \wedge \varphi_2} \supset \mathcal{S}_{\varphi_1} \cup \mathcal{S}_{\varphi_2}.$$

The containment holds separately for each of the two families, so

$$\mathcal{S}_{\varphi_1 \wedge \varphi_2} \supset \mathcal{S}_{\varphi_1} \quad \text{and} \quad \mathcal{S}_{\varphi_1 \wedge \varphi_2} \supset \mathcal{S}_{\varphi_2}.$$

This last set of containments, along with the previous ones for the collections of underlying arrangements, are exactly the conditions needed to show that

$$\varphi_1 \wedge \varphi_2 \prec \varphi_1 \quad \text{and} \quad \varphi_1 \wedge \varphi_2 \prec \varphi_2$$

so the meet is a lower bound for φ_1 and φ_2. This establishes one half of the claim. It still must be shown that it is a greatest lower bound. Suppose that ψ is any other lower bound for φ_1 and φ_2. By definition,

$$\psi \prec \varphi_1 \quad \text{and} \quad \psi \prec \varphi_2$$

and hence, by an argument similar to the one before,

$$\mathcal{A}_\psi \subset \mathcal{A}_{\varphi_1} \quad \text{and} \quad \mathcal{A}_\psi \subset \mathcal{A}_{\varphi_2}$$

and also

$$\mathcal{S}_\psi \supset \mathcal{S}_{\varphi_1} \quad \text{and} \quad \mathcal{S}_\psi \supset \mathcal{S}_{\varphi_2}.$$

These can be combined to yield the two containments

$$\mathcal{A}_\psi \subset \mathcal{A}_{\varphi_1} \cap \mathcal{A}_{\varphi_2} \quad \text{and} \quad \mathcal{S}_\psi \supset \mathcal{S}_{\varphi_1} \cup \mathcal{S}_{\varphi_2}.$$

From the definition of the meet

$$\mathcal{A}_{\varphi_1 \wedge \varphi_2} = \mathcal{A}_{\varphi_1} \cap \mathcal{A}_{\varphi_2}.$$

The closure properties of normal sets then imply that

$$\mathcal{S}_\psi \supset \mathcal{S}_{\varphi_1 \wedge \varphi_2} \supset \mathcal{S}_{\varphi_1} \cup \mathcal{S}_{\varphi_2}.$$

These last two are sufficient to show that

$$\varphi_1 \wedge \varphi_2 \prec \psi$$

and hence that the meet is indeed a greatest lower bound. □

This establishes that $(\mathcal{P}, \wedge, \vee)$ is a lattice. All that remains to be proven is the maximum and minimum. These are easy.

Claim 3:
1 is a maximum in the lattice and 0 is a minimum.

Proof of claim 3:
By definition the null formula has no underlying arrangements and every set is trivially normal. Symmetrically, every arrangement is acceptable for the universal formula but only A and its singletons are normal. Thus

$$\mathcal{A}_0 \subset \mathcal{A}_\varphi \subset \mathcal{A}_1 \quad \text{and} \quad \mathcal{S}_0 \supset \mathcal{S}_\varphi \supset \mathcal{S}_1$$

or, in terms of the partial order,

$$0 \prec \varphi \prec 1$$

Thus 0 and 1 are a minimum and a maximum in the lattice of equivalence classes. □

The algebra $(\mathcal{P}_\Sigma, \wedge, \vee)$ has been shown to be a lattice, with the required maximum and minimum elements.

QED

The lattice is not a Boolean algebra. This is easy to establish using simple counting arguments. Even worse, it is neither distributive nor complemented, both required properties for a Boolean algebra. Actually computing meets and joins is beyond this discussion, although the next chapter will discuss how an algorithm for finding the meet or join of two formulas can be constructed from the reduction algorithm. Perhaps a more interesting pursuit would be to count the number of equivalence classes over an alphabet Σ of a specific size. Here, the lattice properties might be of some use.

Beginning with the definition of PQ-formulas as bracketed arrangements, an algebraic lattice has been described which consists of equivalence classes of formulas. The meet operation corresponds to a generalization of the reduction of a formula with respect to a set. Using the lattice properties, equivalence classes can be represented by either their underlying arrangements or their normal sets. Despite the mathematical lure of this theory the topic of primary interest is the existence of an algorithm for reducing a formula. This reduction is accomplished by first normalizing the formula and then recursively replacing its constituents with reduced formulas, so that the given set is constrained to occur consecutively.

The normalization and reduction algorithms [pages 18 and 25] can be used directly on PQ-formulas. An approach similar to this was taken in [Lem1]. Finding the appropriate normalization requires looking at all of the atoms in a formula. This means that each reduction takes at

least O(m) steps even if |S| << m. A more efficient implementation requires a good data structure. The next chapter introduces a means of representing PQ-formulas which allows both the normalization and reduction operations to be performed easily. The algorithms in later chapters which employ them as primitives can be implemented to run in linear time.

2 Trees

The next step is to devise a good data structure for computing the *reduction of* a PQ-*formula*. *Using trees to represent formulas is not* a new idea. It is standard operating procedure in many areas of computer science such as theorem-proving, algebraic symbol manipulation, natural language processing, and program compilation. In this chapter a new variety of trees is introduced. They are used to represent the formulas discussed in the previous chapter, and are called PQ-trees. Transformations on PQ-trees which correspond to operations on PQ-formulas are defined here and algorithms for computing $1S_1 \cdots S_n$ and $\varphi_1 \wedge \varphi_2$ are described. The first algorithm runs in time which is a linear function of the sum of the sizes of the sets; the second is O(m) when $\Sigma = \{a_1, \cdots, a_m\}$.

I Trees And Formulas

A PQ-formula can be converted into an oriented PQ-tree in much the same way that an arithmetic expression is converted to a tree. The atoms of a formula correspond to leaves of the tree and each compound formula becomes an internal node. P-nodes correspond to products, and Q-nodes correspond to concatenates. The children of an internal node are the constituents of its formula. The node for the enclosing formula is a

parent, and nodes having a common parent are siblings. Descendant and ancestor have the usual meaning. Trees are drawn in the normal way, children below parent, and have a definite left-to-right orientation among the children. A leaf is drawn as the atom itself (a letter), P-nodes as circles, and Q-nodes as rectangles. A subtree is any connected set of nodes within a tree. A subtree need not itself satisfy the conditions required of a PQ-tree.

Just as chapter 1 concentrated on proper PQ-formulas, a proper PQ-tree can be defined, and will be the object of attention here. Each P-node must have at least two children and each Q-node at least three. Every atom is assumed to occur exactly once as a leaf. Mapping a PQ-formula to its tree is simply a matter of replacing each compound subformula by a P-node or Q-node having the appropriate subtrees. The inverse mapping is also easy and is achieved by a walk of the tree. This is summarized by the next lemma.

Lemma 2.1:

There is a 1-1 correspondence between oriented PQ-trees and PQ-formulas.

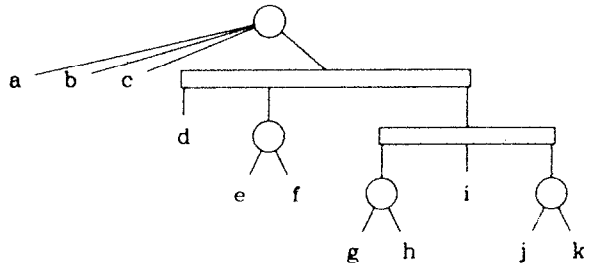
The conversion is quite mechanical so no proof is supplied. Instead, the following example is offered by way of illustration.

Example 2.2:

The formula used as an example throughout chapter 1,

(abc[d(ef)[(gh)i(jk)]])

has the following unoriented PQ-tree.



Each node in the tree corresponds to one of the subformulas given in example 1.7.

All of the results for PQ-formulas carry over to PQ-trees under this correspondence. In particular, the concept of reducibility with respect to a set is still present. The normalization and reduction algorithms could be restated for oriented PQ-trees, but the great advantage of the tree representation is that this need not be done. The equivalence classes are directly accessible without normalization, if the orientation is handled correctly.

Not surprisingly, normalization turns out to be the bottleneck in any implementation where formulas are treated as strings. Eliminating the need to normalize leads to a faster version of the algorithm. The problem is to somehow ignore the orientation of the tree where it gets in the way of efficiency. After all, most, if not all of the orientation should be superfluous, since it is an artifact left over from the linearity of the string representation. Orientation can be largely ignored. This is achieved by relaxing some of the conditions on a tree.

For an internal node, the left-to-right orientation of its children is exactly the order in which constituents occur in its formula. A product is equivalent to any permutation of its factors, so there is no reason to orient the children. An unoriented P-node represents the entire equivalence class of products. The unoriented version is always normalized, in the sense that the representation does not favor any special orientation over all the others. If the same is to hold for

Q-nodes, some allowance must be made for the fact that only a restricted set of transformations can be used on a concatenate — just reversals.

In this case orientation cannot be done away with entirely. The children of a Q-node must have a definite sibling relationship. The key idea is that no absolute left or right is needed. Each child is either endmost or interior, with the latter always having the same two immediate siblings. This mirrors the fact that the order of the components within a concatenate can be reversed, but not arbitrarily permuted, in contrast to the factors within a product. These more general Q-nodes are called unoriented Q-nodes.

An unoriented PQ-tree is composed of leaves and unoriented internal nodes. It corresponds to an entire equivalence class of oriented PQ-trees, if two oriented trees are considered equivalent whenever a sequence of permutations of P-nodes or reversals of Q-nodes transforms one tree into the other. The null tree is easily drawn; it has no nodes at all. The universal tree is a single P-node having exactly one leaf for each atom in the alphabet Σ. These are the unoriented PQ-trees which correspond to the null formula and the universal formula.

The analogy with formulas is now complete. Throughout the remainder of this chapter the term formula will be used to stand interchangeably for a single formula and also its equivalence class. The term tree will always refer to the unoriented tree of a formula.

Lemma 2.3·

There is a 1-1 correspondence between unoriented PQ-trees and equivalence classes of PQ-formulas.

Let $\mathcal{T}$ be a PQ-tree. Definitions from its corresponding formula carry over in an obvious way. Given a set S, a leaf is pertinent if it is a member of S and an internal node is pertinent if one of its descendants is pertinent. The pertinent subtree is the subtree of $\mathcal{T}$ which corresponds to the pertinent subformula.

The tree representation has some additional structure which can be used to great advantage. Not every subtree within the pertinent subtree is pertinent. This was also true for formulas, but there was no easy way to exploit the fact. An algorithm using trees benefits by ignoring the subtrees of $\mathfrak{T}$ which are not pertinent. Excluding the nonpertinent nodes, the part of $\mathfrak{T}$ which remains is called the pruned pertinent subtree. Notice that this is always a subtree of the pertinent subtree.

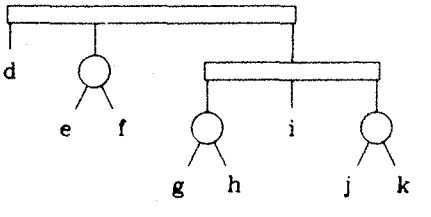
The entire pertinent subtree is usually not interesting, so the notation $\mathfrak{T}_S$ is reserved for the pruned version. It will not always be possible to restrict attention just to $\mathfrak{T}_S$, however, so two final pieces of $\mathfrak{T}$ are singled out. The path from the root of $\mathfrak{T}_S$ to the root of $\mathfrak{T}$ is called the handle of $\mathfrak{T}_S$. It may be the null path. The extended handle is the handle plus a sequence of imaginary nodes which are added as the ancestors of the root of $\mathfrak{T}$. The last piece isn't really part of the tree at all. The algorithms below do not actually operate in the extended handle, but it is a convenient descriptive device and simplifies some of the arguments used in later proofs. The height of a tree $\mathfrak{T}$ is l^*, the longest leaf-to-root distance in $\mathfrak{T}$.

Example 2.4:

Again using the same example as before, the pertinent subtree with its handle is given by the following PQ-tree. The pertinent subtree has height 3; the entire tree has height 4.

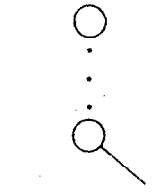
handle:

pertinent subtree:

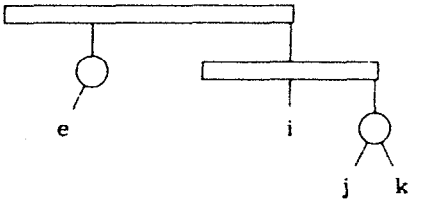


The pruned pertinent subtree is smaller than this. It and its extended handle are indicated below.

handle:



pruned subtree:



Note that this is not actually a PQ-tree, according to the definitions.

The process of reducing a formula with respect to a set S is now greatly simplified. Normalization no longer is an issue. The result of an S-reduction will again be an unoriented tree, since lemma 1.12 shows that the reductions form an equivalence class and all have a common tree. This makes the algorithm particularly easy to state. The following version specifies a set of templates for replacing subtrees during reduction. Again the notation for formulas carries over to the tree. Nodes are classified as empty, full, singly partial, or doubly partial. It is easy to check that this describes the same procedure as the reduction algorithm [page 25] which was stated in chapter 1, this time given as an algorithm on PQ-trees.

Tree Reduction

[1] Let $\mathcal{T}$ be a PQ-tree and S a subset of its leaves. Assume that each node of $\mathcal{T}$ is initially unlabeled. Place each leaf on the queue to be reduced.

[2] Remove the node n from the head of the queue. It is the next candidate for reduction.

[3] Label n as in step 3 of the normalization algorithm [page 18], selecting a case according to the labels on its children. Replace it in $\mathcal{T}$ using the appropriate template below. If none of these templates apply, set $\mathcal{T}$ to be the null tree and halt with the answer "irreducible."

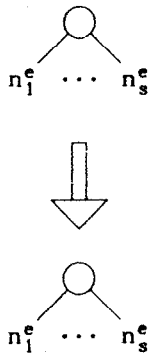
Case 1:

When n is a leaf, there is never anything to do. Leave it alone.

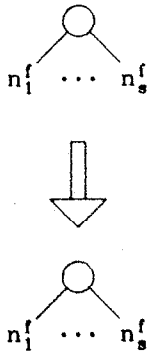
Case 2:

When n is a P-node, there are the usual four subcases.

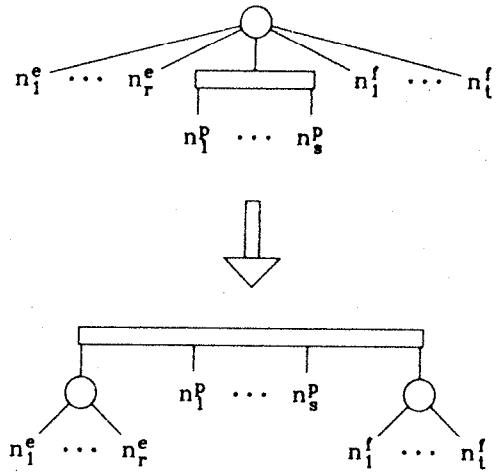
a) If n is empty, leave it alone.



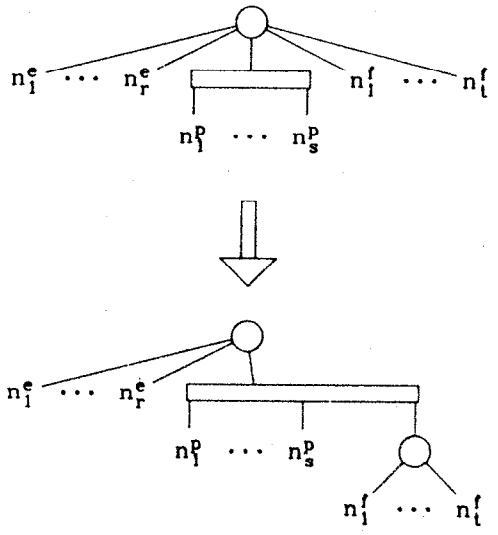
b) If n is full, leave it alone.



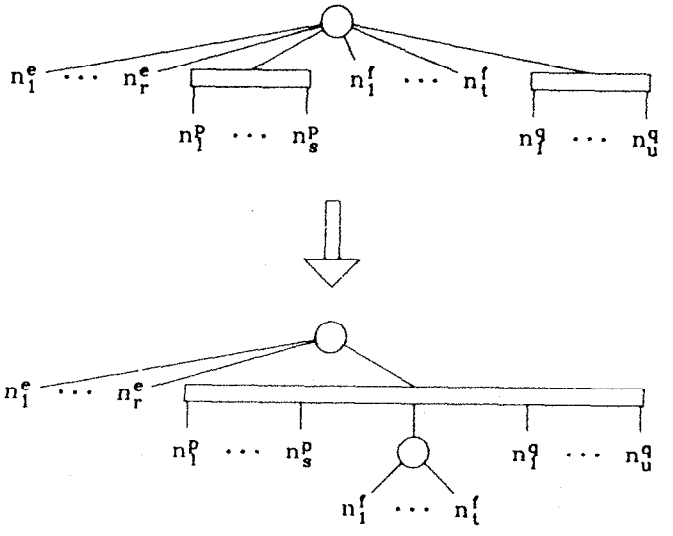
c) If n is singly partial, replace it with a Q-node.



except for the special case when n is the root of $\mathfrak{T}_S$. Then use a P-node instead.



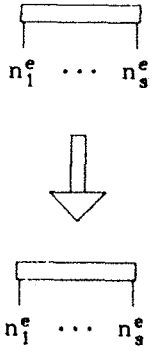
d) If n is doubly partial, replace it with a P-node.



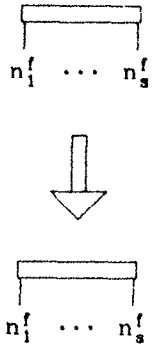
Case 3:

When n is a Q-node, there are also four subcases.

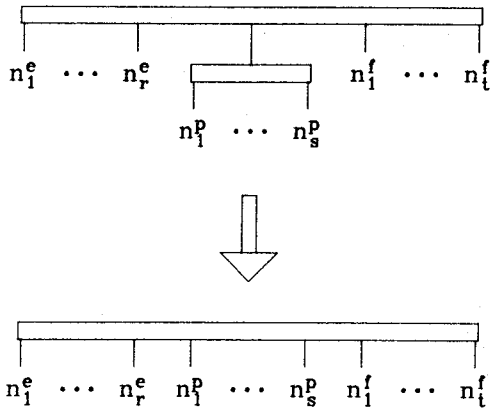
a) If n is empty, leave it alone.



