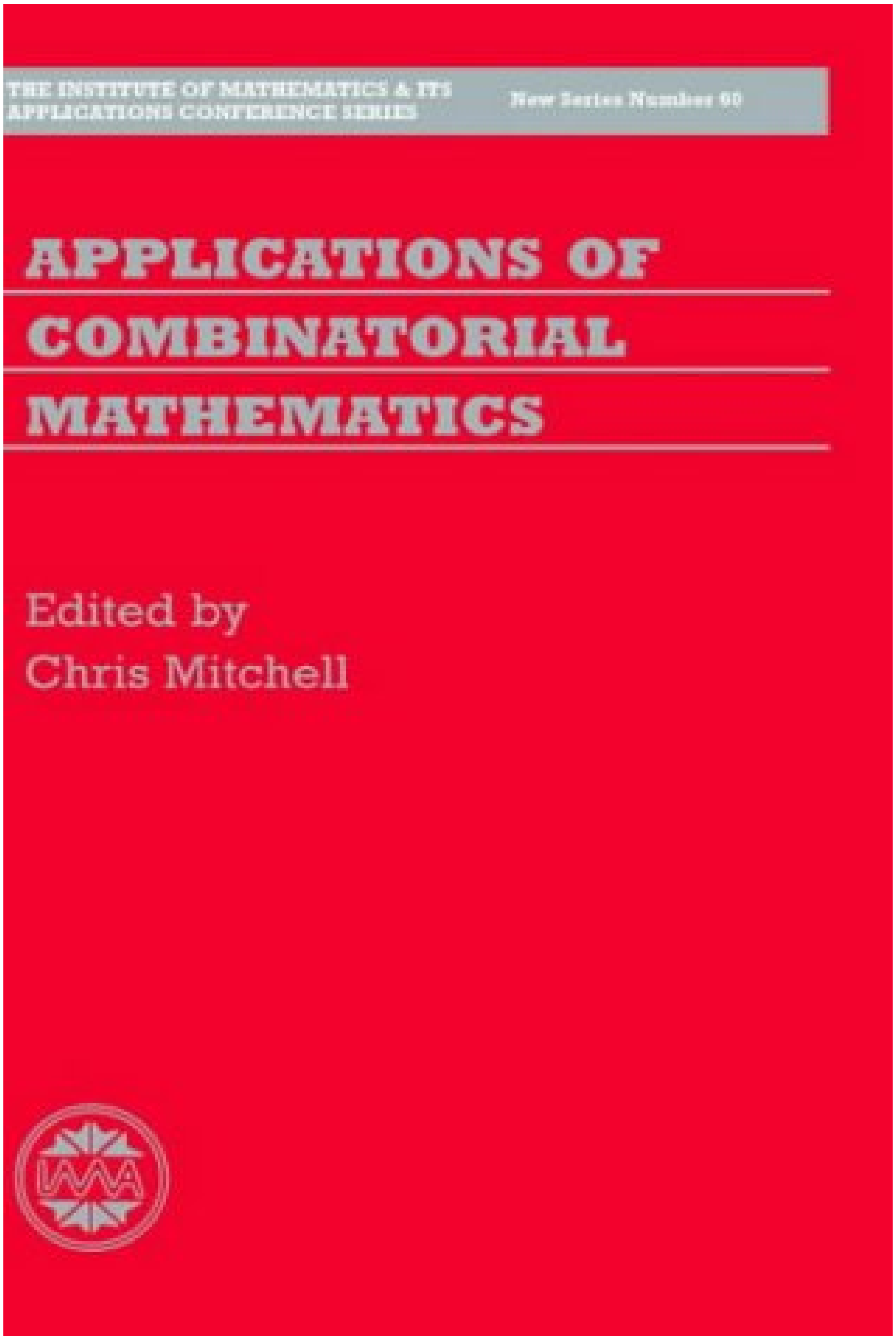


THE INSTITUTE OF MATHEMATICS & ITS APPLICATIONS CONFERENCE SERIES

New Series Number 60

APPLICATIONS OF COMBINATORIAL MATHEMATICS

Edited by
Chris Mitchell

Applications of Combinatorial Mathematics

Based on the proceedings of a conference on the Applications of Combinatorial Mathematics, organized by the Institute of Mathematics and its Applications and held at the University of Oxford in December 1994.

Edited by

CHRIS MITCHELL

Department of Computer Science
Royal Holloway College, University of London

CLARENDON PRESS · OXFORD ·

Continued overleaf

This book has been printed digitally and produced in a standard specification in order to ensure its continuing availability

OXFORD
UNIVERSITY PRESS

Great Clarendon Street, Oxford OX2 6DP

Oxford University Press is a department of the University of Oxford.
It furthers the University's objective of excellence in research, scholarship, and education by publishing worldwide in

Oxford New York

Auckland Bangkok Buenos Aires Cape Town Chennai
Dar es Salaam Delhi Hong Kong Istanbul Karachi Kolkata
Kuala Lumpur Madrid Melbourne Mexico City Mumbai Nairobi
São Paulo Shanghai Taipei Tokyo Toronto

Oxford is a registered trade mark of Oxford University Press in the UK and in certain other countries

Published in the United States by Oxford University Press Inc., New York

Reprinted 2003

ISBN 0-19-851192-2

PREFACE

The IMA Conference on the "Applications of Combinatorial Mathematics" was held at Wadham College, Oxford, between the 14th and the 16th of December 1994. A total of 24 papers were presented, and 16 of these papers are included in this volume.

In the last thirty years, combinatorial mathematics has found itself at the heart of many technological applications. This conference was intended to give an opportunity for papers to be presented on a wide range of different applications of combinatorics. There were two main aims in promoting a conference of this type: to stimulate combinatorial mathematicians to pursue new lines of research of potential practical importance, and to inform all of those who attended of the breadth of the application domain.

The consensus of those attending was that the conference met these objectives very well. Although, or perhaps because, the conference was a small one, a lot of fruitful exchanges of ideas took place in a warm and friendly atmosphere. The range of papers was at least as wide as we could have anticipated, covering topics as diverse as: neural networks, cryptography, radio frequency assignment for mobile telecommunications, coding theory, sequences for communications applications, interconnection networks, data types, knot theory, radar, parallel processing, network reliability, formal specification of programs and protocols, and combinatorial optimisation.

As editor of these proceedings I would like to thank three groups of people without whom the conference could not have been the success that it was. Firstly I should thank the programme committee and the referees, including S. Blackburn, A. Camina, G. Carter, T. Etzion, R. Hill, P. Jeavons, A. Johnstone, R. Leese, C. McDiarmid, P. Mukherjee, K. Paterson, S. Schneider, J. Shawe-Taylor, M. Walker, P. Wild, B. Wilson, L. Wolsey and D. Youngs. Secondly I must thank all the many people at the IMA, for doing much of the hard work in preparing these proceedings, in particular Pamela Bye, for making the conference the success that it was. Thirdly, I would like to thank all the authors for taking the time to prepare and revise their papers for these proceedings.

Chris Mitchell
Royal Holloway, University of London

ACKNOWLEDGEMENTS

The Institute thanks the authors of the papers, the editor Professor Chris Mitchell (Royal Holloway, University of London), Debbie Brown for typesetting the papers, and Simon Byles for proof reading them.

CONTENTS

CONTRIBUTORS

M.D. ATKINSON; Computer Science Division, School of Mathematical and Computational Sciences, University of St Andrews, North Haugh, St Andrews, Fife, Scotland, KY16 9SS.

J.R. BAINBRIDGE; Centre for Systems and Software Engineering, South Bank University, 103 Borough Road, London, SE10 0AA.

S.R. BLACKBURN; Department of Mathematics, Royal Holloway, University of London, Egham, Surrey, TW20 0EX.

S.P. BORGATTI; Department of Sociology, University of South Carolina, Columbia SC29208, USA.

D. COHEN; Department of Computer Science, Royal Holloway, University of London, Egham, Surrey, TW20 0EX.

M. COOPER; IRIT, University of Toulouse III, France.

M.G. EVERETT; School of Computing and Mathematical Sciences, University of Greenwich, Wellington Street, London, SE18 6PF.

G.J. GIBSON; Biomathematics and Statistics Scotland, University of Edinburgh, Mayfield Road, Edinburgh, Scotland, EH9 3JZ.

S.W. GOLOMB; Communication Sciences Institute, University of Southern California, Los Angeles, California 90089-2565, USA.

C.Z.W. HASSELL SWEATMAN; Department of Electrical Engineering, University of Edinburgh, Mayfield Road, Edinburgh, Scotland, EH9 3EA.

S.G. HOGGAR; Department of Mathematics, University of Glasgow, University Gardens, Glasgow, Scotland, G12 8QW.

A.E.D. HOUSTON; Department of Mathematics, Royal Holloway, University of London, Egham, Surrey, TW20 0EX.

D.C. INCE; Department of Computer Science, Open University, Walton Hall, Milton Keynes, MK7 6AD.

P. JEAVONS; Department of Computer Science, Royal Holloway, University of London, Egham, Surrey, TW20 0EX.

R.A. LEESE; Mathematical Institute, University of Oxford, 24–29 St. Giles', Oxford, OX1 3LB, and Smith Institute, Surrey Research Park, Guildford, Surrey, GU2 5YP.

B. MULGREW; Department of Electrical Engineering, University of Edinburgh, Mayfield Road, Edinburgh, Scotland, EH9 3EA.

K.G. PATERSON; Department of Mathematics, Royal Holloway, University of London, Egham, Surrey, TW20 0EX.

A.D. PENGELLY; Department of Computer Science, Open University, Walton Hall, Milton Keynes, MK7 6AD.

K. PICKAVANCE; Department of Mathematics, University of Glasgow, University Gardens, Glasgow, Scotland, G12 8QW.

F. PIPER; Department of Mathematics, Royal Holloway, University of London, Egham, Surrey, TW20 0EX.

S.J. SHEPHERD; Department of Electrical Engineering, University of Bradford, Bradford, BD7 1DP.

D.H. SMITH; Division of Mathematics and Computing, University of Glamorgan, Pontypridd, Mid Glamorgan, Wales, CF37 1DL.

V.A. STRUSEVICH; School of Computing and Mathematical Sciences, University of Greenwich, Woolwich Campus, Wellington Street, London, SE18 6PF.

D. TULLEY; Computer Science Division, School of Mathematical and Computational Sciences, University of St Andrews, North Haugh, St Andrews, Fife, Scotland, KY16 9SS.

R.W. WHITTY; School of Computing, Information Systems and Mathematics, South Bank University, 103 Borough Road, London, SE10 0AA.

D.A. YOUNGS; Communications Security and Advanced Development Group, Vodafone Limited, 2–4 London Road, Newbury, Berkshire, RG13 1JL.

The Combinatorics of some Abstract Data Types

M.D. Atkinson and D. Tulley

School of Mathematical and Computational Sciences, University of St. Andrews, Scotland

Abstract

Abstract data types (ADTs) may be regarded as abstract machines and then a program for an ADT is any sequence of operations allowed by its specification. The effect of such programs on container ADTs is captured by the relationship between each input sequence and the set of possible output sequences that can result from it. This relationship is studied principally in the case of dictionaries, stacks and priority queues and a distinction is drawn between ADTs of unbounded and bounded capacity.

1 Introduction

Abstract machines have a long and honourable history in Computer Science. Turing machines, push-down automata, and finite-state machines are three well known types; they have been used to study general purpose computers, compilers, and string manipulation although their importance goes well beyond these three applications. The central theoretical issue for these (and other) machines is the characterisation of the languages associated with them. The aim of this paper is to study some abstract data types in the same spirit.