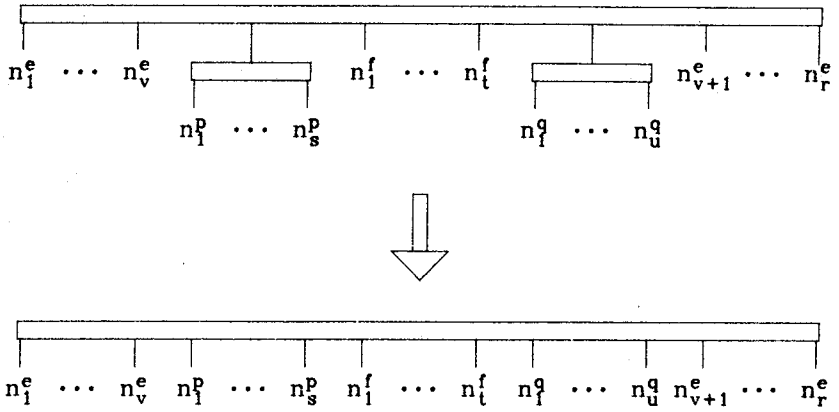
b) If n is full, leave it alone.



c) If n is singly partial, replace it with a Q-node.



d) If n is doubly partial, replace it with a Q-node.



[4] If n was the last unlabeled child of some other node within the pertinent subtree append the parent node to the tail of the queue.

[5] Go back to step 2 if the queue is not empty, otherwise halt.

Because of the duality between trees and formulas, it is obvious that the reduced tree is the same as the tree of the reduced formula. Another important property which is also preserved by duality is the labeling, even after reduction.

Lemma 2.5:

Every node in a reduced tree is either empty or full, except possibly the root of $\mathfrak{T}_S$. If it is a Q-node, it may be partial, but all of its full children must be adjacent.

Proof:

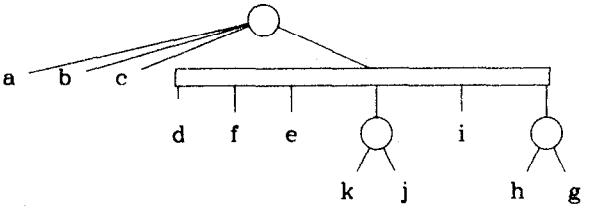
The proof is a repeat of the proof for lemma 1.11, with formulas replaced by trees.

QED

The reduction of a tree is a node-by-node process. Each node is reduced according to information about its immediate subtrees. The sequencing of operations is important! Nodes may be processed only in an order consistent with the requirement that each internal node be processed after all of its children. One acceptable ordering is postorder, although there are obviously others. The order used here is determined entirely by the order in which atoms are first placed into the queue. The first-in, first-out discipline of the queue does the rest.

Example 2.6:

Reducing the tree from example 2.4 yields a new PQ-tree. As it should be, this is the unoriented tree for the reduced formula obtained in example 1.10.



The height of the tree has been reduced by one. All of the nodes, except for the root of the pertinent subtree, are either empty or full.

The work at each node is fairly easy, provided the proper housekeeping details are attended to, and is proportional only to the number of children below the node. This means that the reduction algorithm runs in time proportional to the size of the tree. Denote the size of a tree, its total number of nodes, by $|\mathfrak{T}|$. A proper tree is always at least binary, so the size is clearly O(m). Unfortunately the algorithm may be very inefficient if the size of S is quite small when compared to m. The fact that only the pertinent subtree need be processed doesn't help. It is easy to find examples where this is all of $\mathfrak{T}$, even though S has only two elements.

Notice that the only nodes which are actually changed during tree reduction are those corresponding to partial formulas. It is this fact which enables an algorithm to ignore much of $\mathfrak{T}$. This rather obvious property is stated in the next lemma.

Lemma 2.7:

The only nodes in $\mathfrak{T}$ which are changed by the reduction algorithm are the partial nodes in $\mathfrak{T}_S$.

Proof:

This is immediate from the definition of singly and doubly partial. Either is automatically in $\mathfrak{T}_S$. Since these are the only nodes which are modified the proof is complete.

QED

Ideally, the size of $\mathfrak{T}_S$ would be related in a nice way, hopefully linear, to the size of S. Life is not so easy. Performing a reduction with respect to S may require looking at almost all of $\mathfrak{T}$. Just identifying $\mathfrak{T}_S$ is difficult, and will generally take more than a linear number of steps in the size of S.

II Efficient Implementation

Although no linear solution seems to exist for a single reduction, there is a scheme which will turn out to be linear for a series of reductions. Both of the applications for the reduction algorithm actually use it in this manner, so this is a good consolation prize. Making a number of reductions on the same tree allows the overhead to be averaged out over all of the sets.

The first step toward an efficient algorithm is to identify $\mathfrak{T}_S$ so that only pertinent nodes need to be processed. This can be accomplished using a bottom-up (leaves toward the root) strategy. The root of $\mathfrak{T}_S$ has been found when all of the pertinent leaves have been accounted for as descendants. A naive approach would simply execute the reduction algorithm, bottom-up, beginning with the pertinent leaves.

The problem is that if not all the leaves are to be processed, it is unclear at a general step when an internal node is ready to be processed. All of the pertinent subtrees must have been reduced, or the labeling may not be correct. Waiting until all subtrees have been reduced will not solve the problem. Some subtrees, the nonpertinent ones, will never be processed! A compromise strategy is needed, one which doesn't wait forever, but one which also insures that the child-before-parent rule is enforced. Solving this "halting problem" is the key to achieving a fast algorithm. During reduction, the algorithm must know when to wait for results which will be computed, and, just as important, it must know when to continue on, ignoring those results which will not be needed.

One solution is to use a two-pass approach, first identifying the nodes to be processed and then, after the information necessary to label nodes has been collected, performing the reductions. This second step is easy, and is exactly as before. The harder part is the first pass, which must identify $\mathfrak{T}_S$ without doing too much extra work. This is accomplished using a bubbling-up process.

Bubble-Up I

[1] Place each leaf $a \in S$ of $\mathfrak{T}$ on the queue to be marked.

[2] Remove a node n from the front of the queue. It is the next candidate to be marked.

[3] Increment the pertinent child count by one for the parent of n.

[4] If n is the child of some other node in $\mathfrak{T}$ place the parent on the queue if the parent is unmarked and unqueued.

[5] Go back to step 2 if the queue has more than one entry. Halt otherwise.

Bubble-up will usually mark more than just the partial nodes of $\mathfrak{T}$, but only nodes in $\mathfrak{T}_S$ or its handle will be looked at. It may travel quite far up the handle before it finally stops. It is easiest to imagine this bubbling-up taking place on the extended handle. All nodes are then treated equally. Actual implementations will treat the root of $\mathfrak{T}$ as a special case, stopping the bubble-up if it attempts to pass the root. The following theorem places an upper bound on how far up the handle this process can continue.

Lemma 2.8:

A node is marked by the bubble-up algorithm only if it is in $\mathfrak{T}_S$ or it is a node in the extended handle which is no further than l^* from the root of $\mathfrak{T}_S$.

Proof:

By definition, all of the pertinent leaves are in $\mathfrak{T}_S$. Thus the only nodes which can be marked which are not in the pertinent subtree are internal nodes. These must have at least one child marked so they must be in the handle. Furthermore, no such node is marked until after the root of $\mathfrak{T}_S$ is marked.

When the root is marked its parent in the handle is placed on the queue. The parent cannot be marked until all of the nodes already on the queue have been processed. Similarly its parent will not reach the head of the queue until all of the other nodes placed on the queue after the root's parent and before the grandparent. The obvious induction is omitted.

This is sufficient to show that the number of such nodes in the handle is no more than the longest sequence of nodes in $\mathcal{S}_S$, since once the queue contains only one node, the algorithm halts.

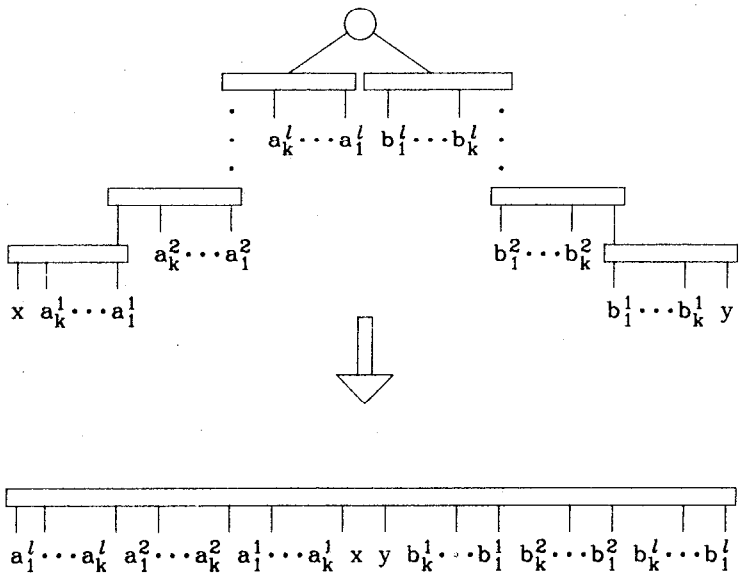
QED

Bubble-up solves the problem of a first pass. It only looks at the nodes which have to be reduced, plus a few extra ones, guaranteed to be no more than a factor of two too many. Everything is great, except that now the second pass is no longer easy. It has a lot more work to do. Too much work! In order for the bubbling-up process to occur at all, each node must have a pointer to its parent in the tree, something which might not have been required before. Maintaining these pointers is quite a chore. The reduction phase must update the pointers every time a node receives a new parent; if the pointers are not updated, subsequent reductions will not have the correct information.

Consider the case in which a sequence of nested partial Q-nodes occurs. During the reduction phase each child of the lowest Q-node will have as a parent every one of the Q-nodes above it. Obviously not every one of these changes has to be made, just the last one. Even so the number of children for the eliminated Q-nodes may be much, much larger than the size of S. Something has to be done to avoid processing nonpertinent nodes in this manner.

Example 2.9:

The following tree is particularly bad. It is its own pertinent subtree when $S = \{x,y\}$. Not only that, reduction requires that almost every parent pointer be changed.



Updating these parent pointers is one of the major obstacles.

There is a way to reduce the amount of work required to maintain the parent pointers. Children of the same Q-node can be placed in a sibling class. Tritter's algorithm for disjoint set union[Aho1] can then be used to keep track of the pointers, maintaining only one for each sibling class. Eventually, being nitpicky about details, this will reduce the total running time to $O(n\alpha(n,n))$, where $\alpha(n,n)$ is the slowly growing, "almost constant" function discussed in [Tar1]. Possibly this bound can be lowered still more, although Tarjan showed that for the general disjoint set union algorithm the bound is tight. But this approach is presented here only as a straw man. There is a better way.

A slight modification to bubble-up will work without necessitating any of the troublesome parent pointers. Specifically, the parent pointers

can be ignored altogether for the interior children of Q-nodes, although pointers are still needed from all of the children of P-nodes and from the endmost children of Q-nodes. This idea is due to Lueker and appears in [Bth1] and [Lue2]. Lueker's original formulation of PQ-trees uses only Q-nodes of this type, exactly for this reason.

Removing these parent pointers means that bubbling-up cannot occur from an interior child of a Q-node. The process will be temporarily blocked at that node. Before the process can continue, one of the endmost children must be reached, causing the process to become unblocked again. Keeping track of the blocking and unblocking requires some additional work, but the results are worth the added expense. If the tree is reducible, $\mathfrak{T}_S$ can be identified in what turns out to be not much time at all.

As the algorithm progresses, a node can be in one of four states: unmarked, queued, blocked, or unblocked. The blocked nodes will be interior children of Q-nodes whose parent pointer cannot be determined until after one of the immediate siblings has been processed. The node becomes unblocked if one of the endmost children is processed and the parent pointer can be propagated across a chain of pertinent siblings. A blocked counter is maintained to keep track of these nodes. The blocked count is the number of consecutive blocks of blocked nodes, not simply the number of blocked nodes. Thus three adjacent blocked nodes count as one block. This may sound a bit confusing, but it should be clearer after a detailed explanation of the bubble-up.

Bubble-Up II

[1] Assume that each node is initially unmarked. Place every leaf $a \in S$ on a queue and mark it pertinent. Set the blocked count to zero.

[2] Remove a node n from the front of the queue to be processed.

[3] Select the appropriate case below and perform the indicated operations. Exactly one case will always apply.

Case 1:

If n is the child of a P-node mark n unblocked and increment its parent's pertinent child count.

Case 2:

If n is an interior child of a Q-node, examine its two immediate siblings and select the appropriate subcase below, depending upon how the siblings are marked.

a) Neither sibling is marked. Mark n blocked and increment the blocked count by one.

b) Only one sibling is marked. Mark n the same. Increment the parent's pertinent child count if n is marked unblocked. The blocked count remains the unchanged.

c) Both siblings are marked blocked. Mark n blocked and decrement the blocked count by one.

d) Both siblings are marked unblocked. Mark n unblocked. The blocked count remains unchanged.

e) The two siblings are marked differently. Beginning with n and moving left or right as appropriate, mark unblocked n and every sibling encountered until finding one which is not marked blocked. Increment the parent's pertinent child count once for each interior node which is changed to unblocked. Decrement the blocked count by one.

Case 3:

If n is an endmost child of a Q-node mark it unblocked, increment its parent's pertinent child count. If its sibling is marked blocked, mark the sibling and all adjacent blocked siblings unblocked, then decrement the blocked counter by one. The blocked counter remains unchanged otherwise. Increment the parent's pertinent child count once for each interior node which is changed to unblocked.

[4] If n was not the interior child of a Q-node append its parent to the tail of the queue if the parent is unmarked and unqueued.

[5] Go back to step 2 unless the queue is empty or it has exactly one entry and the blocked counter is zero. Otherwise halt with the message "irreducible" if the blocked count is greater than one and "reducible" if it is no greater than one, creating a pseudonode, whose children are all the blocked nodes, when the

blocked count is exactly one.

A few points have been glossed over. Each time a node was marked unblocked the pertinent child count was incremented for the parent; even if the node was interior. This is all right, though, because the pointer to the parent was only used after the child was marked unblocked. If the correct parent pointer is propagated from the endmost child as each interior node is marked unblocked, all of them will have a valid parent pointer when the second reduction pass is made. The pertinent child count is maintained precisely so that the reduction pass will be properly sequenced.

The pseudonode is a slight trick. It is created only if the blocked count is still one when the queue empties. This happens when the root of the pertinent subtree is a doubly partial Q-node. A pseudonode must be created because none of the interior children have a valid parent pointer. This doesn't matter. The pseudonode will work just as well. Its pertinent child count is set to be equal to the number of blocked nodes, and each of these has its parent pointer set to the pseudonode. The reduction phase can then splice the pseudonode's children back into the tree after it is finished and no one is the wiser. The exact details are left for the program in the appendix.

This version of bubble-up clearly does everything it should. The only question is whether it does too much. This is easily answered. Bubble-Up II labels a tree exactly the same as bubble-up I. It still traverses only the pruned pertinent subtree and its handle.

Lemma 2.10:

A node is marked by bubble-up II only if it is in $\mathcal{T}_S$ or it is a node in the handle which is no further than l^* from the root of $\mathcal{T}_S$.

The same time bound as in bubble-up I applies. It follows immediately from the lemma since the work at each node is clearly O(1).

Corollary 2.11:

Bubble-Up II requires $O(|\mathfrak{T}_S|)$ steps.

This form of bubble-up does not use any parent pointers from interior children of Q-nodes. The pointers can be deleted entirely from the tree, without effect. The reduction phase can be accomplished using a scheme similar to bubble-up. If each child of an internal node adds one to the pertinent child count of its parent when it is marked, the problem of when to process the parent during reduction goes away. A parent is only queued for processing when all of its pertinent children have been accounted for by earlier reductions. The "halting problem" posed earlier has been sidestepped! Counting the number of pertinent children in a separate pre-pass accumulates all of the information necessary to correctly sequence the second pass, where the actual work of reduction is performed.

Pruned Tree Reduction

[1] Assume that each node of $\mathfrak{T}_S$ has been labeled by the bubble-up [page 67]. If bubble-up returned an answer "irreducible," set $\mathfrak{T}$ to be the null tree and halt with the answer "irreducible." Otherwise place every leaf $a \in S$ on the queue.

[2] Remove the node n at the front of the queue to be processed.

[3] Classify n as in step 3 of the normalization algorithm for formulas [page 18], and replace it in $\mathfrak{T}$ using the templates of the tree reduction algorithm [page 56]. Halt with the message "irreducible" if no template applies.

[4] If n has a parent which was labeled by bubble-up, decrement the pertinent children count by one for the parent. Place the parent on the queue to be processed if the resulting count is zero.

[5] Go back to step 2 if the queue is not empty, otherwise halt with

the message "reducible."

Although this second version of the tree reduction algorithm may seem more complicated with its two separate passes, it actually does less total work and accomplishes the same goal. The verification of this last claim is quite easy.

Lemma 2.12:

Pruned tree reduction produces exactly the same result as the basic tree reduction algorithm.

Proof:

The only way that Bubble-Up II can return with a failure is if some non-root node is doubly partial. The correct answer, "irreducible," is returned in this case. Otherwise the same labels are used and the same templates.

QED

Proving a linear time bound is a bit harder, but only a matter of carefully counting up all of the work. Reduction encounters two types of internal nodes as it travels up $\mathfrak{T}_S$. Binary nodes are those which are within the pertinent subtree and which have at least two pertinent children. They cause no trouble. Unary nodes, on the other hand, are internal nodes within the pertinent subtree which have only one pertinent child. Given a tree $\mathfrak{T}$, the total number of unary nodes encountered during reduction for the set S is defined to be δ_S. Note that, although this is obviously a function of both the tree and the set, only S is used as a subscript, the tree being understood.

The trick to proving a low time bound for reduction is to count the work done on unary nodes separately from the work for binary nodes. From the previous examples, it is clear that the unary nodes are exactly the reason the reduction is not linear in the size of S. The next result tells how to perform this accounting.

Lemma 2.13:

Pruned tree reduction requires $O(|S|+\delta_S+1)$ time to S-reduce a tree $\mathfrak{T}$.