Abstract data types (ADTs) have a relatively short and honourable history. They are central to the point of view adopted in object-oriented programming (which is setting the direction of programming in the 1990s) and which now permeates all large software projects. Although abstract data typing was initially adopted primarily for its use as a software design tool it has always been recognised that each data type had a rigorous mathematical definition.

Each ADT is characterised by the set of operations that can be performed on it. Therefore an ADT can be regarded as an abstract machine whose instruction set is the set of operations it supports. Some of these operations may supply input or output while others may examine or change the state of the ADT. The precise specification of these instruction sets has been studied in considerable depth by algebraic means (see [9,10] for a survey). However, the classical abstract machines are studied at a much deeper level; their behaviour in response to *arbitrary* sequences of instructions (i.e. programs) is studied and this behaviour

Table 1. Some ADTs and their delete operations

Name of ADT	Delete Operation
Queue	Delete the item that has been in the queue the longest
Stack	Delete the item that was placed in the stack most recently
Dictionary	Delete any item
Priority queue	Delete the smallest item

is captured by the idea of the language recognised by the machine. As yet, such a study has hardly begun for ADTs although, as indicated below, there is a very natural extension of the language notion to ADTs.

There is an infinite number of data types and it seems to be infeasible to give a general theory of their associated languages which has deep implications for all of them. However, in practice, only a small number of ADTs recur frequently in software and algorithm design (stacks, queues, arrays, dictionaries etc) and it is perhaps more profitable to study only those which have demonstrable software utility.

This paper will concentrate on *container* ADTs: those for which Insert and Delete operations are defined. Such ADTs act as data transformers, outputting their input data in a permuted order. If, except for house-keeping operations, Insert and Delete are the only operations supported by the data type then the functional behaviour of the ADT is essentially defined by the possible ways in which it can permute the data. A sequence of Insert and Delete operations constitutes a program for the ADT when it is regarded as a machine. For such an ADT it is not sensible to define its associated language to be the set of input sequences that lead to an accepting state, since that discards so much essential information about the output. Instead, we propose that the associated language should be defined to be the set of (input, output) pairs of sequences that can arise from the execution of an ADT program (indeed, even for Turing machines, this definition is attractive since it avoids fudges about how the input is to be encoded). As we shall see, there are a number of questions about the language associated with an ADT whose formulation and solution require combinatorial machinery. We shall list some of these questions and then go on to discuss their solutions for some particular data types.

The different container data types are generally distinguished from each other by the type of Delete operation that they support. Table 1 shows four common container data types and the properties of their Delete operation.

Let A be any container data type. We shall consider only programs for A which begin and end with A in the empty state; this represents the normal way that a container data type would be used. Such a program then consists of a sequence of Insert and Delete operations in which every initial segment contains at least as many Inserts as Deletes (this condition is to ensure that a Delete operation is never executed when A is empty) and having equal numbers of Inserts and Deletes (to ensure that the final state of A is empty). Let σ be any sequence of length n and let P be any program with n Inserts and n Deletes. The execution of P with σ as the input sequence results in the members of σ being inserted into A in order of occurrence in σ; the Delete operations generate a sequence τ which we call the output of P. A pair (σ, τ) which is related by a program P in this way is called *allowable* (or A-allowable when the ADT cannot be deduced from context). The set of allowable pairs is denoted by $L(A)$ and is called the language associated with the data type A. Basic combinatorial questions about $L(A)$ include:

1. How many A-allowable pairs with each component of length n are there?
2. Is there a characterisation of the A-allowable pairs that enables them to be recognised quickly?
3. Is there an efficient algorithm that, given an input sequence σ, can determine how many A-allowable pairs (σ, τ) there are? And, dually,
4. Is there an efficient algorithm that, given an output sequence τ, can determine how many A-allowable pairs (σ, τ) there are?

In practice, container ADTs generally have a bounded size, either enforced by their implementation or the physical limits of the hardware, so it makes sense to consider the above questions when no more than k elements can be stored at any time in the ADT. We therefore introduce the idea of *k-allowability* by defining the language $L_k(A)$ of a *k-bounded* ADT A to be the set of allowable pairs (σ, τ) for which there is a program P which can transform σ into τ without requiring more than k elements to be stored at any one time.

There are other variations we can introduce as well. We have not yet stipulated what form the input sequence takes; it could be taken as a sequence of distinct elements (which we can assume to be the elements $1, 2, \ldots n$ in some order, without loss of generality), as a word over the binary alphabet, or as a reordering of an arbitrary multiset $S = \{1^{a_1}, 2^{a_2}, \ldots, r^{a_r}\}$.

This leads to a large number of questions to be studied and we shall present at least partial solutions to many of them in this paper. In Section 2 we present results about dictionaries concentrating mainly on the case when the input sequence is a word over the binary alphabet. Then, in Section 3 we consider stacks and show that, in the binary case, they behave like dictionaries. Section 4 contains results for queues and deques (*double ended queues*) and finally in Section 5 we study priority queues and double ended priority queues.

Most of the results are related to questions 1, 3 and 4 in the above list but there has been some progress on question 2. This is mostly, but not exclusively, based on the idea of *avoided patterns* as used by Pratt ([15]) and Knuth

([12, 2.2.1 Q5]). A pattern of length m is a permutation $\rho = (\rho_1, \rho_2, \dots \rho_m)$ of $1, 2, \dots, m$, and a sequence $\sigma = (\sigma_1, \sigma_2, \dots \sigma_n)$ is said to *contain* the pattern if there is a subsequence σ' of σ such that $|\sigma'| = m$ and $\rho_i < \rho_j$ if and only if $\sigma'_i < \sigma'_j$. As an example, the sequence $5, 4, 2, 3, 1$ contains the pattern $3, 1, 2$ because it contains the subsequence $5, 2, 3$, but on the other hand $5, 4, 3, 2, 1$ does not contain the pattern $1, 2, 3$. If ρ does not occur within σ we shall say that σ *avoids* ρ.

2 Dictionaries

Dictionaries are the class of abstract data types with the most general delete operation. They allow the removal of any element which is currently stored in the dictionary. Their behaviour was studied in [2] where they are referred to as *buffers* and, in the case of a bounded capacity, as *bounded buffers.* When the input sequence is a permutation of distinct elements, the order of the elements has no effect on the number of outputs possible and so instead of studying the number of allowable pairs it is only necessary to consider how many output sequences are possible from the input sequence $1, 2, \dots, n$. Then the number of allowable pairs is $n!$ times the number of allowable output sequences from $1, 2, \dots, n$. In the unbounded case there is not any great challenge since it is clear that the input sequence can be permuted into any output sequence by merely inserting the entire input sequence and then deleting the elements in the required order. Thus $L(\mathit{Dictionary}) = \{(\sigma, \tau) | \tau \text{ is a rearrangement of } \sigma\}$ and for this reason the unbounded dictionary is the most permutationally powerful abstract data type. The bounded case is also relatively simple for we have in [2]

Lemma 1. *For a bounded dictionary of capacity k, each input sequence of n distinct elements gives $k^{n-k}k!$ allowable output sequences.*

Lemma 2. *The allowable output sequences of a bounded dictionary with input sequence $1, 2, \dots, n$ are precisely the sequences that avoid all patterns of length $k+1$ which begin with their maximal element.*

In the case when the input sequence is a binary sequence the results are a little more complex.

Lemma 3. *For a bounded dictionary of size k the number x_n of binary allowable pairs of length n satisfies the recurrence*

$$\begin{aligned} x_{k+n} &= \sum_{i=1}^{r} (-1)^{i+1} x_{k+n-i} a_{i,k} \quad \textit{with} \quad r = \lceil \tfrac{k}{2} \rceil \\ a_{0,k} &= 1 \\ a_{i,k} &= 2\binom{k-i}{i-1} + \binom{k-i}{i} \end{aligned} \tag{2.1}$$

Proof. Let $w_n^{(i)}$ be the number of allowable pairs of length n of the form $(0^i\alpha, \beta)$, then obviously $x_n = w_n^{(0)}$, and

$$\begin{aligned} w_{n+1}^{(0)} &= 2w_{n+1}^{(1)} \quad \text{for} \quad n \geq 0 \\ w_{n+1}^{(i)} &= w_n^{(i-1)} + \sum_{j=i}^{k-1} w_n^{(j)} \quad \text{for} \quad n \geq i > 0. \end{aligned}$$

The first of these two equations holds because the number of pairs of the form $(0\alpha, \beta)$ is the same as the number of pairs of the form $(1\alpha, \beta)$ and all allowable pairs have one of these two forms.

For the second notice that all pairs $(0^i\alpha, \beta)$, of length $n+1$, either have the form $(0^i\alpha, 0\beta')$ or $(0^i\alpha, 1\beta')$. There are $w_n^{(i-1)}$ pairs of the first form because the first 0 is input and immediately output, then there are $w_n^{(i-1)}$ ways to complete the pair. The second can be split into several further forms, $(0^j1\gamma, 1\beta')$ for $i \leq j < k$. For each of these the dictionary must insert all j 0s and the 1 and then immediately output the 1. There are then $w_n^{(j)}$ ways to complete the pair and thus there are $\sum_{j=i}^{k-1} w_n^{(j)}$ allowable pairs of the second form.

We can represent this recurrence more succinctly using matrix notation. Let

$$w_n = \begin{pmatrix} w_n^{(1)} \\ \vdots \\ w_n^{(k-1)} \end{pmatrix}, \qquad A_k = \begin{pmatrix} 3 & 1 & \cdots & 1 & 1 \\ 1 & 1 & \cdots & 1 & 1 \\ 0 & 1 & \cdots & 1 & 1 \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & 1 & 1 \end{pmatrix}$$

then the recurrence equation is

$$w_{n+1} = A_k w_n. \tag{2.2}$$

Hence, for any constants $d_0, d_1, \ldots d_t$, $\sum_{i=0}^t d_i w_{n+i} = \sum_{i=0}^t d_i A_k^i w_n$ and if we choose the constants so that $\sum_{i=0}^t d_i \lambda^i$ is the characteristic polynomial of A then we shall have $\sum_{i=0}^t d_i w_{n+i} = 0$. We can derive a recurrence equation for the characteristic polynomial, $u_k(\lambda) = det(A_k - \lambda I)$, of A_k by subtracting the $(k-1)^{th}$ column of the determinant from the k^{th} column and expanding it by the k^{th} column. It is then easily seen that

$$u_k(\lambda) = -\lambda(u_{k-1}(\lambda) + u_{k-2}(\lambda)) \text{ for all } k \geq 3.$$

The initial cases

$$\begin{aligned} u_0(\lambda) &= 1 \\ u_1(\lambda) &= 3 - \lambda \\ u_2(\lambda) &= \lambda^2 - 4\lambda + 2 \end{aligned}$$

are calculated directly.