Proof:

Corollary 2.11 states that the first pass, the bubble-up, takes only $O(\mathfrak{T}_S)$ steps. It was already claimed, without proof, that the reduction phase had the same time bound. This is easy to verify.

Claim:
Each time step 3 of the pruned tree reduction algorithm is invoked it requires only O(1) primitive operations.

Proof of claim:
All of the work is obviously involved in changing pointers. A few tricks are all that are needed. Consider case 2c, in which a P-node is replaced by a Q-node. There appear to be a large number of pointers changed. But these are either pointers from full nodes, in which case the work can be charged against the full node, or else they are empty or partial. If the original P-node is used as the new P-node having all of the empty nodes, none of their pointers have to change!

All that remains is the partial node. All of its children have a new parent, but it was agreed that no interior pointers would be kept. The only work is linking the endmost children into a sibling chain. This is easy; there are only two endmost children for each Q-node. All of the other cases are exactly the same. □

This just about wraps it up. The tree $\mathfrak{T}_S$ has at most $O(|S|)$ nodes after it is reduced because every reduced node is full, except for the root, and hence has at least two pertinent children (it is binary). All the other work is for deleting unary nodes, and this is exactly what δ_S counts.

QED

For a single reduction this is the best bound that can be given. The bad news is that the tree in example 2.9 will have to delete all but one of its internal nodes. The good news is that this can't happen very often. If a sequence of reductions is performed on the same tree, there cannot be too many nodes deleted.

Theorem 2.14:

Pruned tree reduction requires

$$O(m+n+\sum_{j=1}^{n}|S_j|)$$

steps to reduce a tree $\mathfrak{T}$ with respect to the sets $S_1,\cdots,S_n$, when $\Sigma = \{a_1,\cdots,a_m\}$.

Proof:

From lemma 2.13 the total work is bounded above by

$$O(m+\sum_{j=1}^{n}(|S_j| + \delta_{S_j}+1))$$

This is almost the desired expression. Somehow, all of the δ_{S_j} terms have to be eliminated.

At any time there are at most m nodes in the entire tree, so when a series of reductions is begun the total number of Q-nodes present can't be any greater than m. Every Q-node which is deleted must either have been in the tree originally, or else must be created by a previous reduction step. The total work eliminating nodes is thus bounded above by the number of original nodes plus the work performed by all other parts of the algorithm. This implies a total number of steps which is linear in the given expression.

QED

It is beyond the scope of this thesis to present all of the details for the lattice operations. There are more important applications for the reduction algorithm which have yet to be examined. It is possible, however, to use the same basic strategy seen in pruned tree reduction to compute either the meet or the join of two arbitrary PQ-trees over the same alphabet. An algorithm for the meet will be given without a proof of correctness or timing, but it is linear in the sizes of the trees, and therefore O(m).

The meet algorithm is controlled by the same type of queueing mechanism which is used in reduction. The items on the queue are not single nodes, however, but are pairs of nodes, one from each of the two trees being reduced. Sometimes the second node of a pair will even be a sequence of consecutive siblings. This is explained below.

Meet

[1] Let $\mathfrak{T}_1$ and $\mathfrak{T}_2$ be PQ-trees. Indicate nodes of $\mathfrak{T}_1$ by a subscript 1 (n_1) and nodes of $\mathfrak{T}_2$ by a subscript 2 (n_2). For every atom a in the alphabet Σ, place the pair of leaves (a_1,a_2) onto a queue for processing.

[2] Select the pair of nodes (n_1,n_2) at the head of the queue.

[3] Add (n_1,n_2) to the list for the parent of n_1 in $\mathfrak{T}_1$; call the parent n_1^p.

[4] If n_1 was the only child of n_1^p which was not on the parent's list, then reduce $\mathfrak{T}_2$ by the nodes in the set

$$\{\, n_2 \mid (n_1,n_2) \text{ is on the list for } n_1^p \,\}.$$

For efficiency, the reduction should start at these nodes, pretending that they are leaves of $\mathfrak{T}_2$. If the reduction fails, set $\mathfrak{T}_2$ to be the null tree and halt with the answer "irreducible." Otherwise append the pair (n_1^p,n_2^p) to the tail of the queue. n_2^p is the root of the reduced pertinent subtree in $\mathfrak{T}_2$. If the root is a partial Q-node, n_2^p will actually be a list

of the adjacent full children.

[5] Go to back to step 2 if the queue is not empty. Otherwise halt with the answer "meet computed."

When the algorithm halts the tree $\mathfrak{T}_2$ will have been transformed into the meet of the original two trees. If the trees are mutually irreducible this will be the null tree. Linearity is immediate because each tree is only scanned once, from leaves-to-root. The calculation of the join is similar, although a bit trickier. No algorithm is given here. At this writing all of the details have not been worked out. It appears that there is an $O(n\alpha(n,n))$ implementation. Further results will be reported in subsequent papers.

A linear-time algorithm has been given for computing a sequence of reductions on a PQ-tree. If the original tree is a universal tree, this becomes an algorithm for computing $S_1 \cdots S_n$, the PQ-formula corresponding to the class of arrangements in which all of the sets are consecutive. This is a problem which has a number of applications. The next three chapters will be devoted to these.

This completes all of the background material. A few of the details may be missing, but the important points have hopefully all been touched upon. A complete implementation of the entire pruned tree reduction algorithm incorporating bubble-up with blocking is included in the appendix. It should answer any remaining questions about the algorithm.

3 Planarity

The idea of a planar graph is quite familiar. A graph G = (V,E) is planar if and only if it can be drawn in the plane with no edge crossings. A number of algorithms have been proposed to test a graph for planarity. Hopcroft and Tarjan have published an algorithm for this problem which runs in time which is a linear function of the size of a graph[Hop1]. Before their paper, one of the better algorithms was the test devised by Lempel, Even, and Cederbaum[Lem1]. Although no bound was given in their original article an $O(n^2)$ running time is easy to implement. Even has subsequently pointed out that the algorithm has been implemented to run in linear time, but the result was not published[Eve2]. This chapter will show how the pruned tree reduction algorithm can be used to speed up the Lempel, Even, and Cederbaum test to so that it runs in linear time.

I Graphs

A brief notational review is in order. The graph G = (V,E) will always have a set of vertices denoted by the set

$$V = \{1,2,\cdots,n\}$$

and its edges will be the set E. The edges are either sets {i,j} of vertices, for the normal undirected graphs, or else ordered pairs (i,j), for the directed graphs. As can be seen, there are n vertices. The number of edges will be represented by e. Using these conventions, the vertex set is just a set of integers. This is convenient in a number of places. Frequently the vertices are numbered in a particular order having desirable properties. More about this later. A subgraph will always mean the induced subgraph of a set of vertices; every edge between the selected vertices is included. This is not the most common definition, but it is one which is useful for what follows.

It may not be immediately obvious that graph planarity is related to the linear arrangement problems discussed in chapters 1 and 2. Although a planar representation is a two-dimensional arrangement of the graph it can be reduced to a one-dimensional problem by considering the edges of the graph in an appropriate order. A few preliminary results will show how to make this simplification.

A graph is biconnected iff every subgraph which results from deleting a single vertex, and its incident edges, is connected; there is a path joining any two different vertices which does not pass through the deleted vertex. The biconnected components are the maximal biconnected subgraphs. Readers unfamiliar with this notion, or any of the other definitions for graphs might consult either [Aho1] or [Har1].

Theorem 3.1:

A graph G = (V,E) is planar iff each of its biconnected components is planar.

Proof=>:

Any subgraph of a planar graph is obviously still planar.

Proof<=:

It is an elementary theorem of graph theory that the biconnected components form an acyclic graph. Every acyclic graph is trivially planar so the planar representations for the biconnected components can be pieced together to form a planar representation for the entire graph, provided some care is taken when choosing the outer faces.

QED

The biconnected components can be found in O(n+e) steps using depth-first search[Aho1]. Moreover, each component can be presented in a particularly convenient form; whenever two vertices s and t are adjacent an s,t-numbering of the vertices $V = \{v_1, \cdots v_n\}$ is one in which vertex s is numbered 1, vertex t is numbered n, and for any other vertex numbered j there are vertices numbered i and k such that $i < j < k$ and both i and k are adjacent to the vertex numbered j.

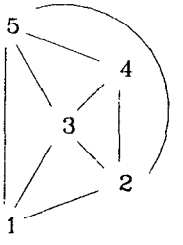
Theorem 3.2 (Lempel-Even-Cederbaum):

For any two adjacent vertices s and t in a biconnected graph there is an s,t-numbering.

The s,t-numbering corresponds to an orientation of the edges in which s is the unique source vertex and t is the unique sink vertex. This result appeared in [Lem1]. An algorithm for computing an s,t-numbering using only O(n+e) steps appears in [Eve1].

Example 3.3:

The following graph is biconnected. The numbering of the vertices is already a 1,5-numbering because vertex 1 is adjacent to vertex 5 and every other vertex is adjacent to at least one higher numbered and one lower numbered vertex.



This graph will be used later as an example. Its edges will be considered to be directed upward, from a lower numbered vertex to a higher numbered vertex.

II A New Implementation

If the edges of the graph are examined in an order consistent with the s,t-numbering, each edge only has to be examined twice, once for each incident vertex. In this manner a planar representation can be built up, edge-by-edge. As the edges for each vertex are added to the graph the edges along the "growth front" of the graph can be considered tc be in a linear arrangement. This is where the PQ-tree algorithms are helpful. The next algorithm is a restatement of Lempel, Even, and Cederbaum's planarity test.

Their concept of reduction is slightly different than that used here, but steps 2 and 3 of this algorithm are equivalent to steps 2 and 3 of their algorithm. A slight reshuffling of operations between the two steps is all that has occurred. The algorithm proceeds by adding edges to the graph according to the order determined by the s,t-numbering. Edges leaving vertex 1 are added first. At a general step the edges leaving vertex k are added.

Edges form the alphabet, which is constantly changing. The PQ-formula has an atom for each edge currently being considered. Step-by-step the edges entering a particular vertex are brought together; this is the reduction operation. Once all of the edges entering the vertex are known to be consecutive along the "growth-front" of the graph, they are replaced by the edges which leave the vertex. A complete interpretation of the formulas at each step within the algorithm is provided in [Lem1].

Planarity

[1] Let $G = (V,E)$ be a biconnected graph with an s,t-numbering. Consider each edge to be directed from its lower numbered vertex to its higher numbered vertex. Set $\varphi = (e_1 \cdots e_l)$ where each e_j is an edge of the form $(1,i_j)$. Set $k = 2$.

[2] Let $S = \{e \mid e = (j,k)\}$. Reduce φ with respect to S. If φ is irreducible halt with the answer "nonplanar."

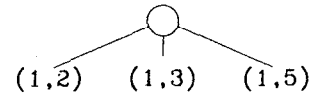
[3] Delete all of the edges of S (they are consecutive) from the reduced φ. Also delete any bracketing surrounded by these edges. Replace the deleted pieces with the product $(e_1 \cdots e_j)$, where each e_j is an edge leaving k; $e_j = (k,i_j)$. If this replacement produces an improper formula, use the appropriate substitution, as in the reduction algorithm, to insure a proper formula. This is the new φ.

[4] Set $k = k+1$. If $k < n$ go back to step 2. Otherwise halt with the answer "planar."

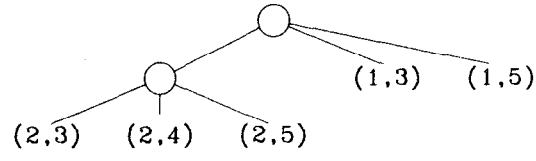
A few points are worth noting. The fact that G has an s,t-numbering means that each time step 2 is executed S will be a non-empty set. Similarly, when step 3 is performed, the product will contain at least one new edge. The algorithm is thus well-defined and it must either process every edge of the graph, halting with the message "planar," or else it must stop during some iteration of step 2 and give the message "nonplanar."

Example 3.4:

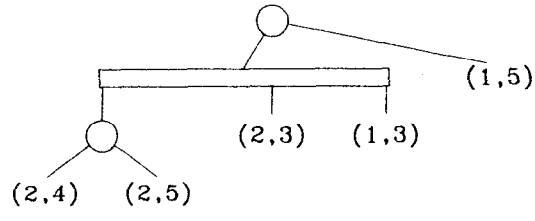
Using the graph from example 3.3 the planarity algorithm proceeds as follows. The initial tree is a P-node with three leaves.



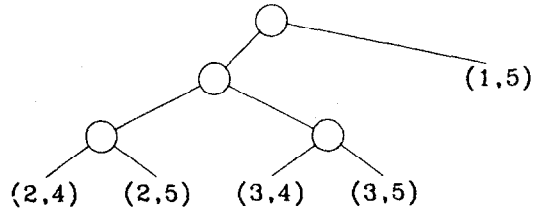
This is already $\{(1,2)\}$-reduced. The replacement of step 3 yields the next tree.



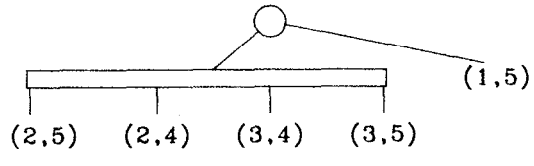
This must be $\{(1,3),(2,3)\}$-reduced. The result is the tree below.



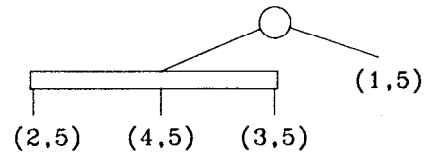
Replacement then yields the tree below.



The last reduction is with respect to $\{(2,4),(3,4)\}$.



The last replacement leaves the tree with only edges going into the sink vertex 5.



The algorithm halts with the message "planar," surely the correct response in light of the planar representation initially given in example 3.3.

The example is for a planar graph. Note that if the edge (1,4) is added to the graph it becomes K_5, one of the two forbidden subgraphs of Kuratowski's theorem[Kur1]. K_5 is known not to be planar. The interested reader should repeat the example using this second graph. Does the algorithm halt with the message "nonplanar?"

The uninterested reader will not be kept in suspense. The next theorem is stated without proof. It appears in [Lem1] as their main result.

Theorem 3.5 (Lempel-Even-Cederbaum):

A biconnected graph G is planar iff the planarity algorithm halts with k = n.

Correctness is not the issue here. The difference is in the timing. The key observation to make is that step 2, the reduction, is exactly the operation performed by the tree reduction algorithm of the last chapter. Using it as a subroutine, the entire planarity test can be performed in linear time.

This is not an immediate consequence of theorem 2.10. The reductions performed here are not all on the same tree! Each time step 3 is executed the alphabet Σ is changed. Lemma 2.13 can be used to prove a result similar to theorem 2.10 which will demonstrate the desired bound.

Theorem 3.6:

Given a biconnected graph G the planarity algorithm requires at most O(n) steps.

Proof:

Another standard result from graph theory is that any planar graph has at most 3n edges. Thus step 1 requires only O(n) steps, since it is easy to insert a check to handle the case of too many edges. Except for the actual work performing reductions, the rest of the algorithm is also proportional to the number of edges and hence O(n). It remains only to show that all of the reductions can be performed in the given number of steps.

From lemma 2.13, the total time spent in reduction is given by

$$O(\sum_{j=2}^{n-1}(|S_j| + \delta_{S_j}+1))$$

The $|S_j|$ terms all sum to O(e). All of the other terms are work spent deleting nodes. The same argument presented in theorem 2.10 applies here. Nodes cannot be deleted if they are not first created. The tree initially has no nodes, so the total number of nodes ever created is bounded above by O(e). This implies that the sum of the δ_{S_j}'s is also bounded by O(e), and hence everything is O(n).

QED

The question of optimality will not be examined very closely here. Without formalizing a computational model which includes algorithms using edge lists, it is difficult to rigorously verify the intuitive notion that any correct algorithm must look at almost all of its input. This is a much easier task for algorithms which use the adjacency matrix. The next result is simply cited here for completeness.

Theorem 3.7 (Best-van Emde Boas-Lenstra):

Any algorithm which correctly tests a graph on n vertices for planarity must examine $O(n^2)$ of the entries in the adjacency matrix.

The proof of theorem 3.5 is based on geometric arguments, properties of closed Jordan curves, etc. It does not use Kuratowski's result[Kur1]. Theorem 1.18 is a structure theorem for reducible formulas. It has the same flavor as Kuratowski's structure theorem for planar graphs. Possibly this could be manipulated into a new, perhaps simpler proof for Kuratowski's theorem. Theorems 4.16 and 5.6 are both examples of such a technique.

This chapter is intentionally short. The planarity algorithm offered here is not new; it is identical to the Lempel, Even, and Cederbaum algorithm. What is new is the implementation. Using pruned tree reduction to implement step 2 of the planarity test provides an overall linear bound for the algorithm. The main point to be made is that planarity testing is a practical application for the theory developed in the first two chapters. The next chapter will discuss some more applications.

4 Consecutive Ones

This chapter will introduce a number of properties for matrices and will examine the time required to test for these properties. All are related to the consecutive ones property, which was first defined by Fulkerson and Gross[Ful1] as part of a recognition algorithm for interval graphs, the topic of chapter 5. The consecutive ones property is of interest in its own right, and there are a number of applications other than testing for interval graphs. Various other properties have been studied which reduce to the question of consecutive ones. Among these are the circular ones property[Rys1] and the circularly-compatible ones property[Tuc2]. Not all matrices satisfy these special conditions, so some generalizations have been made which are useful for approximating the stricter definitions. Most of these extensions give rise to NP-complete problems.

1 Matrix Properties

An m×n (0,1)-matrix M has the consecutive ones property (for columns) iff there exists a permutation matrix P such that, in each column of PM, all of the ones form a consecutive sequence. That is, no two ones within a column are separated by a zero also in that column.

Example 4.1:

Both of the following matrices have the consecutive ones property. The left matrix is already in consecutive form. The right matrix can be rearranged so that it is in consecutive form.

$$\begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \begin{pmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

This is accomplished by interchanging the first and second rows and then the third and fourth rows. No rearrangement works for the next two matrices.

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix} \qquad \begin{pmatrix} 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Neither one has the consecutive ones property.

A matrix which has consecutive ones without any rearrangement of its rows is in Petrie form. This name is derived from a problem in archeology which was first posed in 1899 by Flinders Petrie[Ken1]. Given a collection of graves and a variety of artifacts which occur in the graves, an m×n (0,1)-matrix can be defined having

$$m_{i,j} = \begin{cases} 1, & \text{if the } j^{th} \text{ artifact } \in \text{ the } i^{th} \text{ grave} \\ 0, & \text{if the } j^{th} \text{ artifact } \notin \text{ the } i^{th} \text{ grave.} \end{cases}$$

Assigning a chronological ordering to the graves is called sequence dating. If each grave contains exactly the set of artifacts that existed during the period in which the grave was filled, the correct sequence dating would result in a matrix whose columns have consecutive ones. Turning this idea around, a sequence dating which achieves consecutive ones is a likely candidate for being the actual chronology; hence the interest in consecutive ones algorithms.

The chances of such a perfect archeological record are remote, however, so the real interest is in matrices which only approximate the consecutive ones property. This generalization will be analyzed after the exact consecutive ones problem has been discussed.

An application closer to computer science is in the field of information retrieval. Ghosh has examined query systems for which optimal storage schemes can be achieved on disk- or drum-like devices[Gho1]. An information retrieval system consists of a collection

$$R = \{r_1, r_2, \cdots, r_m\}$$

of records and a family

$$\mathcal{Q} = \{Q_1, Q_2, \cdots, Q_n\}$$

of queries. Each query is a subset of R. A record is pertinent to a particular query only if it is one of the records in the query. The query incidence matrix, $M_{\mathcal{Q}}$, is defined to be the m×n (0,1)-matrix having

$$m_{i,j} = \begin{cases} 1, & \text{if } r_i \in Q_j \\ 0, & \text{if } r_i \notin Q_j. \end{cases}$$

Records can be arranged linearly so that each query is a consecutive set iff $M_{\mathcal{Q}}$ has the consecutive ones property.

Such a storage layout is optimal in the sense that each query requires only one positioning move (seek operation) every time the pertinent records are fetched. Ghosh coined the phrase consecutive retrieval property to describe query systems having such an optimal arrangement[Gho1]. This is clearly just a restatement of the consecutive ones property.

Most actual systems will not have the consecutive retrieval property, but testing for the special case is of some interest. The general case of assigning storage, even when the query matrix does not have the consecutive ones property, is discussed further in [Esw1] and [Gho1-4].

A final example involves what has been called the intervalization of parsing tables[Jen1]. Compiler generating systems routinely turn out table-driven parsers for various subclasses of the deterministic context-free languages. One description of a parsing machine is the state transition matrix for the automaton. Looking at the zero / nonzero structure, an efficient storage scheme will order the states (or symbols) in such a way that each column (or row) has consecutive ones.

A number of other applications should come to mind in which it would be convenient to arrange objects so that various sets of them appear consecutively. The rest of this chapter will concentrate on a mathematical treatment of this idea, the consecutive ones property, and also some of its variants. For the most part, the terminology is that of Fulkerson and Gross in [Ful1].

An efficient test for the consecutive ones property is a straightforward application of the pruned tree reduction algorithm [page 70], and can be solved using the techniques developed in chapter 2.

Lemma 4.2:

If M is an m×n (0,1)-matrix, let

$$\Sigma = \{1,,2,\cdots,m\}$$

and let

$$\mathcal{S}_M = \left\{S_j = \{i \mid m_{i,j} = 1\} \mid j=1,2,\cdots,n\right\}.$$

M has the consecutive ones property iff the PQ-formula

$$\varphi_M = S_1 \cdots S_n$$

is not the null formula.

Proof:

It is easy to verify, using the definitions plus properties of the reduction algorithm, that all of the following are equivalent statements:

(1) M has the consecutive ones property.

(2) There is a permutation of the rows of M in which all of the columns have their ones consecutive.

(3) There is an arrangement of Σ in which all of the sets S_j are consecutive.

(4) The PQ-formula φ_M is not the null formula.

The lemma is simply the equivalence of the first and fourth statements in this list.

QED

Lemma 4.2 shows that one way to test a matrix for the consecutive ones property is to construct a universal PQ-tree having m leaves, one for each row, and then repeatedly reduce the tree with respect to the columns of the matrix. If all of the reductions can be performed, the matrix has the consecutive ones property. If any reduction fails, the matrix cannot be placed into Petrie form. In the case of success, a single permutation matrix P can easily be reconstructed in linear time from the tree which is left at the end of the algorithm. Even better, all of the admissible permutations can be enumerated, in only the time required to print the results, by exhaustively obtaining all orientations of the PQ-tree.

Theorem 4.3:

Given a list of its e nonzero entries, an m×n (0,1)-matrix M can be tested for the consecutive ones property in O(m+n+e) steps.

Proof:

Use the procedure outlined in lemma 4.2. The sets described there can be constructed in O(n+e) steps from the list of nonzero entries. The reductions require only O(m+n+e) steps if pruned tree reduction is used. By the preceding lemma, this is a complete test for the consecutive ones property.

QED

When M is not given as a list of nonzero entries the algorithm cannot capitalize on the sparseness of M. A slightly longer running time results, but one which is still linear in the total size of the matrix.

Corollary 4.4:

An m×n matrix M can be tested for the consecutive ones property in O(mn) steps.

Proof:

This follows immediately from the theorem because mn is an upper bound for e.

QED

Proving optimality for this algorithm is a bit tricky. Intuitively, any algorithm which performs a consecutive ones test must examine all of its input. Pinning down the exact computational model is awkward, however, especially if edge lists are involved. It is easier to make statements about algorithms which operate only on the matrix itself.

The Aanderaa-Rosenberg Conjecture[Bes1, Rsn1] attempts to formalize this

idea for general graph properties, claiming that every nontrivial, monotone property of undirected graphs requires $O(n^2)$ probes of the adjacency matrix. Rivest and Vuillimin proved that such a graph property will indeed require $O(n^2)$ probes for some graph having n vertices[Riv1]. Theirs is a theorem about graphs, but the consecutive ones property can easily be cast as a bipartite graph property. This is done in [Tuc3].

$G = (V_1, V_2, E)$ is a bipartite graph with disjoint vertex sets V_1 and V_2 iff all of the edges E are of the form

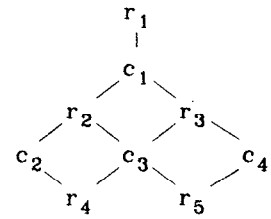
$$\{i,j\} \quad \text{where} \quad i \in V_1 \quad \text{and} \quad j \in V_2.$$

The correspondence with matrices is obvious. Rows in the matrix are the vertices of V_1 and columns are the vertices of V_2. Adjacency is determined by the nonzero entries in the matrix. The results of Rivest and Vuillemin can then be used.

Example 4.5:

As an example consider the following matrix.

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



Its bipartite graph is given on the right. The rows are the vertices r_i and the columns are vertices c_j. This particular matrix does not have consecutive ones. Its bipartite graph is the same as one of the forbidden subfamilies in theorem 1.16. More about this later.

Using someone else's theorem is always the easiest way out. Unfortunately, the hypothesis doesn't apply here. The consecutive ones property is not a monotone graph property. That is, given a matrix (bipartite graph) which has the consecutive ones property, matrices which do not have consecutive ones can be obtained either by deleting ones from the matrix or by adding ones. The conclusion of Rivest and Vuillemin is still true, however, and can be proven using *ad hoc* arguments specific to this problem. A number of such results are included in [Bes1]. The result here can most probably be sharpened considerably, but is sufficient to show optimality to within an order of magnitude.

Theorem 4.6:

Let m and $n \geq 3$. Any algorithm which correctly tests an m×n matrix for the consecutive ones property, using only the matrix itself as input, must examine O(mn) of the entries.

Proof:

Assume that some algorithm correctly tests any m×n matrix without examining at least $\frac{1}{3}mn-2(m+n-2)$ of the entries. A contradiction will be derived.

Claim:
No 3×3 submatrix of a matrix with consecutive ones can look like the following matrix.

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

Proof of claim:
By inspection, the reduction algorithm fails on this matrix. Adding extra rows and columns can only make things worse. □

Consider a matrix of all ones. Partition it into 3×3 blocks. The algorithm must guarantee that none of these blocks is the matrix

above. To do so, it has to check at least three ones within each block before it can differentiate a 3×3 block of ones from the illegal matrix. This requires examining about one third of the total entries. The expression above is a more precise lower bound.

QED

This last result more or less disposes of the question of an optimal test for consecutive ones, at least as far as algorithms operating only on matrices are concerned. Any matrix having the consecutive ones property can be recognized in linear time, and those which do not can also be detected. But there is more to be said about the failure cases.

Since not all matrices can be placed in Petrie form, it is reasonable to look at modifications of the definitions which may be more easily satisfied by arbitrary matrices. The next section of this chapter will examine different ways in which the conditions can be relaxed, and also review the proof of a structure theorem characterizing matrices which fail to have the consecutive ones property.

There are a number of possibilities for relaxing the conditions. Ryser defined a circular analog, saying that an m×n (0,1)–matrix M has the circular ones property (for columns) iff there exists a permutation matrix P such that, in each column of PM, all of the ones form a consecutive sequence, where the sequence is allowed to wrap around the top and bottom of the column[Rys1]. That is, if the column is treated as a continuous circular band, no two ones are separated from each other by two zeros in that same column. This condition includes consecutive ones as a special case, but also admits matrices which do not have the consecutive ones property.

Example 4.7:

All of the matrices in example 4,1 have the circular ones property, even the ones which don't have consecutive ones. These are clearly in circular form already.

$$\begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Not all matrices have circular ones, however, since neither of the two shown above have any permutation of the rows in which all the columns are circular.

Tucker has shown that the circular ones property can be reduced to the consecutive ones property[Tuc2]. His reduction forces at least one row of the matrix to be all zeros, effectively removing the freedom to wrap around and eliminating the truly circular case. Let x^c denote the one's complement of a number. Given a matrix M, and a particular row k, define a new matrix M' using the rule

$$m'_{i,j} = \begin{cases} m_{i,j} & \text{if } m_{k,j} = 0 \\ m^c_{i,j} & \text{if } m_{k,j} = 1. \end{cases}$$

Each column of M having a one in the k^{th} row is complemented. This has the effect of setting every entry in row k to zero, the ones being complemented to zeros and the zeros remaining unchanged. This is desirable for the following reason.

Theorem 4.8 (Tucker):

A matrix M has the circular ones property iff M′ has the consecutive ones property.

Proof =>:

Suppose that M has the circular ones property. There exists a permutation matrix P such that PM has circular ones in each column. Let P′ be the permutation matrix obtained from P by pre-multiplying with yet another permutation matrix which cyclically shifts the permuted rows of PM until the original k^{th} row is at the top. This in no way affects the circularity of ones in the columns.

Complementing any particular column does not affect circularity either, since both the ones and the zeros are circular. Thus P′M′ has circular ones. But the top row, which used to be row k of M, is now all zero. This precludes the possibility of any column having ones which actually wrap around. P′M′ is thus in Petrie form, and hence M′ has the consecutive ones property.

Proof <=:

If PM′ has consecutive ones, it automatically has circular zeros. Complementing columns doesn't change the circularity of either the ones or the zeros, so PM has circular ones.

QED

Tucker's reduction works for any choice of k. If an arbitrary k is chosen, the matrix M′ may have a very large number of ones added by the complementation. This all by itself could be quite a chore! A careful choice of k will avoid this pitfall.

Theorem 4.9:

Given a list of its e nonzero entries, an m×n (0,1)-matrix M can be tested for the circular ones property in O(m+n+e) steps.

Proof:

Choose a row k having a minimum number of ones. This can be done in O(m+n+e) steps by simply maintaining counts of the nonzero entries in each row. Form M′ by complementing the appropriate columns. This can be accomplished by scanning each row in parallel with row k.

Claim:
M′ has e′ nonzero entries, where $e' \leq 2e$.

Proof of claim:
Since row k was chosen to have a minimum number of ones, the claim is easy to verify. Let e_i be the number of ones in row i of M and consider what happens to row i. The total number of ones after complementation is at most e_i+e_k, the number of ones in row i before complementation plus the maximum number of complementations which could occur in any row. But $e_k \leq e_i$ by assumption. Thus $e_i' \leq 2e_i$. □

The claim guarantees that the complementation requires at most O(e) steps and also that the consecutive ones check takes only O(m+n+e) steps. Adding up all the work still gives an overall bound of O(m+n+e). By theorem 4.8, this is equivalent to testing M for the circular ones property.

QED

Again the lack of information about the sparseness of M increases the amount of work which must be done, but no more than before.

Corollary 4.10:

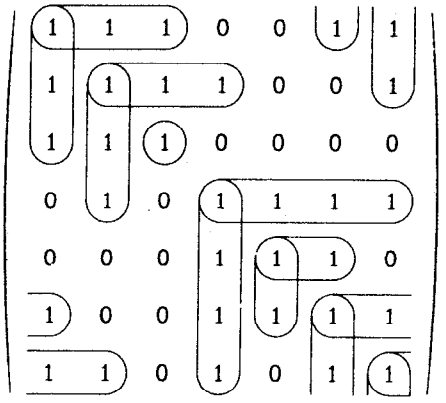
Any m×n matrix M can be tested for the circular ones property in O(mn) steps.

Tucker defined two additional variations. These properties are defined only for a special type of matrix. The reason this is done will become clear later, when the properties are used to characterize families of graphs. The definitions which follow actually paraphrase Tucker's, because consistency with earlier definitions demands that a matrix have a specified property whenever any suitable permutation of that matrix has the property. This is a minor technicality, and does not affect any of the results.

Given a symmetric (0,1)-matrix having ones along the diagonal, let U_i be the set of ones that appear in row i, beginning on the diagonal and continuing right, possibly wrapping around circularly, until the first zero is encountered. Let V_j be the similar set of ones in column j, begining on the diagonal and continuing downward until the first zero. M has the quasi-circular ones property iff each one appearing in PMP^T is in some U_i or V_j of the permuted matrix.

Example 4.11:

This following matrix has the quasi-circular ones property, the sets U_i and V_j being circled.



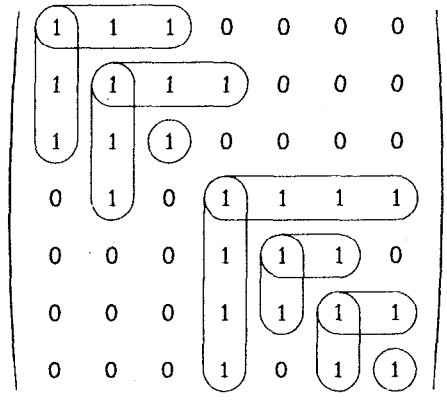
This example is given in [Tuc2] and is shown not to have the circular ones property. An easy check is to run the algorithm.

Stricter definitions of U_i and V_j, in which no circular wrap around is allowed, lead to the idea of the quasi-consecutive ones property. Apparently, this term has not appeared in the literature. This does not mean that the concept has not already been explored! A symmetric matrix with ones on the diagonal, having all of its ones accounted for as members of the U_i and V_j, is said to have dense profile. Well, sort of; the standard definition of dense profile actually has the sets U_i going up from the diagonal and the sets V_j going tc the left.

The reason the standard form was chosen this way will be clearer after Gaussian elimination schemes have been examined in the next chapter. Which definition is used makes no real difference since one is just the reversal of the other. Not wanting to introduce extraneous terminology at this point, it is suggested that the phrase quasi-consecutive ones be discreetly retired and reserved only for purposes of comparison with its circular counterpart. Dense profile is a more descriptive term anyway.

Example 4.12:

Here is a matrix with dense profile. As before, the sets U_i and V_j are circled.



Not every matrix having dense profile has the consecutive ones property. The matrix above is a case in point.

Chapter 5 will have more to say about both of these properties. In particular, there is an O(n+e) test for dense profile, but there is no known polynomial algorithm to test for quasi-circular ones. There is good reason to suspect that this may be substantially harder to recognize than some of the other variations of consecutive ones.

Tucker's second generalization is also for symmetric matrices having ones along the diagonal. Such a matrix has the circularly compatible ones property iff the ones are circular in each column and, for every permutation of the rows and columns which is a cyclic shift or inversion of their order, the "last" one in the first column does not occur after the "last" one in the second column, excluding columns which are either all zeros or all ones. The "last" one in a column is the one which is followed circularly by a zero.

Example 4.13:

The matrix below has circularly-compatible ones.

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

Try all of the shifts and inversions.

This rather complicated property will also be seen again in chapter 5. For now it is sufficient to note that a recognition algorithm is no harder than for consecutive ones. The proof is in Tucker's thesis, where he shows how to modify a consecutive ones test to check for circularly compatible ones[Tuc1]. Theorem 4.3 finishes the job.

Theorem 4.14 (Tucker):

$O(m^2)$ steps are sufficient to decide whether an m×m matrix M has the circularly compatible ones property.

The consecutively compatible ones property is simply the requirement that every row and column of PMP^T have consecutive ones. The additional condition regarding cyclic shifts and inversions of the rows and columns is redundant. It is always satisfied by a symmetric matrix having ones on the diagonal if the matrix is in Petrie form. Nothing new is added by the seemingly stronger conditions. Just like quasi-consecutive ones, this property has also appeared under a different name in the

literature, where it is called monotone dense profile.

Example 4.15:

The matrix on the left does not have monotone dense profile. No matter how the rows and columns are permuted, it will not have consecutive ones

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} \qquad \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}$$

The matrix on the right does have monotone dense profile. All of its ones are consecutive in every row and column.

The history of the consecutive ones property is interesting. Its first appearance was in [Ful1], where Fulkerson and Gross defined it as an aid in testing for interval graphs. They also proved something akin to a canonical form theorem for matrices which have the property. This involved the matrix product M^TM. The computation of this product was of central importance to their test for consecutive ones.

M^TM gives the inner product of each pair of columns. If the matrix has only zeros and ones, the inner product is the number of ones overlapping in the two columns. Using these values, the columns can be adjusted, one by one, until the entire matrix M has been reconstructed in Petrie form. Most of the work involved is in the calculation of the matrix product.