From the recurrence it follows easily by induction on k that there exist polynomials y_{2r} and y_{2r+1} each of degree $r+1$ for which

$$u_{2r}(\lambda) = \lambda^{r-1} y_{2r}(\lambda) \tag{2.3}$$

$$u_{2r+1}(\lambda) = \lambda^{r} y_{2r+1}(\lambda) \tag{2.4}$$

and that the polynomials satisfy

$$y_{2r}(\lambda) = -\lambda y_{2r-1}(\lambda) - y_{2r-2}(\lambda) \tag{2.5}$$

$$y_{2r+1}(\lambda) = -y_{2r}(\lambda) - y_{2r-1}(\lambda). \tag{2.6}$$

Now a standard inductive proof using binomial coefficient identities proves

$$y_{k-1}(\lambda) = (-1)^{k-1} \sum_{i=0}^{r} \lambda^{r-i} (-1)^i a_{i,k} \tag{2.7}$$

where

$$r = \left\lceil \tfrac{k}{2} \right\rceil, a_{0,k} = 1, a_{i,k} = 2\binom{k-i}{i-1} + \binom{k-i}{i}$$

Combining (2.3) and (2.4) gives

$$\begin{aligned} u_{k-1}(\lambda) &= \lambda^{\lfloor \frac{k-2}{2} \rfloor} y_{k-1}(\lambda) \\ &= (-1)^{k-1} \sum_{i=0}^{r} \lambda^{r-i+\lfloor \frac{k-2}{2} \rfloor} (-1)^i a_{i,k}. \end{aligned}$$

Therefore the sequence (w_n) satisfies

$$\sum_{i=0}^{r} w_{r+\lfloor \frac{k-2}{2} \rfloor + n - i} (-1)^{i+1} a_{i,k} = 0 \quad \text{where} \quad r = \left\lceil \frac{k}{2} \right\rceil$$

Since $r + \lfloor \frac{k-2}{2} \rfloor \leq k$, $w_n^{(1)}$ is an element of the vector w_n and $x_n = w_n^{(0)} = 2w_n^{(1)}$, we have

$$x_{k+n} = \sum_{i=1}^{r} x_{k+n-i} (-1)^{i+1} a_{i,k}.$$

The recurrence of this lemma shows how x_n can be computed once initial values $x_0, x_1, \ldots x_{k-1}$ are known. However, for $t \leq k-1$ (indeed $t \leq k$) the number of binary allowable pairs of length t is unconstrained by the dictionary size k. Therefore, if $t \leq k$, it is easily seen that

$$x_t = \sum_{i=0}^{t} \binom{t}{i}^2 = \binom{2t}{t}.$$

For example, with $k = 3$, the recurrence becomes

$$x_{n+3} = 4x_{n+2} - 2x_{n+1}$$

and the values of x_n are

n	0	1	2	3	4	5	6
x_n	1	2	6	20	68	112	312

3 Stacks

Unbounded stacks were studied extensively in the 1970s and a number of significant connections with other combinatorial objects were found. Because the permutational power of the stack comes from its structure and not from the relative values of the elements it is processing, the allowable pairs are closely related to the valid programs of a stack. This leads to several correspondences; for example, the number of valid programs of length $2n$ which a stack can execute, and thus, given a fixed input sequence, the number of allowable output sequences of length n, is in a one to one correspondence with the number of balanced bracket sequences of length $2n$. There are then similar correspondences with trees ([12, 2.3.4.4],[16]), triangulations of polygons ([11, pp. 320–324]), Young Tableaux ([13, pp. 63–64]), lattice paths ([14]) and ballot sequences ([13, p. 531],[17]). These connections are of great interest in the design of efficient algorithms ([13,15,18]) and all point to the stack's fundamental role in giving precise expression to informal concepts such as "nesting", "structured decomposition" and "hierarchy".

It is known from the many correspondences above that, given a fixed input sequence of distinct elements, there are $c_n = \binom{2n}{n}/(n+1)$ (the n^{th} Catalan number) possible output sequences. Therefore the number of allowable pairs of length n is $n!c_n$. It is also known that, if the input sequence is $1, 2, \ldots n$, the allowable output sequences of an unbounded stack are those sequences which avoid the pattern $3, 1, 2$ ([12]).

In the bounded case we can apply the techniques of [2] which show that the number of output sequences for a fixed input sequence of distinct elements rises exponentially in the length of the input sequence. The base α_k of the exponent depends on the stack size k and some values of α_k are given in Table 2. Because

Table 2. α_k for small values of k

Stack Size	α_k	Numerical Estimate
1	1	1
2	2	2
3	$\frac{3+\sqrt(5)}{2}$	2.61
4	3	3
5	largest root of $x^3 - 5x^2 + 6x - 1$	3.247

of the asymptotic behaviour of c_n, $\alpha_k \to 4$ as $k \to \infty$. This case was also studied in [8], where it was posed as the problem of finding the average height of planted plane trees. This corresponds to the average capacity require to generate a randomly chosen output sequence. The result is that on average

$$k = \sqrt{\pi n} - \frac{1}{2} + O\left(\frac{1}{\sqrt{n}} \log n\right)$$

stack locations are required.

When the input sequence is a binary sequence the stack has exactly the same behaviour as a dictionary for both the bounded and unbounded cases, as the following shows.

Lemma 4. *The allowable output sequences of a stack on a binary input are precisely those of a dictionary of the same capacity on a binary input.*

Proof. Suppose $(\sigma, \tau) \in L_k(\mathit{Dictionary})$, then there is a program of insert and delete operations which transform σ into τ. Among all such programs we choose one, C say, in which all the insert operations are delayed as long as possible; in C an insert operation only occurs if it is not possible to produce any more of the output sequences τ from the elements stored in the dictionary. So, when C is executed, the only point at which both a 0 and a 1 are stored in the dictionary is when there are one or more 0s stored already, a 1 was inserted by the last operation in C which has been executed and the 1 will be deleted by the next operation in C (and the same situation with 1 and 0 interchanged).

A stack can execute the same sequence, C, and will produce the same output sequence τ from the input sequence σ because, for every delete operation, the element which the dictionary would output is either the last one inserted (in which case the stack can also output it), or there are only elements of a single value stored in the dictionary/stack. Therefore $(\sigma, \tau) \in L_K(\mathit{Stack})$ and so $L_k(\mathit{Dictionary}) \subseteq L_K(\mathit{Stack})$.

The other inclusion is trivial and so we have $L_k(\mathit{Dictionary}) = L_K(\mathit{Stack})$.

It follows from this that the number of allowable pairs of a stack with binary input sequences satisfies the recurrence given for the bounded dictionary in Section 2.

When the input sequence is allowed to have duplicated elements, little is know about the behaviour, but some progress has been made in [2] in the case that equal symbols occur adjacently.

4 Queues and deques

The analysis of queues is entirely trivial. In both the bounded and unbounded cases, the only possible output sequence is the input sequence, so there are $n!$ allowable pairs of length n. It is interesting to note that with such a permutationally weak data type even the transition from bounded to unbounded has no effect on its power. In all the other data types considered here it is a significant transition.

Double ended queues (*deques*) are queues which allow two insert and two delete operations. It is possible to insert elements at both ends of the queue and it is possible to delete elements from both ends. There are also variants of the deque; an *input restricted deque* has only one input operation (it can only insert elements at one end of the queue) and an *output restricted deque* has only one delete operation (it can only remove elements from one end of the queue). The third possible variant, where we have only one input operation and one output operation, can have two forms. If we allow insertion and deletion at the same end of the queue we have a stack, otherwise we have an ordinary queue. Both of these possibilities have been considered previously so we shall ignore them here. Deques and their two variants were studied in [15], where most of the results are presented as excluded pattern conditions.

The number of cases to study is reduced a little because the properties of input restricted deques and output restricted deques are symmetric. This is because, a pair (σ, τ) is allowable (or k-allowable) for an input restricted deque if and only if the pair (τ^R, σ^R) is allowable (or k-allowable) for an output restricted deque (σ^R denotes the reversal of the sequence σ). Knuth [13, p. 534, Q 13] states that the generating function for the number of allowable output sequences for an output restricted deque, with some input sequence σ, is

$$G(z) = \frac{1}{2}(1 + z - \sqrt{1 - 6z + z^2}).$$

Pratt then shows that there is a 2:1 correspondence between these outputs and honest trees with n leaves (an honest tree is a general tree with no nodes of out degree 1). He then goes on to prove that, on input $1, 2, \ldots n$, the allowable outputs are precisely those which avoid the patterns $4, 2, 3, 1$ and $4, 1, 3, 2$. The corresponding patterns for an input restricted deque are $4, 2, 3, 1$ and $4, 2, 1, 3$. He then shows that the allowable outputs for a deque with input $1, 2, \ldots n$ are those sequences which avoid the infinite set of patterns shown in Table 3.

Table 3. The excluded patterns which characterise a deque

5, 2, 3, 4, 1
5, 2, 7, 4, 1, 6, 3
5, 2, 7, 4, 9, 6, 3, 8, 1
5, 2, 7, 4, 9, 6, 11, 8, 1, 10, 3
5, 2, 7, 4, 9, 6, 11, 8, 13, 10, 3, 12, 1
etc.
and those obtained by exchanging 1 and 2 and/or the last two elements in each pattern

The bounded versions of the above deque variants can be handled by the techniques of [2]. For example, for a deque of size 3 we can show there are 2.3^{n-2} allowable outputs for any fixed input. Similarly for a deque of size 4 we can show that the number of allowable outputs for a fixed input sequence is 6.4^{n-3}. For size 5 the number of allowable outputs grows exponentially with base α, where α is the largest root of $x^3 - 7x^2 + 10x + 2$ (approximately 4.855).

5 Priority queues

There has been a great deal of work in recent years on the combinatorial properties of priority queues, both bounded and unbounded, operating on input sequences formed from distinct elements, the binary alphabet and multisets.

It is convenient to define

$$\begin{aligned} s(\tau) &= |\{\sigma|(\sigma,\tau) \in L(\text{Priority Queue})\}| \\ t(\sigma) &= |\{\tau|(\sigma,\tau) \in L(\text{Priority Queue})\}| \\ s_k(\tau) &= |\{\sigma|(\sigma,\tau) \in L_k(\text{Priority Queue})\}| \\ t_k(\sigma) &= |\{\tau|(\sigma,\tau) \in L_k(\text{Priority Queue})\}|. \end{aligned}$$

In [5] it is shown that, when the input is formed from n distinct elements, there are $(n+1)^{n-1}$ allowable pairs of length n for an unbounded priority queue. For this case algorithms are presented in [1] which calculate $s(\tau)$ in $O(n)$ time and $t(\sigma)$ in $O(n^4)$ time. The transitive closure of the allowability relation is also found.

When the inputs are restricted to binary sequences it was shown in [6] that there are c_{n+1} allowable pairs of length n. An $O(n^2)$ algorithm is then presented which calculates $s(\tau)$. It is also shown that (σ,τ) is allowable if and only if (τ^R, σ^R) is allowable and this gives an $O(n^2)$ algorithm to compute $t(\sigma)$.

Some very recent work in [3] gives the only result we know of in the case when the input is a rearrangement of a multiset $S = \{1^{a_1}, 2^{a_2}, \ldots, r^{a_r}\}$. Here it is shown that there are

$$\frac{1}{n+1}\prod\binom{n+1}{a_i}$$

allowable pairs.

All the results above are for unbounded priority queues. The study of the bounded case was begun in [4]. For input sequences of distinct elements the only progress made has been for the priority queue of size 2. In this case, if x_n is the number of allowable pairs then

$$\sum x_n \frac{t^n}{n!} = \frac{1}{1+\log(1-t)}$$

and from this it can be deduced that $x_n/n!$ is asymptotic to $(e/(e-1))^n$ as $n \to \infty$. It was also shown how to compute $s_2(\tau)$ and $t_2(\sigma)$ in time $O(n^2)$.

When the input is a binary sequence more general results are known. For example, $s_k(\tau)$ and $t_k(\sigma)$ can be computed in time $O(n^2)$. It was also shown in [4] that there is a $1-1$ correspondence between k-allowable pairs of length n and ordered forests of height no more than $k+2$ on $n+2$ nodes.