Fulkerson and Gross give an overall bound of $O(mn^2)$ for m×n matrices, but their work was pre-Strassen[Str1]. Using a faster matrix multiplication, an improved bound characteristic of problems reducible to matrix multiplication is obtained,

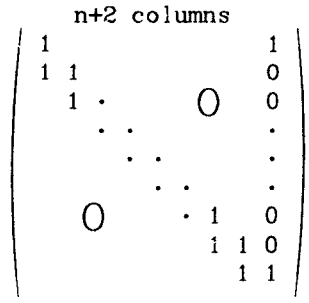
$$T(m,n) = \begin{cases} O(m^{(\log_2 7)-2}n^2), & \text{if } m < n \\ O(m^{(\log_2 7)-1}n), & \text{if } m \geq n. \end{cases}$$

In either case, the algorithm presented here is clearly superior, with its worst case O(mn) running time.

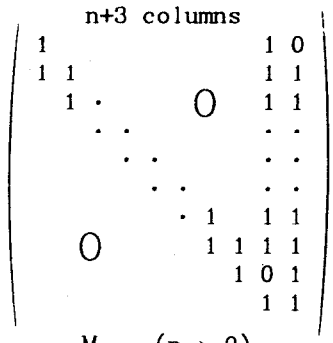
After Fulkerson and Gross published their results, a number of other researchers began studying the property. Tucker was able to completely characterize matrices having the consecutive ones property using a forbidden submatrix approach. The proof here is different than Tucker's in that it is algorithmic, following directly from the results in chapter 1.

Theorem 4.16 (Tucker):

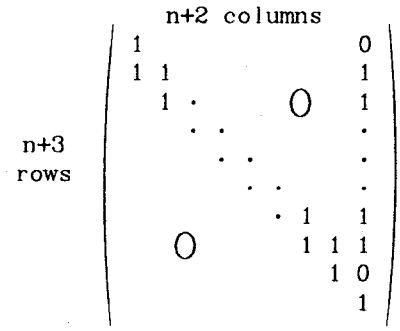
A matrix M has the consecutive ones property iff it has no submatrix whose rows can be ordered to be one of the following types of matrices:



M_{I_n} $(n > 0)$



M_{II_n} $(n > 0)$



M_{III_n} $(n > 0)$

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

M_{IV}

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

M_V

(These are the forbidden submatrices. The first three types are actually matrix schemata, having a parameter $n > 0$, while the last two are simply matrices.)

Proof:

The theorem is an immediate corollary of theorem 1.18. These are exactly the five types of matrices which correspond to the five types of forbidden subfamilies of normal sets. The matrices are the bipartite adjacency matrices for the families. Rows (elements of V_1) correspond to atoms and columns (elements of V_2) to normal sets, a matrix element being nonzero if and only if the atom is a member of the set.

QED

The search here for an easy-to-satisfy generalization of consecutive

ones has so far been unsuccessful. Most matrices have none of the previous properties. What is needed is a "closeness" parameter which can be adjusted so that any matrix can be classified according to its degree of fit. The remainder of the chapter will be concerned with generalizations of this type.

The fact that the properties are being weakened does not, however, mean that testing for the new conditions will be correspondingly easier. On the contrary, the tests become much harder! Most of the results which follow will be negative. Dealing only with finite structures, problems are always solvable, but the solutions can often become totally impractical. This is exactly what happens here.

II NP-Complete Problems

At the beginning of this chapter, the sequence dating problem of Flinders Petrie was cited as a possible application for the consecutive ones test. The fact that real-world problems very seldom exhibit such ideal characteristics somewhat dampens enthusiasm for this example. A more appropriate question, for this and many other related problems, is whether an approximately consecutive matrix has been found. There are a number of different ways of defining approximate and different applications will naturally prefer one over another. Most of these lead to NP-complete problems.

The following result is due to Cook[Coo1]: any problem which can be solved by a nondeterministic polynomial time algorithm can be reduced in deterministic polynomial time to an instance of the problem of satisfiability of propositional formulas in conjunctive normal form.(1) Satisfiability is itself known to be solvable by a nondeterministic polynomial time algorithm, but no solution is known which runs in deterministic polynomial time. Karp defined the class of NP-complete problems as those problems which could be substituted for satisfiability in Cook's theorem[Kar1-2]. The implication of Cook's result is that

---

(1) It is beyond the scope of this thesis to review all of the theory behind NP-complete problems. The references cited here provide a very thorough discussion and do a much better job than will be attempted here.

deterministic polynomial time solutions for NP-complete problems are like the Three Musketeers, existence for even one implies existence for all.

A large number of problems are known to be NP-complete. Additional examples, and a general review of the definitions can be found in any of [Aho1], [Coo1], [Gar1], or [Kar1-2]. For the present development, it is sufficient to define a problem as being NP-complete iff it has a nondeterministic polynomial time algorithm and if, given a deterministic polynomial time algorithm for the problem, some other known NP-complete problem can be solved in deterministic polynomial time.

Many NP-complete problems have been studied for years and yet have no known polynomial algorithm. This is fairly compelling evidence that they are, in fact, inherently combinatorial in nature, and that no polynomial algorithm exists for any one of them. A typical such problem, and one which is useful here, is the following described in [Gar1].

An ordering for a set V is any numbering of the elements. A graph G = (V,E) has a simple optimal linear arrangement of weight W iff there is an ordering α of the vertices such that

$$\sum_{\{i,j\}\in E} |\alpha(i) - \alpha(j)| \leq W$$

Less formally, the problem is to decide whether there is a numbering of the vertices such that weighting edges by the difference of the endpoints results in a sum of no more than W. This is a difficult problem in the following sense.

Theorem 4.17 (Garey-Johnson):

Deciding if G has a simple optimal linear arrangement of weight W is NP-complete.

The generalization of consecutive ones which is of interest for the

sequence dating problem is the addition of enough ones to a matrix so as to achieve the consecutive ones property. This corresponds to adding hypothetical occurrences of artifacts in graves. A good approximation will be one which adds a small number of artifacts. Say that a matrix M has the k-augmented consecutive ones property iff there exists a matrix M′ which has the consecutive ones property and which is identical to M, except for at most k entries in which M′ has a one where M has a zero.

Example 4.18:

These are the same two matrices as in example 4.5.

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} \qquad \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}$$

The right one is a 2-augmentation of the left one. Ones have been added in the (3,4) and (4,3) positions and then rows (and columns) 1 and 2 have been interchanged.

A similar definition can be made for the k-augmented circular ones property. In both cases the parameter k is a measure of closeness to the exact consecutive or circular ones property. Papadimitriou has independently observed the following result[Pap1].

Theorem 4.19:

Deciding if a matrix M has the k-augmented consecutive ones property is NP-complete.

Proof:

Given a k-augmentation for M, it is easy to verify that the resulting matrix has the consecutive ones property using the reduction algorithm [page 70]. A nondeterministic algorithm can generate a k-augmentation in O(mn) steps by guessing which elements to change. This implies that there is a nondeterministic algorithm for testing the k-augmented consecutive ones property in polynomial time. It must be shown that, given a deterministic polynomial time algorithm for testing k-augmented consecutive ones, some other NP-complete problem can be solved in deterministic polynomial time. Simple optimal linear arrangement will serve this purpose.

Given any graph G = (V,E), and a weight W, construct M, the incidence matrix for the graph. The rows correspond to vertices and the columns to edges, with

$$m_{i,j} = \begin{cases} 1, & \text{if } v_i \in e_j \\ 0, & \text{if } v_i \notin e_j. \end{cases}$$

M will be an n×e matrix if G has n vertices and e edges.

Claim:

G has a simple optimal linear arrangement of weight W iff M has the (W-e)-augmented consecutive ones property.

Proof of claim:

Note that for any ordering of the vertices (arrangement of the rows) the number of zeros between the two ones in each column is exactly one less than the difference between the row numbers of the two incident vertices. The total number of holes (zeros to be augmented by ones) is thus exactly e less than the weight of the linear arrangement. □

Suppose there were a deterministic polynomial time algorithm which decided if a matrix had the k-consecutive ones property. Using

the construction above, any instance of simple optimal linear arrangement can be reduced in deterministic polynomial time to an instance of the k-augmented consecutive ones problem. There would thus be a deterministic polynomial time algorithm for simple optimal linear arrangement, itself NP-complete.

QED

This last result effectively kills any hope for an optimal algorithm to test for the k-augmented consecutive ones property. The evidence against the existence of polynomial algorithms for all NP-complete problems is quite substantial. A similar negative result holds for the k-augmented circular ones property.

Corollary 4.20:

Deciding if a matrix M has the k-augmented circular ones property is NP-complete.

Proof:

Let M be any m×n (0,1)-matrix. Unless $k \leq mn$ the answer is trivial, so consider only problems in this range. Obtain M′ from M by adding k+1 rows of all zeros. M has the k-augmented consecutive ones property iff M′ has the k-augmented circular ones property. Constructing M′ takes only $O(m^2n)$ steps, so any polynomial algorithm for the k-augmented circular ones problem gives a polynomial algorithm for the k-augmented consecutive ones property.

QED

It is easy to show that both of these results, and the others which follow, are true whether matrices are presented explicitly as zeros and ones (as assumed here) or as lists of nonzero entries. The details are left to the interested reader.

Another criterion for approximating solutions arises from the

application to information retrieval systems. If a query set does not have the consecutive retrieval property there are two different ways to relax the conditions. The first is to allow redundant records, while enforcing the requirement that each query be stored consecutively. The second maintains a single copy of each record, but provides for queries which are scattered into a number of blocks of consecutive storage locations. The amount of redundancy or scattering is then a measure of the closeness of the solution to the ideal case. These schemes were suggested by Eswaran in his thesis [Esw1]. Kou then proved that both questions are NP-complete problems[Kou1].

The two extensions have a natural interpretation in terms of matrices. A row split of M is any matrix M′ formed from M by replacing a single row with two new rows whose ones are a partitioning of the ones in the original row. A similar definition applies for column splits.

Example 4.21:

Again the left matrix of example 4.15 serves as a guinea pig. The left matrix below is a row split. Row 1 has been split into two pieces, one remaining at the top and the other placed between rows 3 and 4.

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix} \qquad \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

The right matrix is a column split. Column 3 was split with one of its ones being moved to column 5. Rows 1 and 2 were then interchanged.

Adding redundant records is row splitting. Each time a record is

duplicated its row is removed from the query matrix and is replaced by two rows, one corresponding to the queries which access the first copy of the record and the other for the queries which access the second copy. In a similar fashion, column splits represent queries which have more than one block of records. Either extension leads to an NP-complete problem, as shown by the next (double) theorem.

Theorem 4.22 (Kou):

Deciding if a matrix M has the consecutive ones property after either k row splits or k column splits is NP-complete.

Proof:

The proof is a polynomial time reduction of the Hamilton path problem to either the problem of row splits or column splits. The details are in [Kou1].

QED

As before, a similar theorem can be obtained for the circular case. The analogy carries over completely, the path becoming a circuit. Although not stated in his original paper, Kou's proofs still hold when the Hamilton circuit problem is substituted for the Hamilton path problem.

Corollary 4.23:

Deciding if a matrix M has the circular ones property after either k row splits or k column splits is NP-complete.

The final two generalizations are again Ryser's suggestions[Rys1]. A matrix is k-consecutive iff every m×k submatrix has the consecutive ones property; it is k-circular iff every m×k submatrix has the circular ones property. Ryser observed that testing for either of these two properties appears to be harder, in the general case, than the special cases of n-consecutive or n-circular ones (the consecutive ones or circular ones property, respectively). How much harder is not known,

but it is conjectured here that deciding if a matrix has either of these two properties is NP-complete. It is easy to show that the existence of a single consecutive or circular submatrix is hard to verify.

Theorem 4.24:

Let M by an m×n matrix. Deciding if there exists any m×k submatrix having the consecutive ones property is NP-complete.

Proof:

A simple reduction can be made from the Hamilton path problem. Given a graph G = (V,E), let M be the incidence matrix. G has a Hamilton path iff M has an n×(n-1) submatrix with the consecutive ones property. The Hamilton path problem is known to be NP-complete.

QED

The corollary is immediate, another instance of the almost complete analogy between the linear and circular cases. An n×n submatrix with the circular ones property corresponds to a Hamilton circuit in the graph.

Corollary 4.25:

Let M by an m×n matrix. Deciding if there exists any m×k submatrix having the circular ones property is NP-complete.

Using the reduction algorithm for PQ-trees, linear-time algorithms have been given for testing a matrix for the consecutive ones property, the circular ones property, and the circularly-compatible ones property. Tucker's forbidden submatrix characterization of matrices having the consecutive ones property was shown to be a direct by-product of the reduction algorithm stated in chapter 1. A number of generalizations for the consecutive ones property have been shown to be NP-complete and hence most probably unsolvable by any polynomial time algorithm. The

question of how hard it is to test for quasi-circular ones remains open, as do the questions for k-consecutive and k-circular ones.

5 Graph Models

Fulkerson and Gross defined the consecutive ones property as an aid in solving another problem, recognizing interval graphs[Ful1]. Their test had two parts: first compute the dominant-clique vs. vertex matrix for the graph and then test the matrix for the consecutive ones property. The algorithm in chapter 4 handles the second task in linear time. The first phase of the test can also be accomplished in linear time, using a new test for chordal graphs[Lue1-3, Ros3]. Combining the two gives an algorithm for testing interval graphs which is optimal in a sense to be described later. Interval graphs are just one special case of a more general type of graph called chordal graphs. Two more classes are properly contained between these classes; a number of recognition and isomorphism questions are of interest for each of these classes of graphs. In particular, it has been shown that the question of general graph isomorphism is no harder than isomorphism for chordal graphs, and that interval graph isomorphism is actually solvable in linear time[Bth1, Lue2].

I Models And Matrices

An intersection model for a graph G = (V,E) is a family of sets

$$\mathcal{S}_G = \{ S_i \mid i \in V = \{1,2,\cdots,n\} \}$$

having the property that two vertices i and j are adjacent in G iff the corresponding sets S_i and S_j have a non-empty intersection. If this is the case, G is called the intersection graph for the family of sets. An intersection model is proper if no member of the family is properly contained within another. The corresponding graph is called a proper intersection graph.

A graph G = (V,E) is an interval graph iff it has an intersection model composed entirely of intervals chosen from some total ordering. The underlying set for the ordering may be of arbitrary cardinality. Obviously only finite graphs are of interest when discussing algorithms, so for these purposes the underlying set may be standardized as the real line, or any finite interval on the real line. Also without loss of generality, the intervals can be chosen to be finite and closed. This is convenient, so it will be assumed that all interval models are of this form. If there is a proper interval model, G is called a proper interval graph.

The first mention of interval graphs to appear in the literature is apparently the note by Hajós which posed the question of characterizing the intersection graphs of intervals on a line[Haj1]. The first solution was by Lekkerkerker and Boland[Lek1]. They cited the work of the biologist Benzer as a motivation for studying interval graphs[Ben1-2].

Benzer was interested in modeling the fine structure of genes. Just as chromosomes are linear arrangements of genes, Benzer proposed that the genes themselves might also be linear arrangements of subelements. To test this hypothesis a number of recombining experiments were performed using mutants. Observing a large number of these produced a body of data concerning the intersections of the various mutations. His problem was to reconstruct a linear model for the gene given this recombination data. As the applications in chapter 4 tend to only approximate systems in which exact consecutive ones arrangements exist, so the genetic data, with its imperfections due to sampling techniques, tends to only approximate an interval model.

Nonetheless, such work has served as the inspiration for the futher mathematical analysis of interval graphs. Since then a number of more important applications have surfaced, although the primary justification is still probably the combinatorial problem itself. After reviewing the known results for chordal and interval graphs, a number of new results will be discussed. After that, some specific applications to numerical analysis will be treated.

A convenient way to represent a graph is by its augmented adjacency matrix $M^*(G)$. This is just the adjacency matrix M(G) with ones added along the diagonal. There are a number of interesting characterizations which can be made in terms of the augmented adjacency matrix.

Theorem 5.1:

G is an interval graph iff its augmented adjacency matrix has dense profile.

Proof =>:

Suppose that G = (V,E) is an interval graph. G has an intersection model composed of closed, finite intervals on the real line. Let the model be

$$\mathcal{I}_G = \{ [l_i, r_i] \mid i \in V = \{1,2,\cdots,n\} \}.$$

Number the vertices of V according to the increasing order of their left endpoints. In this numbering,

$$i < j < k \quad \Rightarrow \quad l_i \leq l_j \leq l_k.$$

Present the augmented adjacency matrix so that the rows for the vertices have the same order as in the numbering.

Claim:
$M^*(G)$ has dense profile with its rows in this order.

Proof of claim:

It is sufficient to examine only elements above the diagonal, since the matrix is symmetric. Consider any nonzero entry $m_{i,k}$ for which $i < k$. In order that M have dense profile, it must be true that

$$i < j < k \quad \Rightarrow \quad m_{i,j} \neq 0.$$

But this is almost immediate from the way the ordering of the vertices (intervals) was chosen!

Because $m_{i,k}$ is nonzero, it is certainly true that

$$l_k \leq r_i$$

but transitivity then implies that

$$l_j \leq r_i.$$

This is all that is needed to guarantee that the two intervals

$$[l_i, r_i] \quad \text{and} \quad [l_j, r_j]$$

actually intersect in at least one point, l_j. It follows that $m_{i,j}$ is nonzero. □

Proof <=:

Suppose that $M^*(G)$ has dense profile. Without loss of generality assume that the rows have been ordered so the ones form a dense profile. For each i, let k_i be the index of the last one in the i^{th} column. Consider the family of intervals

$$\mathcal{S}_G = \{ [l_i, r_i] \mid i \in V = \{1, 2, \cdots, n\} \}$$

where

$$l_i = i \quad \text{and} \quad r_i = k_i.$$

Claim:

$$I_i \cap I_j \neq \phi \quad <=> \quad m_{i,j} \neq 0.$$

Proof of claim:

Without loss of generality, assume that $i < j$. If the intervals intersect, k_i must be at least as large as j, implying that $m_{i,j}$ is nonzero. Conversely if $m_{i,j}$ is nonzero the two intervals intersect since k_j is at least j. □

QED

This last proof is almost identical to the one used by Fulkerson and Gross in their characterization of interval graphs[Ful1]. It is even closer to the following result which Roberts obtained for proper interval graphs[Rob1-2].

Theorem 5.2 (Roberts):

G is a proper interval graph iff its augmented adjacency matrix has the consecutive ones property.