The study of double ended priority queues has been rather less productive. This data type has two kinds of delete operation: Delete-Minimum and Delete-Maximum (denoted by "d" and "D" respectively). Many experimental results and conjectures are reported in [19]. Linton has given a characterisation of allowable pairs in terms of avoided pairs of patterns extending the idea of pattern avoidance in Section 1. The only other results we know of bear on questions 3 and 4 of the introduction.

Let $\mathcal{D}$ be any sequence of n Delete-Minimum and Delete-Maximum operations and let $\pi(\mathcal{D})$ be the permutation of $1, 2, \ldots, n$ defined by

$$\pi(\mathcal{D})_i = \begin{cases} n-j & \text{if} \quad \mathcal{D}_i = D \\ i-1-j & \text{if} \quad \mathcal{D}_i = d \end{cases}$$

where j is the number of Ds among $\mathcal{D}_1 \ldots \mathcal{D}_{i-1}$.

Also define the complementary permutation $\bar{\pi}(\mathcal{D})$ by $\bar{\pi}(\mathcal{D})_i = n+1-\pi(\mathcal{D})_i$. Then we have

Lemma 5. *Consider the set of double ended priority queue programs in which the sequence of delete operations is a fixed sequence $\mathcal{D}$, and let $A(\mathcal{D})$ be the set of all (σ, τ) related by such programs. Further, let $s_{\mathcal{D}}(\tau) = |\{\sigma | (\sigma, \tau) \in A(\mathcal{D})\}|$ and $t_{\mathcal{D}}(\sigma) = |\{\tau | (\sigma, \tau) \in A(\mathcal{D})\}|$. Then*

1. $s_{\mathcal{D}}(\tau)$ *is maximal when* $\tau = \pi(\mathcal{D})$.

2. $s_{\mathcal{D}}(\tau)$ *is minimal when* $\tau = \bar{\pi}(\mathcal{D})$.

3. $t_{\mathcal{D}}(\sigma)$ *is minimal when* $\sigma = \pi(\mathcal{D})$.

6 Conclusion

We have presented a unified framework in which the (input,output) relation for various container data types can be studied. The properties of the relation for stacks, queues, dictionaries and priority queues are fairly well understood.

The main unsolved problems requiring further research include

1. A treatment of deques and double ended priority queues as complete as that for stacks, queues, dictionaries and unbounded priority queues.
2. A better understanding of bounded priority queues on non binary inputs.
3. An extension of the theory to networks of container data types (as proposed by Tarjan for stacks and queues in [18]). Note that networks of bounded dictionaries, queues, deques and stacks can, in principle, be handled by the techniques of [2].

References

1. Atkinson, M.D. and Beals, R. (1994). Priority queues and permutations. *SIAM J. Comp.*, **23**, 1225–1230.
2. Atkinson, M.D., Livesey, M.J. and Tulley, D. Networks of bounded buffers. Submitted to *Theoretical Computer Science.*
3. Atkinson, M.D., Linton, S.A. and Walker, L.A. Priority queues and multisets. Submitted to *Electronic J. Combinatorics.*
4. Atkinson, M.D. and Tulley, D. Bounded capacity priority queues. Submitted to *Theoretical Computer Science.*
5. Atkinson, M.D. and Thiyagarajah, M. (1993). The permutational power of a priority queue. *BIT*, **33**, 2–6.
6. Atkinson, M.D. (1993). Transforming binary sequences with priority queues. *Order*, **10**, 31–36.
7. Atkinson, M.D. and Walker, L.A. (1993). Enumerating k-way trees. *Information Processing Letters*, **48**, 73–75.
8. de Bruijn, N.G., Knuth, D.E. and Rice, S.O. (1972). The average height of planted plane trees. *Graph Theory and Computing*, Editor: R.C. Read, Academic Press, 15–22.

9. Classen, I., Ehrig, H., Mahr, B. and Orejas, F. (1992). Introduction to algebraic specification. *Computer J., Part 1: Formal Methods for Software Development*, **35**, 451–459.

10. Classen, I., Ehrig, H., Mahr, B. and Orejas, F. (1992). Introduction to algebraic specification. *Computer J., Part II: From Classical View to Foundations of Systems Specifications*, **35**, 460–467.

11. Cormen, T.H., Leiserson, C.E. and Rivest, R.L. (1992). *Introduction to Algorithms*, McGraw-Hill, Cambridge, Mass.

12. Knuth, D.E. (1973). *Fundamental Algorithms - The Art of Computer Programming*, Addison-Wesley, Reading, Mass.

13. Knuth, D.E. (1973). *Sorting and Searching - The Art of Computer Programming*, Addison-Wesley, Reading, Mass.

14. Mohanty, S.G. (1979). *Lattice Path Counting and Applications*, Academic Press, New York.

15. Pratt, V.R. (1973). Computing permutations with double-ended queues, parallel stacks and parallel queues. *5th ACM Sym. "Theory of Computing"*, 268–277.

16. Roton, D. (1975). On a correspondence between binary trees and a certain type of permutation. *IPL*, **4**, 58–61.

17. Roton, D. and Varol, Y.L. (1978). Generation of binary trees from ballot sequences. *JACM*, **25**, 396–404.

18. Tarjan, R.E. (1972). Sorting using networks of queues and stacks. *JACM*, **19**, 341–346.

19. Thiyagarajah, M. (1993). Permutational power of priority queues. *Master's Thesis*, Carleton University.

Recent Results in the Theory of Program Flowgraphs

J.R. Bainbridge and R.W. Whitty

South Bank University, London

Abstract

Attributed grammars, which play a major role in the theory of compiler design, provide a neat way of defining flowgraph functions (that is, digraph functions where the digraphs are modelling programs). We use this idea to review some recent research in which flowgraph functions are used as complexity metrics for programs. The choice of grammar is seen to influence the expressive power of this approach: some aspects of program complexity, expressed as flowgraph functions, cannot be defined using some grammars. The lesson seems to be that, at least from a mathematical viewpoint, we are not comparing like with like when we study the many complexity metrics which have been proposed.

1 Introduction

Graph theory has a natural role in the study of control flow in computer programs. The different possible routes of execution through the program code correspond to the walks from a source to a termination vertex in a digraph called a "flowgraph". The flowgraph may be used as a basis for testing the program ([21]); for restructuring it to make it more comprehensible ([10]); for optimizing its performance ([9]); or for measuring its complexity, as relating to any of the above tasks, or some other ([6]).

Much of this analysis of flowgraphs revolves around *flowgraph decomposition* which uncovers the structure of a flowgraph in terms of how it has been built up using two digraph operations: *sequencing* and *nesting*. Sequencing concatenates two or more flowgraphs together; nesting inserts one or more flowgraphs into another. Often the analysis of flowgraphs amounts to defining suitable functions on flowgraphs and decomposition offers a possible basis for defining such functions. There are intuitive reasons for doing this: Prather and Guilieri [12] advocate decomposition as a precursor to restrucuring, in order to preserve as far as possible the original features of program structure. This is also the main idea in the work of Cowell et al. [4] and the approach is implicit in earlier work such as that of Urschler [17] and Linger et al. [10]. More recently, program decomposition has been the basis for generating a class of software complexity metrics known as *hierarchical metrics* ([6,7,13,22]). It is this latter idea which we will concentrate on but most of what we say applies equally well to other aspects of flowgraph analysis. By the way, we use the word "metric" in a non-mathematical sense: the term has nothing to do with the triangle inequality but has stuck in programming circles.

There are three possible reasons for being interested in hierarchical metrics, that is, in trying to define complexity metrics using the decomposition structure:

1. **divide and conquer**: the decomposition can often be expected to reduce a flowgraph to "prime" components of only a few vertices each. This may greatly simplify the calculation of the metric function. Later, we shall see that counting the number of trails through a flowgraph is an example of this approach ([3]).
2. **a descriptive framework**: many different complexity metrics can be defined in a uniform way using the decomposition structure. This allows a comparison of like with like, an approach adopted by Zuse [22] and Whitty [20], and generalized by Fuchs and Stainer [8].
3. **axiomatization**: it has been proposed as an axiom ([13]) that the "complexity" of a program should reduce to the complexity of its prime components and the complexity of the sequencing and nesting operations. This idea has recently been taken further by Prather [16] and applied to functional programs by van den Berg and van den Broek [18].

Whatever the reason for studying hierarchical metrics, there have recently been several papers which have examined the particular forms which these flowgraph functions can take. These have been presented in different terminologies and our main purpose is to review them in a uniform way using the language of attributed grammars. This seems to us to be the most natural setting, not least because of the link to standard program analysis techniques based on parsing program syntax (see for example, [1]). In the next section we give some necessary background in flowgraph theory. In Section 3 we explain how to define flowgraph functions using an attributed grammar. In Section 4 we discuss some variations of this basic grammar which have recently been proposed. Section 5 presents what might be considered the state-of-the-art in terms of what flowgraph functions can be represented using grammars: the particular problem being that of defining metrics connected with program testing. Finally, Section 6 tries to draw some conclusions from the preceding discussion.

2 Flowgraph theory

Many variants of the basic flowgraph model exist, all more or less equivalent, and this paper adopts the one introduced in ([5]):

Definition 1. *A* flowgraph $F = (G, s, t)$ *is a triple consisting of a digraph G together with two distinguished vertices s and t of G satisfying the* flowgraph property*: every vertex of G lies on some walk from s to t. Additionally, t is required to have outdegree zero.*

The vertices s and t are referred to as the *start* and *termination* vertex of F respectively. Vertices of outdegree one play a special role and are referred to as *process vertices*. They represent the actions performed or executed by the computer program.

Informally, the sequencing operation may be described as follows: given two flowgraphs, F_1 and F_2, one produces a new flowgraph $(F_1 \; ; \; F_2)$ by simply regarding the termination vertex of F_1 as identical with the start vertex of F_2.

Again, given F_1 and F_2, it may happen that F_1 has a process vertex v. In that case, we may produce a new flowgraph $(F_1 \uparrow_v F_2)$ by replacing the single edge leading from v by the whole flowgraph F_2. Thus, the start vertex of F_2 is identified with v and the termination vertex of F_2 is identified with the successor vertex of v in F_1. This is the process of *nesting* flowgraphs.

If a flowgraph F has several flowgraphs $F_1, \ldots, F_n$ nested into it, then clearly they will be mutually edge-disjoint in the resulting flowgraph. So we may without ambiguity represent the nesting as happening "simultaneously" via the expression $F \uparrow_{v_1,\ldots,v_n} F_1, \ldots F_n$.

Sometimes the actual vertex v nested onto is of no importance. Where there is no danger of confusion we use notation like $F_1 \uparrow F_2$.

An example of sequencing and nesting is shown in Figure 1.

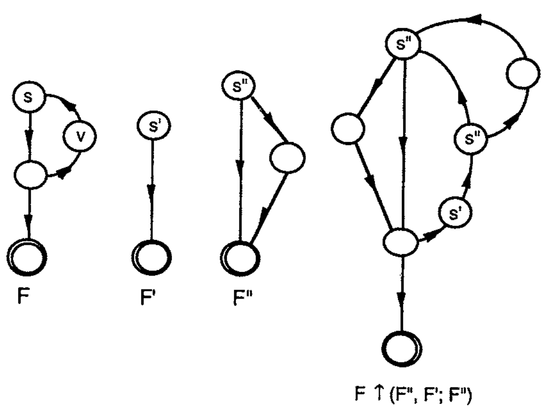


Figure 1. Sequencing and nesting flowgraphs

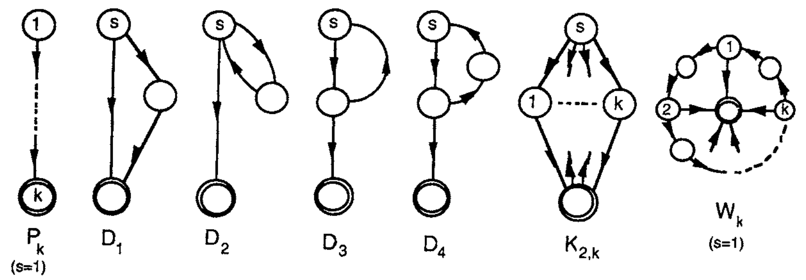


Figure 2. Some prime flowgraphs

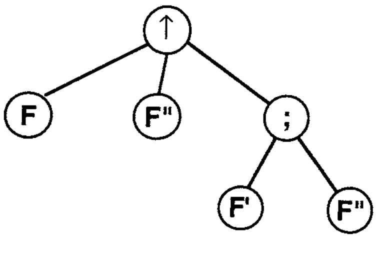


Figure 3. Decomposition tree of $F \uparrow F''$, $F'; F''$

Prime flowgraphs are flowgraphs which cannot be built up non-trivially by sequencing or nesting. The notion of primality has been traced in ([10]) to the thesis of Maddux [11]. Some small primes which occur commonly in practice are shown in Figure 2. Note that, by convention, the path graphs $P_k, k \geq 2$ are counted as primes. Looking again at Figure 1 we can see that the flowgraph resulting from the sequencing and nesting is, in fact, $D_4 \uparrow (D_1, P_2; D_1)$.

Associated with any flowgraph is a *decomposition tree* which describes how the flowgraph is built by sequencing and nesting primes. Decomposition trees are just the syntax trees which arise in the "arithmetic" of flowgraphs in which the two operations are sequencing and nesting and the "numbers" are prime flowgraphs. For instance, Figure 3 shows the decomposition tree associated with the nested flowgraph in Figure 1.

3 A flowgraph grammar

The idea of flowgraph decomposition is exploited by using attributed grammars (see [1]) to define techniques for analysing flowgraphs: a divide and conquer approach whereby analysis of prime flowgraphs, and of sequenced and nested

Table 1. The flowgraph grammar

Production	Semantic Rules
1. $F \rightarrow F_1; \ldots ; F_n$	$\mu(F) = f(\mu(F_1), \ldots , \mu(F_n))$
2. $F \rightarrow P \uparrow_{v_1,\ldots,v_n} F_1, \ldots , F_n$	$\mu(F) = g_P(\mu(F_1), \ldots , \mu(F_n))$
3. $F \rightarrow P$	$\mu(F) = \text{val}(P)$
P is a prime flowgraph	

Table 2. Counting flowgraph vertices

Production	Semantic Rules
1. $F \rightarrow F_1; \ldots ; F_n$	$k(F) = 1 - n + \sum_{i=1}^{n} k(F_i)$
2. $F \rightarrow P \uparrow F_1, \ldots , F_n$	$k(F) = k(P) - 2n + \sum_{i=1}^{n} k(F_i)$
3. $F \rightarrow P$	$k(F) = \text{val}(P)$
P is a prime flowgraph	

flowgraphs is represented in terms of semantic rules. If, by "flowgraph analysis", we mean calculating some graph function μ for flowgraphs, then much of flowgraph analysis can then be expressed quite neatly in standard computing science terms within a "template" as shown in Table 1.

As an example of how this template is used, let us define a very simple flowgraph function, $k(F)$, the number of vertices. This may be calculated as shown in Table 2. The calculation assumes that the function values for prime flowgraphs are given (via the function "val"). The attributed grammar then describes how the function behaves under sequencing and nesting.

The action of the grammar on the flowgraph in Figure 1 is shown in Figure 4. The decomposition tree of Figure 3 is annotated to show how the function values are calculated "up" the tree.

The flowgraph grammar has to be interpreted in the right way if the flowgraph functions are to be well-defined. It is to be assumed that none of the F_i in rule (1) can be further decomposed as a sequence of two flowgraphs; the functions f and g_P are assumed to be defined on lists of numbers, so that they are defined in

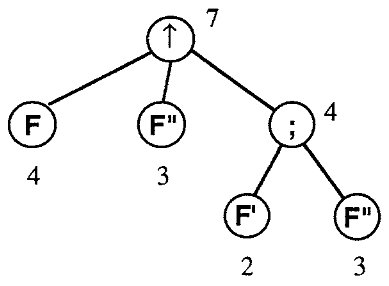


Figure 4. Calculating the number of vertices

the same way irrespective of the arity n in $F_1, \ldots, F_n$. We assume that none of $P, F_1, \ldots, F_n$ in rule (2) are the prime P_2, since

$$F \uparrow P_2 = P_2 \uparrow F = F, \tag{3.1}$$

for any flowgraph F. Moreover, $P \neq P_n$, for any $n \geq 2$ since nesting into a P_n is the same as sequencing:

$$P_{n+1} \uparrow F_1, \ldots, F_n = F_1; \ldots; F_n. \tag{3.2}$$

Unless we adopt such interpretations it becomes hard to define flowgraph functions unambiguously using the flowgraph grammar. Consider, for example, Table 3 which defines the function $d(F)$: the maximum depth of nesting within a flowgraph. The left-hand side of the identity Equation 3.2 increases d by 1; the right-hand side leaves it unchanged.

The interpretation of the grammar begs several questions which we will meet with in the next section. For now, we will identify an important class of flowgraph functions for which the grammar may be somewhat simplified.

Table 3. Depth of nesting in a flowgraph

Production	Semantic Rules
1. $F \to F_1; \ldots; F_n$	$d(F) = \max_{i=1}^{n} d(F_i)$
2. $F \to P \uparrow F_1, \ldots, F_n$	$d(F) = 1 + \max_{i=1}^{n} d(F_i)$
3. $F \to P$	$d(F) = 0$
P is a prime flowgraph	

The function g_P in rule (2) of the flowgraph grammar (Table 1) may or may not depend on the prime P. In the definition of $k(F)$ in Table 2 the role of the prime P was no different from that of the nested flowgraphs $F_1, \ldots, F_n$: g_P depended only on $k(P)$, not on the structure of the digraph P itself. For the function $d(F)$, Table 3, g_P does not even require $d(P)$, since this is always 0. This characteristic of certain flowgraph functions was formulated by Prather [14]:

Definition 2. *A flowgraph function defined as in Table 1 is called* recursive *if there is a function g such that, for all primes P,*

$$g_P(\mu(F_1), \ldots, \mu(F_n)) = g(\mu(P), \mu(F_1), \ldots, \mu(F_n)).$$

For recursive flowgraph functions we must ignore the nesting locations $v_1, \ldots v_n$ in the rule $F \rightarrow P \uparrow_{v_1,\ldots,v_n} F_1, \ldots, F_n$ because the v_i are vertices of P, whose structure is ignored in a recursive definition. This was the case in the two examples in Tables 2 and 3. It means we must make yet another assumption about the grammar: the function g must treat the list $\mu(F_1), \ldots, \mu(F_n)$ as an *unordered* list.

4 Flowgraph string grammars

Although the grammar specified in Table 1 is a convenient way of expressing flowgraph functions, it is in fact not quite standard computing science because grammars are supposed to be deterministic devices for generating sets of strings of symbols. Devices such as "$F \rightarrow F_1; \ldots; F_n$" are not allowed since they are nondeterministic. Moreover, we had to surround the grammar with "semantic" interpretations to avoid questions about the role of P_2 (identity Equation 3.1) and the relationship between sequencing and nesting (identity Equation 3.2).

Table 4. Flowgraph string grammar

Production	Semantic Rules
1. $F \rightarrow F_1; F_2$	$\mu(F) = f(\mu(F_1), \mu(F_2))$
2. $F \rightarrow F_1 \uparrow F_2$	$\mu(F) = g(\mu(F_1), \mu(F_2))$
3. $F \rightarrow (F_1)$	$\mu(F) = \mu(F_1)$
4. $F \rightarrow P$	$\mu(F) = \text{val}(P)$

P is a prime flowgraph

Table 5. String grammar version of Table 2

Production	Semantic Rules
1. $F \to F_1; F_2$	$k(F) = k(F_1) + k(F_2) - 1$
2. $F \to F_1 \uparrow F_2$	$k(F) = k(F_1) + k(F_2) - 2$
3. $F \to (F_1)$	$k(F) = k(F_1)$
4. $F \to P$	$k(F) = \text{val}(P)$
P is a prime flowgraph	

The grammar shown in Table 4 is deterministic and generates strings of symbols which may represent flowgraphs (for example, the flowgraph of Figure 1, is represented by the string "$F \uparrow (F'', F'; F'')$" which is generated by this grammar). Notice that F_1 in rule (2) does not necessarily represent a prime in this new grammar.

The flowgraph string grammar avoids the interpretive difficulties inherent in the flowgraph grammar of Table 1. But does this new grammar have the same power of expression as Table 1? Clearly, the function $k(F)$ can be expressed in the new template, as is demonstrated in Table 5.

However, it is not hard to find functions which cannot be calculated within the new template. For example, nesting is not associative: if F_3 is being nested into a vertex of F_1 in the expression $(F_1 \uparrow F_2) \uparrow F_3$, then we necessarily have

$$F_1 \uparrow (F_2 \uparrow F_3) \neq (F_1 \uparrow F_2) \uparrow F_3. \tag{4.1}$$

Thus, consider the recursive flowgraph function $d(F)$ giving the maximum depth of nesting in a flowgraph, which was defined in Table 3. Given the flowgraph $F = (P \uparrow F_1) \uparrow F_2$, the value of $d(F)$ will depend not only on $d(P \uparrow F_1)$ and $d(F_2)$, but also on whether F_2 is nested into F_1 or some other part of P. The value of $d(F)$ will be $d(P \uparrow F_1) + d(F_2)$ or $1 + \max\{d(F_1), d(F_2)\}$, respectively. This example is taken from van den Broek and van den Berg [19], who give a criterion for deciding which flowgraph functions can be defined unambiguously using the string grammar of Table 4:

Theorem 3. *Let μ be calculated for flowgraphs as shown in Table 4. Then μ is a well-defined flowgraph function if and only if the following conditions hold:*

$$g(\mu(F_1), \mu(P_2)) = g(\mu(P_2), \mu(F_1)) = \mu(F_1) \tag{4.2}$$

$$f(\mu(F_1), \mu(F_2)) = g(g(f(\mu(P_2), \mu(P_2)), \mu(F_1)), \mu(F_2)) \tag{4.3}$$

$$g(\mu(F_1), g(\mu(F_2), \mu(F_3))) = g(g(\mu(F_1), \mu(F_2)), \mu(F_3)) \tag{4.4}$$

and

$$g(g(\mu(F_1), \mu(F_2)), \mu(F_3)) = g(g(\mu(F_1), \mu(F_3)), \mu(F_2)) \tag{4.5}$$

(provided that F_3 is not nested into F_2).

Conditions 4.2, 4.3 and 4.4 ensure that Equations 3.1, 3.2 and 4.1 do not cause ambiguity. Condition 4.5 ensures that the choice of which order flowgraphs are nested does not cause ambiguity.

As an example, consider the definition of $k(F)$ given in Table 5, which we maintained was equivalent to Table 2. Let us confirm that rule (4.3) in Theorem 3 is satisfied:

$$\begin{aligned} g(g(f(k(P_2), k(P_2), k(F_1)), k(F_2)) &= g(g(f(2, 2), k(F_1)), k(F_2)) \\ &= g(g(3, k(F_1)), k(F_2)) \\ &= g(k(F_1) + 1, k(F_2)) \\ &= k(F_1) + k(F_2) - 2 \\ &= f(k(F_1), k(F_2)) \end{aligned}$$

as required. The other rules may be checked in a similar manner.