Proof =>:

The fact that no interval is properly contained in another means that ordering the intervals by right endpoint is exactly the same as ordering by left endpoint. Otherwise some interval is completely contained within another. It is then easy to see that all of the ones in a column, both above and below the diagonal, are consecutive.

Proof <=:

The construction used in theorem 5.1 is almost all that is needed. The only problem is guaranteeing that the model is proper. This is accomplished by defining

$$\mathcal{I}_G = \{ [l_i, r_i] \mid i \in V = \{1,2,\cdots,n\} \}$$

where

$$l_i = i \quad \text{and} \quad r_i = k_i + 2^{i-n-1}.$$

Each of the intervals is just a little bit longer than before. Not enough longer to introduce new intersections, but sufficient to eliminate any proper containments. This is easy to check.

Claim:

$\mathcal{I}_G$ is a proper interval model.

Proof of claim:

Suppose that I_i actually contained I_j properly. This is the same as saying that

$$l_i \leq l_j \qquad \text{and} \qquad r_j \leq r_i$$

These two conditions imply that

$$i < j \qquad \text{and} \qquad k_j < k_i$$

The first holds because of the way the intervals were numbered and the second because the extra power-of-two is a fraction (the exponent is always negative) which means that k_i must be strictly greater than k_j. $M^*(G)$ is symmetric, so the rows also have consecutive ones. This immediately gives a contradiction, however, because

$$i < j < k_j < k_i$$

yet both $m_{k_i,i}$ and m_{k_i,k_i} are clearly nonzero; consecutive ones in row k_i would require that $m_{k_i,j}$ also be nonzero, which is impossible if $m_{k_j,j}$ is the last nonzero entry in column j. □

QED

What happens if the intersection model consists of intervals chosen from a circle instead of just from a line? Interval graphs are certainly included as a special case, but so are many other new graphs. This generalization was suggested by Klee[Kle1] and analyzed by Tucker[Tuc1-2]. A circular-arc graph is one which has an intersection model composed entirely of arcs from a circle; a proper circular-arc

graph is one which has a proper intersection model composed entirely of arcs from a circle. As before, it is sufficient to consider only closed arcs and also only arcs which are not the entire circle.

Theorem 5.3 (Tucker):

G is a circular-arc graph iff its augmented adjacency matrix has the quasi-circular ones property.

Proof =>:

Beginning with any arc, number all of the arcs in clockwise order of their counter-clockwise endpoint. An argument similar to that of theorem 5.1 establishes the fact that $M^*(G)$ has quasi-circular ones when presented in this order.

Proof <=:

Let k_i be the index of the last one in the set U_i. It may be less than i because the sets are allowed to be circular! Assuming that G has n vertices, consider an n-hour clock. Let I_i be the clockwise arc which begins at i o'clock and which ends at k_i o'clock. These arcs form an intersection model for G, using arguments similar to those of theorem 5.1.

QED

Not surprisingly, this last theorem characterizes circular-arc graphs using the circular variant of dense profile. Tucker's theorem actually appeared first[Tuc1] and the corresponding theorem 5.1 was pointed out by Tarjan[Tar2]. Both are modifications of Roberts' result for proper interval graphs. Tucker's second contribution is a circular analog for Roberts' characterization.

Theorem 5.4 (Tucker):

G is a proper circular-arc graph iff its augmented adjacency matrix has the circularly compatible ones property.

Proof =>:

The same construction used in the first half of the proof for theorem 5.3 works here. As before, the fact that the arcs are not properly contained in each other guarantees that the matrix has circularly compatible ones instead of only quasi-circular ones.

Proof <=:

To insure a proper model, slightly extend each arc to end at k_i+2^{i-n-1} o'clock. The same argument used in theorem 5.2 then applies.

QED

For both interval and circular-arc graphs there are linear-time algorithms which test for proper models. These utilize the matrix characterizations just given and the algorithms of chapter 4, both of which are based on reduction. A direct test for dense profile seems difficult. Instead it is more convenient to use an alternate characterization of interval graphs, treat the matrix as a graph, and then use the interval graph test. A test for quasi-circular ones seems much harder. Why this should be true will be explained after looking at the test for interval graphs.

A clique is a completely connected subgraph. Every pair of vertices is adjacent. A dominant clique is a clique which is maximal with respect to set containment. Usually a dominant clique will be writen as $\mathcal{C}$, possibly with subscripting or superscripting. It is easy to see that the dominant cliques completely determine the adjacency relation of a graph since two vertices are adjacent iff they are in a common dominant clique. The adjacency relation can thus be represented by the dominant-clique vs. vertex incidence matrix in which each row

corresponds to a dominant clique and each column to a vertex. Usually this will be called simply the dominant clique matrix for the graph. An entry is nonzero iff its vertex is in the corresponding clique.

Theorem 5.5 (Fulkerson-Gross):

G is an interval graph iff its dominant-clique vs. vertex incidence matrix has the consecutive ones property.

Proof =>:

Consider an interval model for G. Each dominant clique is a family of mutually intersecting intervals. Given a dominant clique $\mathcal{C}$, define

$$l_{\mathcal{C}} = \max \left\{ l \mid [l,r] \in \mathcal{C} \right\}$$

and

$$r_{\mathcal{C}} = \min \left\{ r \mid [l,r] \in \mathcal{C} \right\}.$$

Obviously $l_{\mathcal{C}} \leq r_{\mathcal{C}}$, since otherwise the interval corresponding to $l_{\mathcal{C}}$ would not intersect the interval corresponding to $r_{\mathcal{C}}$, a contradiction of the fact the all of the intervals in $\mathcal{C}$ intersect. An even stronger statement can be made. Any point in $[l_{\mathcal{C}},r_{\mathcal{C}}]$ is common to exactly the intervals in the clique. Order the rows (dominant cliques) of the matrix according to increasing values of $l_{\mathcal{C}_i}$.

Claim:
The matrix has consecutive ones when presented in this order.

Proof of claim:
Consider any column. It represents a vertex which is an interval $I = [l,r]$ in the model. Suppose that the i^{th} and k^{th} entries in the column are nonzero for $i < k$. From the previous remarks, it is clear that both $l_{\mathcal{C}_i}$ and $l_{\mathcal{C}_k}$ are points of I. The order in which rows (cliques) are arranged requires that

$$i < j < k \quad \Rightarrow \quad l_{\mathcal{C}_i} \leq l_{\mathcal{C}_j} \leq l_{\mathcal{C}_k}.$$

This is sufficient to guarantee that l_{C_j} is in I and hence that the j^{th} entry in the column is also nonzero. This is the consecutive ones property.

Proof <=:

If the matrix has consecutive ones, let each vertex correspond to the interval $[l_i, r_i]$ where l_i is the index of the first one in the column and r_i is the index of the last one in the column. Two intervals intersect iff they have a point in common. The common point can always be chosen an integer, and thus two intervals intersect iff their vertices belong to a common clique. As remarked before, the dominant cliques completely characterize the adjacency relation for the graph; the previous statement is thus equivalent to saying that two intervals intersect iff their vertices are adjacent.

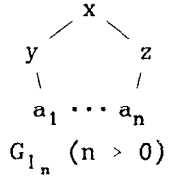
QED

Unfortunately this characterization cannot be extended to include circular-arc graphs by simply substituting circular ones for consecutive ones. The forward implication does not hold. This is because not every dominant clique corresponds to an arc. The construction used in theorem 5.5 implies that there are at most n dominant cliques in an interval graph since there are only n choices for $l_{\mathcal{C}}$ (the value of $r_{\mathcal{C}}$ is determined by $l_{\mathcal{C}}$). If every dominant clique in a circular-arc graph corresponded to an arc, the same result would hold, but Gavril has shown that there may be as many as $2^{n/2}$ dominant cliques[Gav6].

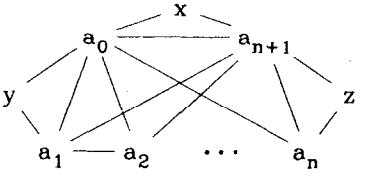
Fulkerson and Gross' theorem for interval graphs was actually the third characterization to appear in the literature. Gilmore and Hoffman[Gil1] earlier proved that interval graphs were properly contained in the complement of the transitively orientable graphs. The earliest appearance of interval graphs, except for the note by Hajós[Haj1], is the characterization given in [Lek1]. Although the next result predates Tucker's structure theorem for consecutive ones, the proof given here uses Tucker's result.

Theorem 5.6 (Lekkerkerker-Boland):

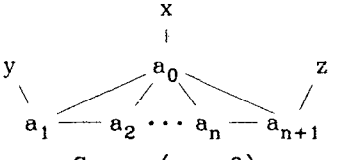
A graph G = (V,E) is an interval graphs iff it has no subgraph isomorphic to one of the following types of graphs:



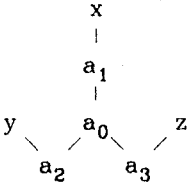
G_{I_n} (n > 0)



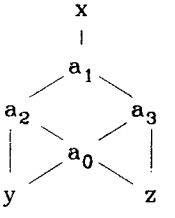
G_{II_n} (n > 0)



G_{III_n} (n > 0)



G_{IV}



G_V

(These are the forbidden subgraphs. As in chapter 1, the vertices x, y, and z represent asteroidal triples: three vertices any two of which are connected by a path to which the third is not adjacent. The first three types are graph schemata, having a parameter n > 0, while the last two are simply graphs.)

Proof =>:

Every subgraph of an interval graph is also an interval graph. None of the fives types of graphs shown above is an interval graph. This is easily verified by computing the dominant-clique matrices and recognizing that they are exactly the forbidden submatrices of theorem 4.16.

Proof <=:

Suppose that some graph having none of the forbidden subgraphs was still not an interval graph. Then by theorem 5.5 its dominant-clique matrix does not have the consecutive ones property. It therefore has one of the forbidden submatrices. The bipartite graph interpretation is handier in this case; one of the

families from theorem 1.18 occurs, where the cliques are atoms (the objects being rearranged) and vertices are the sets.

There are a multitude of cases to consider, but arguments similar to those given in theorem 1.18 show that each instance of a forbidden subfamily implies the existence of one of the forbidden subgraphs given above. This is accomplished by replacing the cliques with the appropriate edges and/or vertices which must be present in G.

QED

Theorem 5.6 is due to Lekkerkerker and Boland. Tucker proved a similar structure theorem for proper circular-arc graphs as part of his thesis[Tuc1]. The subgraphs involved were more complex than these. Roberts had earlier proved a proper interval graph structure theorem in his thesis[Rob1]. The theorem for general circular-arc graphs is left as an exercise for the reader's thesis.

II Chordal Graphs

The Fulkerson-Gross algorithm for identifying interval graphs has two parts. It first finds the dominant cliques and then determines if the resultant dominant-clique vs. vertex incidence matrix has the consecutive ones property. It turns out that the size of this matrix is no larger than the size of the adjacency matrix. Even better, the number of nonzero entries is bounded above by the total number of edges in the graph. Thus the second phase is easily handled in linear time using the techniques of chapter 4.

Calculating the dominant cliques is also easy. Interval graphs are a special case of a class of graphs which have a particularly convenient dominant clique structure. These have been called chordal graphs, rigid circuit graphs, triangulated graphs, acyclic graphs, zero fill-in graphs, and perfect elimination graphs. Some of these names are confusing since they conflict with other graph properties. The term chordal will be used here exclusively, partially because it is widely found in the literature, but primarily because it has no other commonly

accepted meaning.

A graph is chordal iff every cycle of length greater than three has a chord. A chord is an edge joining two vertices which are not connected by an edge of the cycle. Fulkerson and Gross[Ful1], and earlier Lekkerkerker and Boland[Lek1], have given algorithms which effectively test a graph for chordality. Gavril was able to improve on these by reducing the question of chordality to a matrix multiplication, producing an $O(n^{log_2 7})$ recognition algorithm[Gav4]. This too can be improved upon.

An ordering α for the vertices of a graph G = (V,E) is a perfect elimination ordering iff it satisfies the property that, given a vertex i and any two vertices j and k which are adjacent to i, if

$$\alpha(i) < \alpha(j) \qquad \text{and} \qquad \alpha(i) < \alpha(k)$$

then j and k must be adjacent to each other. The explanation of why this is called a perfect elimination ordering requires a knowledge of Gaussian elimination.

Given a symmetric matrix M, the zero / nonzero structure represents an adjacency relation for a graph whose vertices are the rows (or columns) of the matrix. If the graph has a perfect elimination ordering performing the elimination steps on the matrix in the order given by α will guarantee that no zero element of the matrix becomes nonzero as the result of an elimination step. This explains why the name zero fill-in is applied to graphs whose adjacency matrices have this property. Numerical stability might be a problem, but if the matrix is also positive definite this can be safely ignored, at least with respect to the order of elimination.

Zero fill-in is a distinctly different definition than chordality. Rose has shown that the two conditions define exactly the same class of graphs[Ros1]. The proof of this fact requires some preliminaries.

An algorithm for testing chordality exists which is a single-pass search of the graph. Not any search will do, but a particular type does work.

The one used here will be a lexicographic breadth-first search. This idea was independently discovered by Lueker[Lue1] and also Rose and Tarjan[Ros2]. It has been used to solve processor scheduling problems[Cof1]. The search proceeds by assigning a sequence of labels to the vertices of the graph and then using the labels to govern the order in which vertices are visited. At each step the vertex with the largest label is chosen. Largest means dictionary order, where longer labels are greater than their proper prefixes. The numbering scheme is outlined in the next algorithm.

Lexicographic Breadth-First Search

[1] All vertices are unnumbered. Label each vertex with the null string. This is always the smallest label. Initialize k to n, the number of vertices in G.

[2] Of all unnumbered vertices having a largest label, choose one and assign it the number k.

[3] Append k to the label of every unnumbered vertex to which vertex k is adjacent in G. Also append k to the label for the vertex numbered k.

[4] Decrement k and, if it is nonzero, go back to step 2. Otherwise halt.

The important property of this numbering is that the vertices are processed in exactly the order (actually the reverse order) in which they should be eliminated in a perfect elimination scheme. The reason that this is true becomes apparent after investigating a few elementary properties of lexicographic breadth-first search, especially as applied to chordal graphs. The first is easy to verify given the definition of a search.

Lemma 5.7:

Lexicographic search labels each vertex with a list of the adjacent vertices having larger numbers plus its own number. The numbers in a label are sorted from largest to smallest.

The second lemma is a bit more complicated, but it is exactly what is needed to characterize chordal graphs.

Lemma 5.8 (Tarjan):

Let G = (V,E) be any graph numbered by lexicographic breadth-first search. If j and k are vertices whose numbers are larger than i which are both are adjacent to i then there is a path in G between j and k all of whose internal vertices are both nonadjacent to i and numbered larger than i.

Proof:

Suppose that there exist vertices numbered i, j, and k satisfying the hypothesis but not the conclusion. Let

$$\{j_1,\cdots,j_r\} \qquad \text{and} \qquad \{k_1,\cdots,k_s\}$$

be the vertices having paths to j or k, respectively, which are nonadjacent to i and pass only through vertices numbered greater than i. Without loss of generality, assume that these are in decreasing order.

$$j_1 > \cdots > j_r \qquad \text{and} \qquad k_1 > \cdots > k_s.$$

Note that each of the vertices is necessarily greater than i. Using these, a contradiction will be derived.

Claim 1:
For any choice of p and q the two vertice j_p and k_q are not only not equal, they are nonadjacent in G.

Proof of claim 1:

If one of the j_p were also one of the k_q then there would be a path, starting with j and ending with k, which had no internal vertices either adjacent to or numbered less than i. This is contrary to the choice of j and k. A similar argument precludes j_p and k_q being adjacent. □

Consider the series of labels which are assigned by the lexicographic breadth-first search. Initially, all of the j_p, all of the k_q, and i have empty labels. Look at the first time one of these labels is changed by some vertex l, not one of the j's or k's, being numbered.

Claim 2:

The vertex l is added to the label of i and every one of the j_p and k_q.

Proof of claim 2:

If l were not adjacent to i it would itself be one of the j_p or k_q. It isn't. Thus it is adjacent to i. But i gets numbered after all of the others, so l must also be added to every one of their labels in order to keep those labels at least as large as the label for i. □

This second claim is equally valid for each subsequent vertex which is numbered. Numbering the vertex either changes all of the labels for the j's and k's or else it changes none of their labels. Thus when the first j_p or k_q receives a number, all of the labels are still identical. The two cases are symmetric, so assume that k_1 gets numbered first.

From the first claim it is clear that no label of any j_p is changed. But the label of some k_q is, because there is a path to k from k_1, passing through higher-numbered k's. Thus k_q will be numbered before any of the j_p. But k_q is itself adjacent to some k_q, which will in turn be numbered before any of the j_p. This continues until eventually k and then i is numbered, all before any j_p. This contradicts the fact the i is numbered last.

QED

So far none of the properties of chordal graphs have been used. Now is the time. This last lemma, in the special case of a chordal graph, is enough to guarantee that the lexicographic numbering is in fact a perfect elimination ordering.

Theorem 5.9 (Rose):

A graph G = (V,E) is chordal iff it has a perfect elimination ordering.

Proof =>:

Suppose that G is chordal. Number the vertices according to lexicographic breadth-first search. Lemma 5.8 can be restated in a stronger form.

Claim:
If i < j < k and j and k are both adjacent to i, then they are also adjacent to each other.

Proof of claim:
Using lemma 5.8 there must be a path, nonadjacent to i, whose internal vertices connect j and k and are all numbered higher than i. Choose a minimal path which satisfies this condition. Add the edges {i,j} and {i,k} to form a cycle. The cycle obviously has no chord, because the path is minimal and i is only adjacent to j and k, by assumption. But if the cycle has length greater than three, the graph could not be chordal. This implies that the path in question actually is a single edge and hence that j and k are adjacent in G. □

The condition stated in the claim is exactly the definition of a perfect elimination ordering for G

Proof <=:

Suppose that G has a perfect elimination ordering. Consider any cycle of length greather than three. Select i to be the lowest numbered vertex in the cycle and let j and k be the two immediately adjacent vertices in the cycle. The perfect elimination condition requires that they be adjacent, and hence a chord.