Flowgraph functions which can be defined as in Table 4 using functions f and g satisfying the conditions of Theorem 3 are called *strong* by van den Broek and van den Berg [19]. As we have seen, the depth of nesting function is not strong, and van den Broek and van den Berg [19] propose a relaxation of the definition of strong flowgraph functions to allow depth of nesting to be calculated. Effectively this relaxation re-introduces some of the semantic interpretation required for the original flowgraph grammar: in rule (2) of Table 4, it is required that F_1 cannot be expressed as a sequence, and that F_2 be maximal with respect to nesting (loosely speaking, it is not being nested into something already nested in F_1, the problem identified in Equation 4.1). They call flowgraph functions satisfying these requirements *semi-strong*.

We would like to propose a further relaxation of the definition of strong flowgraph functions as embodied in a string grammar which can capture the idea of simultaneous nesting for recursive flowgraph functions. Such a grammar is given in Table 6. The extra nonterminal symbol L represents a list of the flowgraphs to be nested into P in rule (2). Not only can this grammar allow us to define the depth of nesting function but it can also allow definitions which require a "collective" knowledge about the nested flowgraphs. For example, consider the adaptation of Prather [13] of the depth of nesting function (Table 3) defined in Table 7, where $\pi(F)$ is the number of non-process vertices in F. This flowgraph function is not semi-strong but can easily be seen to be definable using the grammar of Table 6.

Table 6. Enhanced flowgraph string grammar

Production	Semantic Rules
1. $F \rightarrow F_1; F_2$	$\mu(F) = f(\mu(F_1), \mu(F_2))$
2. $F \rightarrow P \uparrow L$	$\mu(F) = g(\mathrm{val}(P), \mu(L))$
3. $L \rightarrow L, F$	$\mu(L) = h(\mu(L), \mu(F))$
4. $L \rightarrow F$	$\mu(L) = \mu(F)$
5. $F \rightarrow (F_1)$	$\mu(F) = \mu(F_1)$
6. $F \rightarrow P$	$\mu(P) = \mathrm{val}(P)$
P is a prime flowgraph	

Table 7. Prather's μ metric

Production	Semantic Rules
1. $F \rightarrow F_1; \ldots; F_n$	$\mu(F) = \sum_{i=1}^{n} \mu(F_i)$
2. $F \rightarrow P \uparrow F_1, \ldots, F_n$	$\mu(F) = \pi(P) \max_{i=1}^{n} \mu(F_i)$
3. $F \rightarrow P$	$\mu(F) = \pi(P)$
P is a prime flowgraph	

Finally, we mention a refinement of the original flowgraph grammar due to Prather [15]. Here it is assumed that decomposition is carried on until all nested flowgraphs *or nesting locations* are identified. This means that we treat process vertices as having P_2 nested on to them. For example, from Figures 1 and 2 we can rewrite the decomposition of the flowgraph $D_4 \uparrow (D_1, P_2; D_1)$ as $D_4 \uparrow (D_1 \uparrow P_2, P_2; D_1 \uparrow P_2)$, since the two copies of D_1 have nothing nested onto their process vertices. Prather [15] then suggests assigning a nominal value of 1 to each P_2 identified by decomposition, thereby replacing Table 1 with the grammar shown in Table 8. Prather [15] calls this reduction to nested P_2s *normalization*. The motivation lies in the third reason for defining hierarchical metrics, given in Section 1: axiomatization. In the end, the purpose of control flow is to combine executable statements, therefore decomposition should reduce a flowgraph to a

Table 8. Normalized flowgraph grammar

Production	Semantic Rules
1. $F \rightarrow F_1; \ldots; F_n$	$\mu(F) = f(\mu(F_1), \ldots, \mu(F_n))$
2. $F \rightarrow P \uparrow_{v_1,\ldots,v_n} F_1, \ldots, F_n$	$\mu(F) = g_P(\mu(F_1), \ldots, \mu(F_n))$
3. $F \rightarrow P_2$	$\mu(F) = 1$
P is a prime flowgraph	

combination of executable statements. And these statements, modelled as P_2s, should have a nominal complexity of 1. The approach represents a considerable simplification but it is not clear how many flowgraph functions can still be defined in this way. It effectively replaces $\mu(F) = \text{val}(P)$ (rule (3) of Table 1), with the rule $\mu(F) = g_P(1, \ldots, 1)$, (where there are as many 1s as the number of process vertices in P. For example, Prather's μ metric (Table 7) is of this form. But it seems to rule out the kind of divide and conquer approach to flowgraph function evaluation which allows the function to be expensive to evaluate for primes as long as it is cheap for nesting and sequencing.

5 Testability measures

Testability measures ([2]) attempt to quantify aspects of the structural complexity of code which might give useful information about the testing stage of software production. In this section we describe recent work in defining these metrics using the flowgraph grammar template.

The path followed through a program when it is executed from a particular state with a specific input is mirrored by a walk through the flowgraph which models that program. Intuitively, describing the set of possible walks through the flowgraph should provide information relating to the testability of the program. One way of characterising these walks is to enumerate different classes of walks. For instance we might count the number of different trails through a flowgraph or the number of walks through a flowgraph which do not pass through any vertex more than k times (k-walks). These counts are then viewed as measurements of a program using the flowgraph as an intermediate model. Many of these enumerations are known to be NP-Hard for flowgraphs in general. Efficiency in their calculation therefore becomes reliant on exploiting knowledge of likely flowgraph populations derived from modeling programs. The attributed grammar description of measurements discussed above is a way by which some of the testability measures can be efficiently calculated for flowgraphs derived from structured programs (i.e. those which are constructed by nesting and sequencing a small well-behaved set of primes: we shall use those in Figure 2). As an example we take a definition from ([3]) of the Number of Trails, $t(F)$ in a flowgraph F.

Suppose that $F = (P \uparrow F_1, F_2, ..., F_n)$ is the flowgraph constructed by nesting the flowgraphs $F_1, F_2, ..., F_n$ onto P, a prime flowgraph possessing n process vertices. Assume that the order of the flowgraphs $F_1, F_2, ..., F_n$ prescribes which process vertices they are nested onto. To calculate the number of trails in the flowgraph F it is not enough to know the number of trails in $F_1, F_2, ..., F_n$. If, in addition, the number of ordered pairs of edge-disjoint trails of $F_1, F_2, ..., F_n$ are known it becomes possible to calculate the number of trails in F, provided

Table 9. Definition of the number of $s - t$ trails $t(F)$

Production	Semantic Rules	
$F \rightarrow F_1; \ldots ; F_k$	$t(F) = \prod_{i=1}^{k} t(F_i)$	$\hat{t}(F) = \prod_{i=1}^{k} \hat{t}(F_i)$
$F \rightarrow D_1 \uparrow F_1$	$t(F) = 1 + t(F_1)$	$\hat{t}(F) = 2\hat{t}(F_1)$
$F \rightarrow D_2 \uparrow F_1$	$t(F) = 1 + t(F_1)$	$\hat{t}(F) = 0$
$F \rightarrow D_3 \uparrow F_1$	$t(F) = t(F_1) + \hat{t}(F_1)$	$\hat{t}(F) = 0$
$F \rightarrow D_4 \uparrow F_1, F_2$	$t(F) = t(F_1) + \hat{t}(F_1)t(F_2)$	$\hat{t}(F) = 0$
$F \rightarrow K_{2,k} \uparrow F_1, \ldots, F_k$	$t(F) = \sum_{i=1}^{k} t(F_i)$	$\hat{t}(F) = 2\sum_{i=1}^{k-1} \sum_{j=i+1}^{k} t(F_i)t(F_j)$
$F \rightarrow W_k \uparrow F_1, \ldots, F_k$	$t(F) = 1 + \sum_{i=1}^{k} \prod_{j=1}^{i} t(F_j)$	$\hat{t}(F) = 2\sum_{i=1}^{k-1} \prod_{j=1}^{i} t(F_j)$
$F \uparrow P_k, k \geq 2$	$t(F) = 1$	$\hat{t}(F) = 0$
$F \uparrow D_1$	$t(F) = 2$	$\hat{t}(F) = 2$
$F \uparrow D_2$	$t(F) = 2$	$\hat{t}(F) = 0$
$F \uparrow D_3$	$t(F) = 1$	$\hat{t}(F) = 0$
$F \uparrow D_4$	$t(F) = 1$	$\hat{t}(F) = 0$
$F \uparrow K_{2,k}, k \geq 2$	$t(F) = k$	$\hat{t}(F) = k(k-1)$
$F \uparrow W_k, k \geq 2$	$t(F) = k + 1$	$\hat{t}(F) = 2(k-1)$

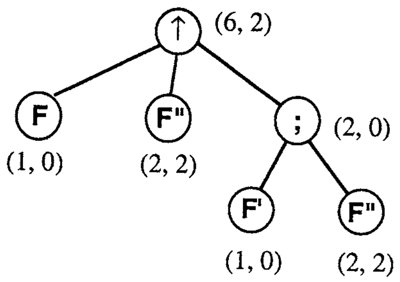


Figure 5. Calculating the number of trails

P is structured. This additional attribute, the *number of ordered pairs of edge-disjoint trails* will be denoted by $\hat{t}$.

Table 9 shows the formulae which enable $t(F)$ to be calculated for flowgraphs which decompose into the structured primes.

The calculation of $t(F)$ is not recursive: a different g function is here used for each prime P which is the target of nesting. Moreover, we have resorted to using a pair of attributes, a technique introduced in [20]. The evaluation of $t(F)$ for the flowgraph in Figure 1 is shown in Figure 5 ($t(F)$ is the first number in each pair, $\hat{t}(F)$ is the second). This flowgraph function seems a long way from the strong functions of van den Broek and van den Berg or the normalized functions of Prather, but our understanding of what exactly is this distant relationship is still slight.

6 Concluding remarks

We have seen that the basic idea of defining flowgraph functions hierarchically reveals some subtleties once we try to be precise about the way in which the definitions are specified. An attributed flowgraph grammar is a convenient format for the definitions but has to be hedged about with interpretation rules. If we want to stray strictly within the bounds of formal language theory then we can use string grammars for our definitions but this seems to be at the expense of some expressive power: in our most extreme example, the hierarichal definition of the testability metric "number of trails" is well beyond what is easy to express using a string grammar.

In Section 1, we gave three justifications for making hierarchical definitions. The first of these, divide and conquer, is not affected by our discussion—provided a flowgraph function definition is sufficiently clear for an algorithm to be implemented from it, we will derive the benefits of greater efficiency no matter what the format of the definition. This is clearly the case for the number of trails: the calculation would in general require an enumeration of the actual paths;

Table 9 shows that for structured flowgraphs we can do much better than this. Nevertheless, it would presumably be much easier to implement algorithms expressed in terms of formal language theory, *and to prove these implementations correct.* For instance, parser generators such as YACC, allow attributed grammars to be implemented with a high degree of ease and reliability. Such parser generators could in principle be used for the string grammars in Tables 4 and 6, but not for the grammar of Table 1.

The second justification, achieving a unification of metric definitions, is more affected: it does not seem to be easy to make this unification while at the same time keeping the definitions as simple and natural as possible. Some flowgraph functions can be defined using string grammars, for others it is not obvious that this is possible; some functions can be normalized, others can not. The most we can say is that attributed grammars offer a possible way of characterizing this diversity, and we have tried to make a very limited start in the present paper. At any rate we should take note that we are not comparing like with like when we compare complexity metrics.