QED

The last proof showed that any lexicographic breath-first numbering of a chordal graph is a perfect elimination ordering. The converse is not true. There are perfect elimination orderings which cannot be realized by any lexicographic breadth-first search of the graph. For the purpose of recognizing chordal graphs, this is no drawback. Not all perfect elimination orderings are required, just one. A bit more on chordal graphs is still needed before returning to interval graphs.

The next theorem characterizes chordal graphs as being exactly those graphs whose intersection models can be constructed from a tree. Here tree means a topological tree, where each vertex of the graph corresponds to a connected subset of the tree, which is a subtree in the topological sense. Two points are worth noting: every interval graph has a model of this type because a collection of intervals is a linear tree; secondly, just as the interval model can be constructed from dominant cliques, the theorem will construct a tree model whose nodes are the dominant cliques of the graph.

The proof given here is different from the one in [Gav2]. Gavril's construction is based upon an $O(n^4)$ algorithm for recognizing chordal graphs. The construction here makes use of theorem 5.9. This leads to a new proof of the characterization and a linear-time algorithm for constructing the intersection model.

Theorem 5.10 (Gavril):

G is a chordal graph iff it is the intersection graph of subtrees in a tree.

Proof =>:

Number the vertices of G using lexicographic breadth-first search. A label is maximal if it is not a prefix of any other label. Define a graph $T_G = (V_G, E_G)$ whose vertex set is the set of maximal labels. If l_i and l_j are each maximal labels, let l_i be the lexicographically smaller of the two. Let k be the lowest numbered vertex of l_i which is in some other lexicographically larger maximal label. If k does not exist, the labels l_i and l_j are not adjacent; otherwise, l_i is adjacent to l_j iff l_j is the lexicographically largest maximal label containing k.

Claim 1:
The maximal labels are exactly the dominant cliques of G.

Proof of claim 1:
The lexicographic breadth-first numbering is a perfect elimination ordering. This immediately implies that the labels are cliques; every vertex is adjacent to the lowest numbered vertex in the label and all of the higher numbered vertices must be pairwise adjacent according to the perfect elimination property.

Dominance of the cliques is easy to demonstrate. Maximality of the labels implies that no label is a prefix of another. It might still be possible for the set of vertices in one label to be contained within the set of vertices in another label, without the second label being a prefix of the first. It is easy to show that this is actually impossible. Suppose that some label l_1 contains all of the vertices in l_2 without l_2 being a prefix of l_1. Let i be the lowest numbered vertex in l_1 and j the lowest numbered vertex in l_2. Necessarily i < j because j is by assumption in l_1. In order for the containment to hold l_1 would have to have some vertex k > j which was not in l_2, otherwise l_2 is a prefix of l_1.

But this is a contradiction because obviously j and k are adjacent (they are in the clique l_1) and hence k must be in the label for j, which is the clique l_2.

All that remains of the claim is to verify that every dominant clique occurs as a label. Consider any clique at all. Choose the lowest numbered vertex in the clique, call it i. From lemma 5.7 it is clear that its label includes all of the vertices of the clique because the label contains all of the higher numbered vertices adjacent to i. By definition either this label or some properly containing label is included in V_G. □

Claim 2:
The graph T_G is a forest.

Proof of claim 2:
This is immediate. From the definition it is obvious that any label l_i is adjacent to at most one lexicographically larger label. The new graph, T_G, is thus acyclic because the lexicographically smallest label in a cycle has to be adjacent to two distinct lexicographically larger labels. □

Arbitrarily connect the trees in the forest to obtain a a single tree. This is just a nicety to handle graphs which are disconnected. Define a family of subgraphs by the following.

$$\mathcal{S}_G = \left\{ T_i \mid i \in V=\{1,2,\cdots,n\} \right\}$$

where T_i is the subgraph of T_G induced by the labels (dominant cliques) to which i belongs.

Claim 3:
Each T_i is a tree.

Proof of claim 3:
It is sufficient to show that for every vertex j all of the dominant cliques to which it belongs are connected in T_G. Consider the order in which lexicographic breath-first search

numbers the vertices. The first maximal label to be created presents no problem. There are no previously created labels to which it must be connected. An informal induction can then be performed on the remaining cliques as they are added. As each subsequent maximal label is found, let j be any vertex in the new label. If j appears in no previously created maximal label there is nothing to prove. Otherwise let i be the lowest numbered vertex within the newly created clique such that i is in some previously created clique.

From the way the edges E_G were chosen, the new clique must be adjacent to the lexicographically largest label of which i is a member. The lexicographic order insures that this will contain the largest label assigned to i as a prefix. Note that the choice of i guarantees $i \leq j$. Vertex i is adjacent to j (the new label is a clique) hence j is in the label of i and thus a member of the previously created clique to which the new clique is connected. By assumption the earlier clique is connected to all of the other cliques to which j belongs.

Vertices which are not in the new maximal label are still in a connected set of labels since no edges are deleted from E_G when a new maximal label is found. □

Claim 4:

$\mathcal{S}_G$ is an intersection model for G

Proof of claim 4:

Two vertices are adjacent in G iff they are in a common dominant clique. Two subtrees intersect iff they have a common node. The nodes are exactly the dominant cliques to which the vertices belong. □

Proof <=:

Suppose that G is a graph having an intersection model which is composed of subtrees from some tree. Choose any cycle in G of length greater than 3. The cycle corresponds to a path in the

tree which cyclically visits each subtree. An edge in the cycle is a point in the tree which is in two subtrees adjacent in the cycle. A chord for the cycle is a "shortcut" in the tree which eliminates mention of at least one of the subtrees; the subtrees at each end of the chord share a common point. A cyclic path with no "shortcuts" is thus a cycle in the tree. This is a contradiction.

QED

This result shows that a slight modification to lexicographic breadth-first search is sufficient to test for chordality and also to find the dominant cliques.

Theorem 5.11 (Lueker-Rose-Tarjan):

A graph G = (V,E) can be tested for chordality in O(n+e) steps and if it is chordal, the dominant cliques can be listed at no additional cost.

Proof:

The lexicographic search is easily implemented to run in linear time. This is shown in [Lue1] and [Ros2]. The test for chordality consists of verifying that each label is a dominant clique. Both Lueker, and Rose and Tarjan show that this too can be accomplished in O(n+e) steps because it is sufficient to check that each vertex in a label is adjacent to the second lowest numbered vertex in the label.

The number of edges which must be checked is O(e) since each such edge can be paired with the corresponding edge to the lowest numbered edge in the label, and each of these is counted exactly once when summed over all of the labels.

QED

It also is possible to maintain all of the dominant cliques and their

interconnections within T_G at no extra expense. Thus the lexicographic breadth-first search makes possible more than just a linear recognition algorithm for chordal graphs; it also effectively constructs an intersection model.

The analogy between theorem 5.10 and theorem 5.5 should be obvious. The intersection model constructed for interval graphs is exactly the same as the intersection model for chordal graphs, except that the dominant cliques are arranged linearly. Straightening the tree out is all that has to be done in order to get an interval model.

The Fulkerson-Gross characterization for interval graphs can be combined with the efficient chordality test and then topped off with pruned tree reduction for the straightening.

Theorem 5.12 (Booth-Lueker):

Given a list of its e edges a graph G can be tested for being an interval graphs in O(n+e) steps.

Proof:

The algorithm is easy. Using lexicographic breadth-first search compute the dominant cliques and build a nonzero-entry list for the dominant-clique matrix. This takes O(n+e) steps. Note that each nonzero entry corresponds to an edge of the original graph. This matrix can be tested for consecutive ones in O(n+e) steps.

QED

Using the characterization of theorem 5.1 this yields an efficient, if roundabout, test for dense profile.

Corollary 5.13:

Given a list of its e nonzero entries, an m×m (0,1) matrix M can be tested for dense profile in O(m+e) steps.

When the graph is specified by an adjacency matrix instead of an edge list the bound is not as good, but it is still a linear function of the size of the input.

Corollary 5.14:

An m×m matrix can be tested for dense profile in $O(m^2)$ steps.

Both of these tests, one for perfect elimination and the other for dense profile, are useful in numerical analysis. Symmetric positive definite matrices have the property that any elimination order is stable. Thus an algorithm is free to choose one which is convenient. As already noted, a perfect elimination ordering will have zero fill-in. This means that storage can be minimized for sparse matrices because only those entries initially present need be stored.

This unstructured sparseness requires a list-processing approach. It may be more convenient to store the columns (or rows) as short vectors. This can be done economically if the matrix has dense profile because again only the elements initially present will ever be required. The only drawback to either of these schemes is that many matrices do not have perfect elimination orderings or dense profile. As might be expected, the problem of obtaining good approximations is difficult.

Rose and Tarjan have shown[Ros2] that the problem of finding a minimum fill-in ordering is NP-complete for the directed case (non-symmetric matrices). The status of the undirected (symmetric) case is at present open. It is also unknown whether finding a minimum augmentation to achieve dense profile is NP-complete, although it is suspiciously similar to the problems analyzed in chapter 4. A closely related problem, minimum bandwith, has been shown to be NP-complete by Papadimitriou[Pap1]. These negative results suggest that practical solutions for numerical applications will have to rely on heuristic techniques.

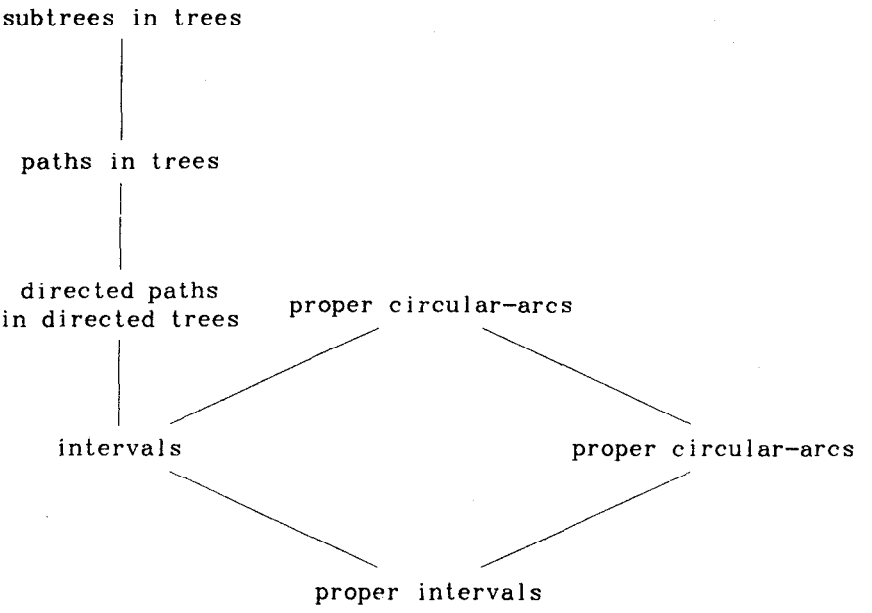
III Some Final Topics

Perhaps the most interesting application for the interval graph test has been saved for next-to-last. Lueker has pointed out[Bth1, Lue2] that the PQ-tree which results from testing the dominant-clique matrix for consecutive ones is a canonical representation of the interval graph. A linear-time isomorphism test for interval graphs is thus an easy modification of the tree isomorphism algorithm given in [Aho1]. Lueker also showed that general graph isomorphism is equivalent to isomorphism for chordal graphs. This means that if a linear-time test exists for chordal graph isomorphism then general graph isomorphism would also be linear. It is tempting to simply extend Lueker's result to the case of chordal graphs.

The intersection model given by theorem 5.10 is not, unfortunately, a canonical form for the chordal graph. The nodes of the tree are always the same, but the edges are a function of the order in which the vertices of G are explored and there is a degree of indeterminacy in the lexicographic search.

Another alternative is to examine graphs of intermediate complexity between interval and chordal graphs. Gavril[Gav3] and Renz[Ren1] have both done this. Instead of looking at subtrees within a tree, restrict each member of the intersection model to be a path within the tree. Graphs of this type form a proper subclass of the chordal graphs. An even stronger restriction, but one which is still more general than the interval graphs, is found by examining the intersection graphs of directed paths within directed trees.

All of the classes of intersection models mentioned in this chapter can be represented in a diagram of their partial ordering under set inclusion.



Proper interval graphs are the most restrictive. Notice that above interval graphs there is no notion of a proper intersection model. Every model can be chosen proper. Some of the earlier examples have indicated that some of the class containments shown above are proper. In actual fact, all of them are proper.

The four types of intersection graphs, interval, directed-path, path, and subtree form a four-class hierarchy, each properly contained within the other. In all cases the intersection models can be chosen as graphs having the dominant cliques as nodes. Gavril[Gav3] has given algorithms which recognize each of the middle two types of graphs. Both of his algorithms are based on the Gilmore and Hoffman algorithm[Gil1] for recognizing interval graphs. Their test does not run in linear time, but it is still polynomial and it has some properties which are convenient for the extensions being tested.

The final two results are similar in spirit to theorems 3.7 and 4.6. Planarity is a monotone graph property, and hence a special case of Rivest and Vuillemin's result. As was remarked before, consecutive ones is not a monotone property; neither is chordality, nor the property of being an interval graph. *Ad hoc* arguments can be concocted, however.

Theorem 5.15:

Let $n \geq 4$. Any algorithm which correctly tests a graph having n vertices for chordality, using only the adjacency matrix as input, must examine $O(n^2)$ entries.

Proof:

Consider the complete graph K_n having n vertices. It is its own dominant clique, every vertex being adjacent.

Claim:
Every correct algorithm must examine all but at most n-1 of the edges of K_n.

Proof of claim:
Suppose n edges exist which were not checked by the algorithm. Restrict attention to the subgraph induced by the unexamined edges. If the edges form more than one component choose any two edges from different components. These have distinct endpoints. Otherwise there is a vertex i having a maximum number of incident unexamined edges. All together these can account for at most n-1 edges.

There must be some other vertex, say j, adjacent to i and also incident with some other unexamined edge. Let l be the other vertex for this edge. If l is also adjacent to i then the three form a triangle and there must be at least one more edge incident with i. This gives a fourth vertex k. The edges $\{i,k\}$ and $\{j,l\}$ have distinct vertices.

If l is not adjacent to i then there is still some other vertex k which is adjacent to i. This again gives vertex-disjoint edges $\{i,k\}$ and $\{j,l\}$. The claim is thus complete because the cycle (i,j,k,l) in K_n cannot have been checked for a chord because neither of the two edges was examined! The algorithm thus cannot differentiate the complete graph from one which is not chordal. □

QED

An identical proof works for interval graph testing because K_n is not only chordal, it is an interval graph.

Theorem 5.16:

Let $n \geq 4$. Any algorithm which correctly tests a graph having n vertices for being an interval graph, using only the adjacency matrix as input, must examine $O(n^2)$ of the entries

It would be interesting to investigate the complexity of graph isomorphism for each of these intermediate families and then see if general graph isomorphism can be reduced to one of them.

This chapter has discussed a number of topics, all related to representing graphs by intersection models. Seven classes of intersection models were examined, each characterizing a class of graphs. Some of the classes were shown to be equivalent to matrix properties introduced in chapter 4, if graphs are represented by their augmented adjacency matrices. Algorithms for recognizing some of these were given, although the case of circular-arc graphs appears to be much harder than the others.

A number of open problems, notably the complexity of chordal graph isomorphism and the minimum augmentation to achieve dense profile, deserve further study.

Summary

The previous chapters have presented a new data structure called PQ-Trees and a set of algorithms for manipulating them. These have been applied to two algorithms which have appeared in the literature - Lempel, Even, and Cederbaum's planarity test and Fulkerson and Gross' interval graphs test. In both cases the algorithms have been speeded up so as to run in a time linear with the size of the input.

The correctness proofs for the algorithms and the time bounds refer to a number of properties of PQ-formulas. The PQ-formulas are a linear representation of PQ-trees. Each tree is an equivalence class of formulas. The equivalence classes, and hence by analogy the trees, form an algebraic lattice. This idea is developed in chapter 1 along with a structure theorem for deciding if a formula (or tree) is reducible. The concept of the reducibility of a formula with respect to a set is the property which is exploited in both the graph applications.

The tree representation of PQ-formulas enables efficient algorithms to be constructed for reduction and hence for the two recognition problems. This is discussed in chapter 2. The exact details of the implementation are in the appendix, which follows.

Chapter 3 is short. A few definitions from graph theory are reviewed. An implementation of Lempel, Even, and Cederbaum's graph planarity algorithm is then given; it runs in O(n) steps for a graph having n vertices. The algorithm is illustrated with an example.

Chapter 4 is somewhat longer. The consecutive ones property for matrices is examined and it is shown that the reduction algorithm is equivalent to a test for consecutive ones. The chapter continues with a discussion of some variations on this theme including some related NP-complete problems. The consecutive ones property is used as a part of the interval graph test in chapter 5.

Interval graphs are an example of a type of intersection model, as are circular-arc graphs and chordal graphs. Various algorithms for recognizing these classes of graphs are discussed. Using the efficient consecutive ones test and an efficient chordality test of Lueker, Rose, and Tarjan, the Fulkerson-Gross test for interval graphs is speeded up to run in O(n+e) steps for a graph having n vertices and e edges.

There are three basically new results: the algebraic formulation of PQ-formulas, the PQ-tree data structure with its transformations, and the linear running times for the two graph recognition algorithms.

Some of the ideas presented here are also explained in a 16mm color movie which is available from the author. Persons interested in obtaining a copy should contact me at the address below.

Kellogg Booth
Mail Stop L-73
Lawrence Livermore Laboratory
P. O. Box 808
Livermore, CA 94550
(415) 447-1100 Ext. 3359

Thanks for reading.

Appendix

The pruned tree reduction algorithm of chapter 2 has been implemented. The program runs on a CDC 7600. It is written in LLLTRAN, a variant of FORTRAN used at Lawrence Livermore Laboratory. A listing of the program and an example of its use are both on the next page. The information is on microfiche because of its volume.

Appendix

Microfiche Listing

Bibliography

The bibliography contains a reference for each of the works cited in the text. A number of these were pointed out to me by George Lueker. There are surely other articles which I have missed, but this list should provide a good start for anyone wishing to further pursue the ideas presented here.

[Aho1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, **The Design And Analysis Of Computer Algorithms**, Addison-Wesley, Reading, Massachusetts (1974).
This is good reading for a general background. Many of the basic concepts used in this thesis are explained: tree data structures, graphs, NP-complete problems, and general methods of analyzing algorithms.