The third justification for hierarchical definitions, axiomatization, is also affected by our analysis. We must be able to explore the consequences of axioms using some mathematical machinery. van den Broek and van den Berg [19] use the theory of functional languages. We have suggested using the theory of formal languages. Either way, so far, the class of metrics which seem to be amenable to analysis is very limited.

Acknowlegement

This research was supported by EPSRC under grant J18880 as part of the DTI/EPSRC programme on Safety-critical Systems. We are grateful to Professor Ron Prather for clarifying the motivation behind his normalization process. The comments of an anonymous referee improved the readability of this paper.

References

1. Aho, A.V., Sethi, R. and Ullman, J.D. (1986). *Compilers. Principles, Techniques and Tools*, Addison-Wesley, Reading, MA Publishing Company, Reading, Massachusetts.

2. Bache, R. and Müllerburg, M. (1990). Measures of testability as a basis for quality assurance. *Software Engineering J.*, **5**, 86–92.

3. Bainbridge, J. (1994). Defining testability metrics axiomatically. *Software Testing, Verification and Reliability*, **4**, 63–80. (To appear.)

4. Cowell, D.F., Gillies, D.F. and Kaposi, A.A. (1980). Synthesis and structural analysis of abstract programs. *Comput. J.*, **23**, 243–247.

5. Fenton, N.E., Whitty, R.W. and Kaposi, A.A. (1985). A generalised mathematical theory of structured programming. *Theoret. Comput. Sci.*, **36**, 145–171.

6. Fenton, N.E. (1991). *Software Metrics: A Rigorous Approach*, Chapman and Hall, London.

7. Fenton, N.E. and Hill, G. (1992). *Systems Construction and Analysis—A Mathematical and Logical Framework*, McGraw-Hill, London.

8. Fuchs, N. and Stainer, S. (1992). Language independent definition of axiomatic metrics. *Formal Aspects of Measurement*, Editors: B.T. Denvir, R. Herman and R.W. Whitty, Springer-Verlag, London, 84–107.

9. Hecht, M.S. (1977). *Flow Analysis of Computer Programs*, Elsevier North Holland, New York.

10. Linger, R.C., Mills, H.D. and Witt, B.I. (1979). *Structured Programming: Theory and Practice*, Addison-Wesley, New York.

11. Maddux, R.A. (1975). A study of computer program structure. *PhD Thesis*, University of Waterloo, Ontario, Canada.

12. Prather, R.E. and Giulieri, S.G. (1981). Decomposition of flowchart schemata. *Comput. J.*, **24**, 258–262.

13. Prather, R.E. (1984). An axiomatic theory of software complexity measure. *Comput. J.*, **27**, 340-347.

14. Prather, R.E. (1987). On hierarchical software metrics. *Software Eng. J.*, **2**, 42–45.

15. Prather, R.E. (1993). Hierarchical metrics and the prime generation problem. *Software Eng. J.*, **8**, 246–252.

16. Prather, R.E. (1994). Axiomatic foundations for structural software metrics. *2nd Int. Sym. Proc. Software Metrics*, IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 36–44.

17. Urschler, G. (1975). Automatic structuring of programs. *IBM J. Res. Develop.*, 181–193.

18. van den Berg, K.G. and van den Broek, P.M. (1994). Axiomatic testing of structure metrics. *2nd Int. Sym. Proc. Software Metrics*, IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 45–53.

19. van den Broek, P.M. and van den Berg, K.G. (1995). Generalised approach to software structure metrics. *Software Eng. J.*, **10**, 61–67.

20. Whitty, R.W. (1992). Multidimensional software metrics. *Formal Aspects of Measurement*, Editors: B.T. Denvir, R. Herman, and R.W. Whitty, Springer-Verlag, London, 148–169.

21. Woodward, M.R., Hedley, D. and Hennell, M.A. (1980). Experiences with path analysis and testing of programs. *IEEE Trans. Software Eng.*, 6, 278–286.

22. Zuse, H. (1991). *Software Metrics: Tools and Techniques*, de Gruyter, Berlin.

Applications of Combinatorics to Security

Simon R. Blackburn[1] **and Fred Piper**

Department of Mathematics, Royal Holloway, University of London

1 Introduction

There are now many areas where users communicating over public channels require either secrecy or assurance of the integrity of their data. Not surprisingly, cryptography is one of the main tools used to meet these requirements and this use of cryptography leads to a number of interesting problems in a wide range of mathematical areas.

In this paper we discuss three particular security requirements and show how they lead to interesting problems in three different branches of combinatorics. The paper places the emphasis on how the security application gives rise to the mathematical problems rather than concentrating on the detailed proofs of the mathematical solutions.

The basic idea of a cipher system is given in Diagram 1. The fundamental requirement is that the cryptogram should be a scrambled version of the message so that unauthorised eavesdroppers who obtain the cryptogram cannot determine the message. Good cryptographic practice suggests that we should assume that the attacker will know the algorithm being used. Thus the security of the system relies on the secrecy of the deciphering key. Clearly, the need to keep this key secret is of paramount importance and key management is one of the most important features of a secure system.

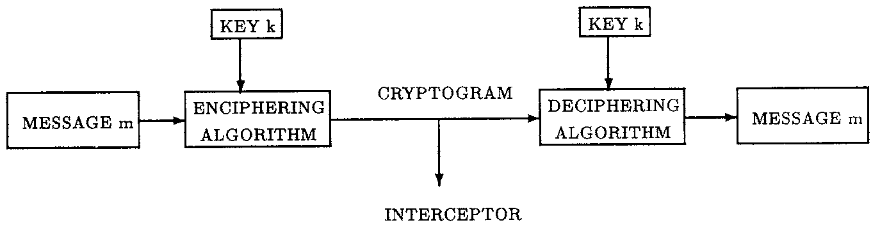


Diagram 1.

[1]Supported by EPSRC Research Grant GR/H23719

Our first problem arises from key management problems associated with large networks and our final two problems are associated with the need for obtaining strong algorithms with large key spaces.

2 Incidence structures

In this section we consider the key management problem in a large network where communicating nodes use symmetric key cryptography to provide end-to-end encryption. Many interesting mathematical applications arise and we concentrate on the use of combinatorics to facilitate a number of important key management issues. Motivated by a recent PhD thesis [30], we describe a combinatorial incidence structure, known as a Key Distribution Pattern. This structure uses subkeys to both reduce storage requirements at the nodes and allow direct secure communication between nodes without further recourse to any central key distribution location. The competing interests of reduced storage and good security pose problems which find solutions in the use of certain combinatorial incidence structures and geometrical configurations.

2.1 Network security

The simplest form of security provided in a network is link encryption. This protects data as it is transmitted along a single communications link joining two nodes. For link encryption, a shared key is installed at each end of a direct link and this is then used to encrypt and decrypt data transmitted between those two ends. Since each node need only share secret keys with its adjoining nodes, the key management problem for link encryption is relatively simple. When a shared key is required for a given link, it may be generated at one end of that link, and local arrangements can be introduced to distribute it to the other end of the link. However, the security offered by link encryption is limited. Data transmitted between two remote nodes may pass through several intermediary nodes. At each node it needs to be decrypted and then re-encrypted under a different key. This key translation process means that it may be vulnerable.

End-to-end encryption refers to the situation where data transmitted between two nodes is encrypted under a secret key shared by those two nodes. This achieves a logical separation of nodes which do not share keys. Although a node may carry (encrypted) data from one node destined for another, it does not need to process that data.

Clearly, two nodes that need to communicate securely must share a common key. Furthermore, greater security is achieved if different keys are used by each pair of communicating nodes. This minimizes the damage caused if a key is compromised and prevents surreptitious or accidental decryption by unintended recipients. However, it also increases the complexity of the problems associated with key management. In particular, keys must be distributed to remote nodes and there may be the need for every pair of nodes to share a unique secret key.

2.2 Key management

The first difficulty to note is the sheer volume of key material that may be required. A network of n nodes in which every pair of nodes shares a key requires $\frac{1}{2}n(n-1)$ keys. If each node were to store all the keys it needed then each node would need to store $n-1$ keys. A large network may consist of some tens of thousands of nodes. If every pair of nodes in a network of 10,000 nodes were to share a DES key of 56 bits then we might require of the order of 5×10^7 keys or 3 Gbits of key material. Moreover, each node would need to store of the order of 10^4 keys, i.e. about a megabit of key material. The problem is compounded by the need to change keys on a regular basis. If, as is often the case, a node needs to store several old keys as well as the current one, then there will be an even heavier storage burden on the nodes.

In order to initialise the security mechanisms, a network is likely to require some facility, in the role of a Key Distribution Centre (KDC), to generate and distribute these keys. This KDC would also keep a register of which nodes own what keys (both current and old versions). Clearly, the storage requirement at the KDC could be very large indeed. Further, in order to distribute the keys, the KDC must be able to communicate securely with every node. This requirement necessitates either having a separate secure channel or using the network with the transmissions secured by another key. There are many practical solutions to this problem. For example, the KDC may establish with each node a local key which is used to encrypt only those messages which contain the keys to be distributed. These local keys will then need to be protected by another key, thus creating a key hierarchy. In most situations the top (master) key will be used infrequently and, as a result, a public key cryptosystem such as RSA could be used.

Whatever system is used, the requirements of a key hierarchical system to install all keys that allow end-to-end encryption between any two nodes is likely to give rise to considerable storage problems at both the KDC and the user nodes.

2.3 Subkeys

One solution to the problem utilises the concept of a trusted centre to generate and distribute keys as required, thereby exchanging a storage overhead for one in communications. An alternative to accepting increased communication as a means of reducing storage is to relax the requirement that the $\frac{1}{2}n(n-1)$ keys, that may be required to secure every pair of nodes in a network of n nodes, are independent. Several authors have taken this approach [7,8,15,28,29]. One solution is the following. Instead of generating $\frac{1}{2}n(n-1)$ independent keys (and distributing $n-1$ of these to each node) the KDC could generate a collection of subkeys and distribute a subset of these to each node. Any pair of nodes could determine a key to secure their communication channel by combining the subkeys they have in common. Thus, after the KDC has distributed the subkeys, the nodes have no need for any further direct communication with the KDC.

In the original scheme there is a unique key among the $\frac{1}{2}n(n-1)$ independent keys which is common to a pair of nodes and no other node holds this key (among his $n-1$ keys). If, however, subkeys are used then it is possible that other nodes hold one (or more) of the subkeys that the two nodes combine to produce their key. Thus these other nodes may have some information about this key. This disadvantage may be tolerated for the potential advantage in reduced storage. For example, if a node holds m subkeys and only 2 subkeys are combined to produce a key then we have $\frac{1}{2}m(m-1)$ different possible combinations. Thus m need only be of the order of $\sqrt{2n}$ in order to provide distinct (though not necessarily all independent) keys for communication with each of the other nodes. So, in a network of 10,000 nodes, each node need only store about 8×10^3 bits rather than the 10^6 bits we calculated earlier in order to share a DES key with every other node.

2.4 A small example

To illustrate the use of subkeys we use the biplane with $v = 7$ and $k = 4$. There are seven nodes, $v_1, \ldots, v_7$ in the network. The KDC generates seven subkeys $k_1, \ldots, k_7$ and distributes them so that each node receives four subkeys as follows:

v_1 :	k_1	k_2	k_3	k_5
v_2 :	k_2	k_3	k_4	k_6
v_3 :	k_3	k_4	k_5	k_7
v_4 :	k_4	k_5	k_6	k_1
v_5 :	k_5	k_6	k_7	k_2
v_6 :	k_6	k_7	k_1	k_3
v_7 :	k_7	k_1	k_2	k_4

It is easy to check that, in this scheme, any two nodes have two subkeys in common. Moreover, only these two nodes have both these subkeys. There are 21 pairs of subkeys and these correspond to the 21 pairs of nodes. Thus each pair of nodes may obtain their own key by combining their two common subkeys.