[Bak1] K. A. Baker, P. C. Fishburn, and F. S. Roberts, "Partial Orders Of Dimension 2," **Networks**, 2 (1972) 11-28.
Partial orders of dimension 2 are equivalent to comparability graphs. These in turn were related to interval graphs by Gilmore and Hoffman in [Gil1].

[Ben1] S. Benzer, "On The Topology Of The Genetic Fine Structure," **Proceedings Of The National Academy Of Science, U.S.A., 45** (1959) 1607-1620.
This is the first application of interval graphs to appear in the literature. Benzer outlines the advantages of a linear model for genes and how recombining tests on mutants can be used to verify the models. Although the techinique is not clearly explained, it is essentially an interval graph test.

[Ben2] S. Benzer, "The Fine Structure Of The Gene," **Scientific American, 206**:1 (1962) 70-84.
This is a popularized version of the previous reference.

[Bes1] M. R. Best, P. van Emde Boas, and H. W. Lenstra, Jr., "A Sharpened Version Of The Aanderaa-Rosenberg Conjecture," preprint (July 1974) 1-12.
Certain classes of graph properties are shown to satisfy the Aanderaa-Rosenberg Conjecture. These are called evasive properties; algorithms for testing evasive properties require $O(n^2)$ probes of the adjacency matrix if the graph in question has n vertices. Graph planarity is shown to be evasive.

[Bth1] K. S. Booth and G. S. Lueker, "Linear Algorithms To Recognize Interval Graphs And Test For The Consecutive Ones Property," **Proceedings Of The Seventh Annual ACM Symposium On Theory Of Computing**, (1975) 255-265.
A preliminary sketch of the authors' joint work. Primarily discussing interval graph testing, the general consecutive ones property and graph planarity are only mentioned. The material is more thoroughly treated in the present thesis and in Lueker's thesis[Lue2].

[Bth2] K. S. Booth and G. S. Lueker, "PQ-Trees," submitted to **Journal Of Computer And Systems Sciences.**

[Cof1] E. G. Coffman Jr. and R. L. Graham, "Optimal Scheduling For Two-Processor Systems," **Acta Informatica**, 1 (1972) 200-213.
An idea very similar to lexicographic breadth-first search is used to solve some scheduling problems.

[Coh1] J. Cohen, "Interval Graphs And Food Webs," The RAND Corporation, D-17696-PR.
This reference was provided by Lueker. He informs me that RAND considers this to be an internal document, not for distribution. Try writing to Cohen directly or to Daniel Ellsberg.

[Coo1] S. Cook, "The Complexity Of Theorem-proving Procedures," **Proceedings Of The Third Annual ACM Symposium On Theory Of Computing**, (1971) 151-158.
Cook's theorem about about the satisfiability of formulas in conjunctive normal form appears here. It is the basis for Karp's definition of NP-complete problems[Kar1].

[Dir1] G. A. Dirac, "On Rigid Circuit Graphs," **Abhandlungen aus dem Mathematishen Seminar der Universitat Hamburg** 25 (1961) 71-76.
This is the origin of the term rigid circuit graph as an synonym for chordal. Some coloring properties of chordal graphs are analyzed and the problem of augmenting a graph with edges to achieve chordality is considered from the standpoint of chromatic number.

[Dus1] B. Dushnik and E. W. Miller, "Partially Ordered Sets," **American Journal Of Mathematics**, **63** (1941) 600-610.
The concept of the dimension of a partial order is defined here. Many of the results concern partial orders of transfinite cardinality; these are not very interesting from an algorithmic viewpoint.

[Esw1] K. P. Eswaran, "Consecutive Retrieval Information Systems," ERL-M384, Electronics Research Laboratory, University of California, Berkeley (May, 1973) 1-113.
This is Eswaran's Ph.D. thesis. Using Ghosh's concept of the consecutive retrieval property he analyzes a number of ways to achieve efficient access characteristics. Most are related to the consecutive ones property. The implementation suggested requires $O(n^3)$ steps.

[Esw2] K. P. Eswaran, "Faithful Representation Of A Family Of Sets By A Set Of Intervals," **SIAM Journal On Computing**, 4:1 (1975) 56-68.
The title is misleading. The representation referred to is not an intersection model, which would imply an interval graph, but it is merely another use of the consecutive ones property. All of the material appears in Eswaran's thesis[Esw1].

[Eve1] S. Even, and R. Tarjan, "Computing An st-Numbering," to appear in **Theoretical Computer Science**.

[Eve2] S. Even, private communication.
Even has implemented a linear version of the Lempel, Even, and Cederbaum planarity algorithm. It does not directly mirror the PQ-formulas, as originally defined in [Lem1]. The program was never actually published.

[Ful1] D. R. Fulkerson and O. A. Gross, "Incidence Matrices And Interval Graphs," **Pacific Journal Of Mathematics**, 15:3 (1965) 835-855.
Interval graphs are characterized as those graphs whose dominant-clique vs. vertex matrices have the consecutive ones property. This is the first appearance of consecutive ones in the literature. The Fulkerson-Gross algorithm has two parts: find the dominant cliquesand then check the matrix for consecutive ones. Both require $O(n^3)$ steps.

[Gar1] M. R. Garey, D. S. Johnson, and L. Stockmeyer, "Some Simplified NP-Complete Problems," **Proceedings Of The Sixth Annual ACM Symposium On Theory Of Computing**, (1974) 47-63.
A number of NP-complete problems are presented, many of them simplified forms of known NP-complete problems. Simple optimal linear arrangement is among these. The article is a good review of the proof techniques used to establish NP-completeness.

[Gav1] F. Gavril, "Algorithms for Minimum Coloring, Maximum Clique, Minimum Covering by Cliques, And Maximum Independent Set Of A Chordal Graph," **SIAM Journal On Computing**, 1-2 (June 1972) 180-187.
All of the indicated problems are NP-complete for arbitrary graphs. The clique structure of chordal graphs allows more efficient algorithms. Most of Gavril's bounds can be improved using the linear chordality test of [Lue1] or [Ros2].

[Gav2] F. Gavril, "The Intersection Graphs Of Subtrees In Trees Are Exactly The Chordal Graphs," **Journal Of Combinatorial Theory**, 16-1 (February 1974) 47-56.
A complete characterization of chordal graphs by intersection models is given. Generalizing from interval graphs, the set is taken to be an undirected tree and the family of subsets is a collection of subtrees. Gavril obtains an $O(n^4)$ algorithm which constructs a tree and thus tests for chordality.

[Gav3] F. Gavril, "A Recognition Algorithm For The Intersection Graphs Of Directed Paths In Directed Trees," manuscript (November, 1973) 1-23.
Two generalizations of interval graphs, both properly contained within the chordal graphs, are examined.

[Gav4] F. Gavril, "An Algorithm For Testing Chordality Of Graphs," **Information Processing Letters**, 3:4 (March 1974) 110-112.

A chordality test which is essentially matrix multiplication is explained. Using Strassen's technique the $O(n^{\log_2 7})$ bound is easily shown.

[Gav5] F. Gavril, "Algorithms On Circular-Arc Graphs," manuscript (January, 1974) 1-21.
Two variations of circular-arc graphs, α-circular-arc and β-circular-arc graphs, are defined. The recognition problem is then solved for these two cases. No solution is given for the general case.

[Gav6] F. Gavril, private communication.

[Gho1] S. P. Ghosh, "File Organization: Consecutive Storage Of Relevant Records On A Drum-type Storage," **Information And Control**, 25-2 (1974) 145-165.
The consecutive retrieval property, an equivalent formulation of the consecutive ones property, is defined for query systems.

[Gho2] S. P. Ghosh, "Consecutive Storage Of Relevant Records With Redundancy," **Communicatons Of The ACM**, 18-8 (August 1975) 464-471.
Minimum redundancy solutions are found for some special cases of query systems. Kou's result (theorm 4.22) shows that the general solution is NP-complete.

[Gho3] S. P. Ghosh, "File Organization: The Consecutive Retrieval Property," **Communications Of The ACM**, 9 (September 1972) 802-808.

[Gho4] S. P. Ghosh, "On The Theory Of Consecutive Storage Of Relevant Records," **Journal Of Information Sciences**, 16 (1973) 1-9.

[Gil1] P. C. Gilmore and A. J. Hoffman, "A Characterization Of Comparability Graphs And Of Interval Graphs," **Canadian Journal Of Mathematics, 16** (1964) 539-548.
Using the idea of a transitive orientation for a graph, an algorithm for testing interval graphs is obtained. It is $O(n^3)$, using the test in [Pnu1]. This is the interval graph algorithm used by ↑gavril in [Gav3].

[Haj1] G. Hajós, "Uber eine Art von Graphen," **Internationale Mathematische Nachrichten, 11** (1957) 65.
The first suggestion in the literature of interval graphs, Hajós poses the question of characterizing the intersection graphs of families of intervals.

[Har1] F. Harary, **Graph Theory**, Addison-Wesley, Reading, Massachusetts (1969).
A general background for most of the graph-theoretic concepts used in this thesis.

[Hop1] J. E. Hopcroft and R. E. Tarjan, "Efficient Planarity Testing," **Journal Of The ACM, 21** (1974) 549-568.
This is a linear test for graph planarity. It computes the biconnected components of the graph as a first step.

[Jen1] G. Jennings, "Representation Of A Collection Of Finite Sets As Intervals On A Line," manuscript, 1-11.
This work parallels some of the developments in [Ful1]. An $O(n^3)$ algorithm is obtained for testing consecutive ones. The idea of computing the closure with respect to operations on normal sets is presented.

[Kar1] R. M. Karp, "Reducibilitiy Among Combinatorial Problems,", in R. Miller and J. Thatcher (eds.), **Complexity Of Computer**

Computations. Plenum (1972) 85-104.
NP-complete problems are defined, using Cook's result on satisfiability[Coo1]. A number of examples are shown to be NP-complete.

[Kar2] R. M. Karp, "On The Computational Complexity Of Combinatorial Problems," **Networks**, 5 (1975) 45-68.
More examples of NP-complete problems. This is a good survey of the subject.

[Ken1] D. G. Kendall, "Incidence Matrices, Interval Graphs And Seriation In Archaeology," **Pacific Journal Of Mathematics**, 28:3 (1969) 565-570.

[Ken2] D. G. Kendall, "Some Problems And Methods In Statistical Archaeology," **World Archaeology**, 1 (1969) 68-76.
The consecutive ones property is applied to sequence dating.

[Kle1] V. Klee, "What Are The Intersection Graphs Of Arcs In A Circle?," **American Mathematical Monthly**, 76:7 (1969) 810-813.
This is a short note suggesting further possibilities for intersection graphs.

[Kou1] L. T. Kou, "Polynomial Complete Consecutive Information Retrieval Problems," to appear in **SIAM Journal On Computing**.
Two extensions of the consecutive ones property: minimizing the number of consecutive blocks of ones appearing in columns and minimizing the number of times a row must be split into two pieces to attain consecutive ones, are shown to be NP-complete. It is suggested that in light of these results heuristic methods must be developed for attaining close approximations to consecutive ones.

[Kur1] K. Kuratowski, "Sur Le Problèm Des Courbes Gauches En Topologie," **Fundamenta Mathematicae**, 15 (1930) 271–283.
This is Kuratowski's famous structure theorem for planar graphs.

[Lek1] C. G. Lekkerkerker and J. Ch. Boland, "Representation Of A Finite Graph By A Set Of Intervals On The Real Line," **Fundamenta Mathematicae**, 51 (1962) 45–64.
Citing the earlier work of Benzer as a motivation, Lekkerkerker and Boland present the first structural characterization of interval graphs. Using the idea of an asteroidal triple they show that a graph is an interval graph iff it does not contain an induced subgraph which is one of five forbidden types. They present an $O(n^4)$ algorithm which finds asteroidal triples.

[Lem1] A. Lempel, S. Even, and I. Cederbaum, "An Algorithm For Planarity Testing Of Graphs," in P. Rosenstiehl (ed.), **Theory Of Graphs: International Symposium, July, 1966**, Gordon and Breach, New York (1967) 215–232.
An algorithm is presented for testing a biconnected graph for planarity. No bound is claimed other than that it is "efficient to implement." The test is carried out as a series of symbolic manipulations on formulas.

[Lue1] G. S. Lueker, "Structured Breadth First Search And Chordal Graphs," Technical Report TR–158, Computer Science Laboratory, Department of Electrical Engineering, Princeton University (August, 1974) 1–30.
Both of Lueker's papers were written before he was aware of either [Ros2] or the present thesis. This paper gives a linear test for chordality.

[Lue2] G. S. Lueker, "Interval Graph Algorithms," Ph.D.. thesis, Princeton University (August, 1975).
With minor differences, Lueker has defined the same PQ–trees as

are defined here. He also presents essentially the same algorithm for reduction. It is applied to a linear test for interval graphs along with the linear chordality test.

[Lue3] private communication.

[Pap1] C. H. Papadimitriou, "The NP-Completeness of the Bandwidth Minimization Problem," Technical Report TR-173, Computer Science Laboratory, Department of Electrical Engineering, Princeton University (December 1974) 1-13.
It is noted that the k-augmented consecutive ones property is NP-complete. A proof based on the simple optimal linear arrangement of [Gar1] is suggested.

[Pnu1] A. Pnueli, A. Lempel, and S. Even, "Transitive Orientation Of Graphs And Identification Of Permutation Graphs," **Canadian Journal Of Mathematics**, 23 (1971) 160-175.
An $O(n^3)$ algorithm for transitively orienting a graph is is given. Using the result in [Gil1] this yields an algorithm for testing interval graphs.

[Ren1] P. L. Renz, "Intersection Representation Of Graphs By Arcs," **Pacific Journal Of Mathematics**, 34:2 (1970) 501-510.
An early paper concerned with the intersection graphs obtained by looking at paths in trees.

[Riv1] R. L. Rivest and J. Vuillemin, "A Generalization And Proof Of The Aanderaa-Rosenberg Conjecture," **Proceedings Of The Seventh Annual ACM Symposium On Theory Of Computing**, (1975) 6-11.

[Rob1] F. S. Roberts, **Representations Of Indifference Relations**, Ph.D. thesis, Stanford University (1968).

A matrix characterization of proper interval graphs is given. Interval graphs are related to the indifference graphs of mathematical psychology.

[Rob2] F. S. Roberts, "Indifference Graphs," in F. Harary (ed.) **Proof Techniques In Graph Theory**, Academic Press, New York (1969) 139–146.

[Rob3] F. S. Roberts, **Discrete Mathematical Models, With Applications To Social, Biological And Environmental Problems**, Prentice-Hall, Englewood Cliffs, New Jersey, to appear.
A compendium of applications for structures such as interval graphs.

[Ros1] D. J. Rose, "Triangulated Graphs And The Elimination Process," **Journal Of Mathematical Analysis And Applications**, 32 (1970) 597–609.
Chordal graphs are shown to be exactly the perfect elimination graphs. These are examined in connection with Gaussian elimination algorithms.

[Ros2] D. J. Rose and R. E. Tarjan, "Algorithmic Aspects Of Vertex Elimination," **Proceedings Of Seventh Annual ACM Symposium On Theory Of Computing**, (1975) 245–254.
The lexicographic breadth first search technique is used to test chordality. This is identical to the method in [Lue1]. Further results are given for the directed case of perfect elimination schemes.

[Ros3] D. J. Rose, R. E. Tarjan, and G. S. Lueker, "algorithmic Aspects of Vertex Elimination on Graphs," to appear in **SIAM Journal on Computing**.

[Rsn1] A. L. Rosenberg, "On The Time Required To Recognize Properties Of Graphs: A Problem," SIGACT News, 5 (1973).

[Rys1] H. J. Ryser, "Combinatorial Configurations," **SIAM Journal Of Applied Mathematices**, 17:3 (May 1969) 593-602.
The consecutive ones property is generalized to the circular ones property and a number of other variants. Some of the matrix properties investigated by Fulkerson and Gross are also extended to these cases.

[Sto1] K. E. Stoffers, "Scheduling Of Traffice Lights — A New Approach," **Transportation Research**, 2 (1968) 199-234.
An application for interval graphs is given.

[Str1] V. Strassen, "Gaussian Elimination Is Not Optimal," **Numerische Mathematik**, 13 (1969) 354-356.
The now-famous result that $O(n^3)$ multiplications are not necessary to perform matrix multiplication.

[Tar1] R. E.. Tarjan, "Efficiency Of A Good But Not Linear Set Union Algorithm," **Journal Of The ACM**, 22:2 (April 1975) 215-225.
An analysis of Tritter's algorithm for disjoint set union introduces a lower bound involving $\alpha(m,n)$, a very slowly growing, but not constant, function.

[Tar2] R. E. Tarjan, "Graph Theory And Gaussian Elimination," in J. R. Bunch and D. J. Rose (eds.) **Sparse Matrix Computations**, Academic Press, New York, to appear.

[Tar3] R. E. Tarjan, private communication.

[Tuc1] A. C. Tucker, "Two Characterizations Of Proper Circular-Arc Graphs," Technical Report No. 69-6, Operations Research House, Stanford University (August 1969) 1-114.
Forbidden subgraphs are given for proper circular-arc graphs. Much of the material in the second two papers is a refinement of this work.

[Tuc2] A. C. Tucker, "Matrix Characterizations Of Circular-Arc Graphs," **Pacific Journal Of Mathematics**, 39:2 (1971) 535-545.
Two variants of the circular ones property are defined. The quasi-circular ones property, which is a generalization, and the circularly compatible ones which is a specialization. Tucker observes that the circular ones property can be tested using any algorithm for consecutive ones, as can the circularly compatible ones, but that no good test is known for quasi-circular ones. Circular-arc and proper circular-arc graphs are shown to be characterized by augmented adjacency matrices having these latter two properties.

[Tuc3] A. C. Tucker, "A Structure Theorem for The Consecutive 1's Property," **Journal Of Combinatorial Theory**, 12B (1972) 153-162.
The earlier work of Lekkerkerker and Boland is repeated for the case of consecutive ones. The same theorem results, except that the asteroidal triple characterization is applied to the bipartitie graph obtained from the rows and columns of the matrix. Five families of forbidden subgraphs, very similar to those of Lekkerkerker and Boland, are found.

Index

Most of the technical terms used in this thesis are indexed here. Authors have also been included where appropriate. This index was prepared using TRIX/RED.