2.4.1 Storage reduction and security for the example

If each subkey consists of 28 bits then the two common subkeys may be concatenated to obtain a 56 bit DES key. In this case each node stores $4 \times 28 = 112$ bits rather than $6 \times 56 = 336$ bits required for 6 independent DES keys. Also, the KDC generates and stores $7 \times 28 = 196$ bits rather than $21 \times 56 = 1176$ bits. However, the disadvantage of this scheme is that a node may know as many as 28 of the 56 bits of a DES key that a pair of nodes shares. That node would then need only 2^{28} trials to determine the key. Moreover, v_3 and v_5 together hold both k_2 and k_3 and so can determine the key used by v_1 and v_2. This scheme provides a marked reduction in storage at the cost of significantly decreasing the security and sacrificing the independence of keys. Greater security can be obtained by using the same biplane but increasing the storage by increasing the size of the subkeys.

If, for instance, each subkey consists of 56 bits, so that the storage at each node is doubled to 224, which is still a reduction on the 336 bits, each pair of nodes can obtain a DES key by XORing their common subkeys. Now, as any other node holds at most one of the two subkeys, no single other node knows as much as a single bit of any key which is shared with any other node. However, as before, v_3 and v_5 can determine the key used by v_1 and v_2.

2.5 Keys from subkeys

There are two aspects of the general use of subkeys that must be discussed. Firstly, there is the question of the distribution of the subkeys to the nodes, i.e. the determination of which subset of subkeys each node is given. Clearly, the subkeys each node holds must be labelled in such a way that it can determine which subkeys it shares with any other node. It is also important to protect each pair of nodes from other users. Secondly, there is the question of how each one of a pair of nodes combines the subkeys that they have in common to produce a key.

There are many ways of tackling the first problem. However, we will concentrate on those which are applications of the study of finite incidence structures.

2.6 Finite incidence structures

A finite incidence structure $\mathcal{S} = (\mathcal{P}, \mathcal{B}, \mathcal{I})$ consists of two finite non-empty sets $\mathcal{P}$ and $\mathcal{B}$ and an incidence relation $\mathcal{I} \subseteq \mathcal{P} \times \mathcal{B}$. The elements of $\mathcal{P}$ are called points and those of $\mathcal{B}$ blocks. If $(P, x) \in \mathcal{I}$, where $P \in \mathcal{P}$ and $x \in \mathcal{B}$, then we say that P is incident with x (or P is on x, or x contains P). Otherwise P and x are not incident [14].

We relate incidence structures to our key distribution problem by identifying the set of nodes of the network and the set of subkeys with the set of points and set of blocks, respectively, of an incidence structure. A node P and a subkey x are incident if and only if subkey x is one of the subkeys distributed to node P.

2.7 Key Distribution Patterns

We have noted that, with this method of using subkeys to determine keys, some nodes may have partial information about a key that is shared by a pair of nodes. That is, they may know some of the subkeys that are combined to produce that key. A basic requirement that we may ask of a scheme such as this is that no single node should hold all of the subkeys that are common to a pair of nodes (except, of course, the two nodes of this pair). We may state this requirement in incidence structure terms as follows. For any three distinct points P_1, P_2, Q of the finite incidence structure $|(P_1) \cap (P_2) \setminus (Q)| \geq 1$ where, for any point P, the set of blocks incident with P is denoted (P).

A finite incidence structure with this property is called a Key Distribution Pattern (KDP) [28,29]. If P_1, P_2 are distinct points of a finite incidence structure the *line* determined by P_1 and P_2 is defined to be the set of all points which are incident with every block in $(P_1) \cap (P_2)$. It is relatively easy to show that a KDP is a finite incidence structure in which every line has size 2 and that, conversely, a finite incidence structure with line size 2 is a KDP [30]. A design in which every pair of points lie on a unique block is trivial and has $\frac{1}{2}v(v-1)$ blocks. We call the corresponding KDP trivial. Note that in a trivial KDP the subkeys are keys.

2.7.1 w-secure KDPs

If subkeys are held by more than two nodes then it is (theoretically) possible that some group of nodes may hold all the subkeys that two other nodes have in common. Such a group of nodes could then pool their subkeys and determine the key that that pair of nodes would use. We say that a KDP is *w-secure* if no group of w or fewer nodes can determine another pair's key. Thus a KDP is w-secure if for every pair of distinct points P_1 and P_2 and every subset of w points $Q_1, \ldots, Q_w$ (distinct from P_1 and P_2) we have $|(P_1) \cap (P2) \setminus \cup_{i=1}^{w}(Q_i)| \geq 1$. We note that every KDP is 1-secure. For every KDP there will be a maximum value of w such that the KDP is w-secure. Usually, for a given network, a decision will need to be made on the value of w required for the w-security of the KDP and this will then influence the way in which subkeys are distributed. Of particular interest to network providers are the KDPs which achieve the desired level of w-security and have the lowest storage requirement.

2.7.2 Storage

The storage requirement takes two forms: storage at the nodes and storage at the KDC. We may assume, by splitting (larger) subkeys if necessary, that all subkeys have the same size. When we do this the storage requirement is given by the number of blocks containing a point ($|(P)|$ for $P \in \mathcal{P}$) and the total number of blocks ($|\mathcal{B}|$) in the KDP. Usually minimising the storage at the nodes is more important. However, depending on the application, users concern may also focus on the average nodal storage or on the maximum storage at a single node. There is another factor which is relevant in discussing storage requirements; this is the size of the key that pairs of nodes share. If this latter consideration is important then the way in which subkeys are combined to produce the key becomes particularly relevant.

If a key is obtained by the concatenation of subkeys then knowledge of any subkey gives part of the key and reduces the size of an exhaustive key search. Thus concatenation is not a sensible method for combining subkeys. Clearly we need a method of combining subkeys that produces a key of the appropriate size but where knowledge of some of the subkeys will not shorten an exhaustive key search. One option might be to let subkeys and keys have the same length and to use XOR as the combining operation. There are many alternatives including the use of orthogonal arrays (or MDS codes).

The use of orthogonal arrays allows any pair of nodes P_1, P_2 to share a key whose size is equivalent to the concatenation of m independent subkeys and which is secure against collusion of up to w other nodes provided that the corresponding KDP satisfies $|(P_1)\cap(P_2)\setminus\cup_{i=1}^{w}(Q_i)| \geq m$ for all w-subsets $\{Q_1,\ldots,Q_w\}$ disjoint from $\{P_1,P_2\}$. When this condition holds for every pair of nodes Quinn [30] says that the KDP has w-residue m. These parameters set the security requirements of the network. The storage requirements of the nodes are reflected in the values $|(P)|$ for $P \in \mathcal{P}$ and that of the KDC in $|\mathcal{B}|$. The greatest key storage reduction is obtained when these values are minimized. It is a challenge to find KDPs in which these values are minimal.

When $w = n-2$ the trivial KDP provides the minimum storage. The size of the key equals the size of a subkey and $|(P)| = n-1$ for every node.

In order to be able to improve upon this storage of the trivial KDP we must have $w < n-2$ and some node P must hold a subkey which contributes to more than one of his keys. Quinn shows that in the case $|(P)| \geq \frac{1}{2}(w+1)(w+2m)$.

In a KDP with w-residue m a pair of nodes may use a key of size m times the size of a subkey. A reduction in storage at a node over that of the trivial KDP is possible only if $\frac{(w+1)(w+2m)}{2m} < n-1$. Quinn considers KDPs in which each subkey is held by the same number of nodes and every pair of nodes have the same number of subkeys in common. She obtains a lower bound on the storage at a node in such a KDP, showing that $|(P)| \geq \frac{1}{2}(1+\sqrt{1+4(w+m)(n-1)})$. (This is equivalent to $|(P)|(|(P)|-1) \geq (w+m)(n-1)$.)

2.7.3 Biplanes

A biplane is a KDP with 1-residue 1. Indeed, any pair of points are incident with 2 blocks and any other point is on at most one of these blocks. So for any three distinct points P_1, P_2, Q we have $|(P_1)\cap(P_2)\setminus(Q)| \geq 1$. In a biplane with n points, every point is incident with k blocks where $k(k-1) = 2(n-1)$. Thus biplanes attain Quinn's lower bound for this class of KDPs [30]. Biplanes have equally many blocks as points and also attain the lower bound $|\mathcal{B}| \geq n$ for storage at the KDC. Unfortunately, no biplane with more than 79 points is known. So we must look elsewhere for KDPs suitable for large networks.

We consider KDPs in which every pair of nodes have the same number, s, of subkeys in common and each subkey is held by the same number, k, of nodes. Let λ be the maximum, over all triples P_1, P_2, P_3 of nodes, of $|(P1)\cap(P2)\cap(P3)|$. If $\lambda = 0$ then any subkey is held by exactly two nodes and we have nothing more than the trivial KDP. Thus we can only achieve some storage reduction with $\lambda \geq 1$. Quinn [30] shows that such a KDP is a w-KDP with w-residue m for any w and m with $w\lambda + m \leq s$. She considers the case $\lambda = 1$ and constructs many examples. The basis of her method is her observation that if $k \geq 3$ then $\lambda = 1$ if, and only if, any pair of blocks have at most two points in common. One of Quinn's constructions uses conics in desarguesian projective planes.

2.7.4 Example 2 (Quinn's Construction)

The desarguesian projective plane of order q has q^2+q+1 points and q^2+q+1 lines (blocks). Every line is incident with $q+1$ points and any pair of points are incident with exactly one common line. A conic is a set of $q+1$ points no three of which belong to any line. It may be shown that the desarguesian projective plane contains a collection of q^2+q+1 conics with the properties that any two of them have exactly one point in common. The structure whose points are the points of the plane and whose blocks are the lines and such a collection of conics clearly satisfies Quinn's criterion for the size of block intersection. The structure is therefore a 1-KDP with 1-residue $m = 1$ ($s = 2, \lambda = 1$).

Quinn generalises her construction by considering incidence structures in which every pair of points has either 0 or s blocks in common. Techniques similar to the example above allow her to construct such structures with the property that any two blocks have at most two points in common. Pairs of nodes having s common subkeys then have m subkeys not held by any group of w other nodes whenever $w+m \leq s$. By introducing m separate subkeys for each pair of nodes with no common subkey we can obtain a w-KDP with w-residue m.

3 Enumerating permutations

In this section we will consider some permutation enumeration questions which can be motivated both from a pure and a practical point of view. We begin by looking at a particular speech scrambling system. For an introduction to speech scrambling techniques, see [3].

A time element speech scrambler is a method of encrypting analogue speech signals which works as follows. A speech signal M is divided up into segments $M_0, M_1, \ldots, M_{l-1}$ of equal duration (T seconds, say). The scrambler permutes the segments of M according to a secret permutation σ of the set $\{0, 1, \ldots, l-1\}$ and then transmits the signal $C = M_{0\sigma}, M_{1\sigma}, \ldots, M_{(l-1)\sigma}$ consisting of the segments of M in this permuted order. The descrambler recovers M by reordering the segments of C.

The permutation σ used in time element speech scrambler systems should be chosen carefully. There are two main requirements that σ should satisfy: it should be chosen so as to make the resulting system secure (in particular, so as to make the signal C unintelligible to any evesdropper) and so as to allow the scrambler to operate efficiently. We will discuss the security criteria for σ at the end of the section, but we start by showing how σ should be chosen for efficiency.

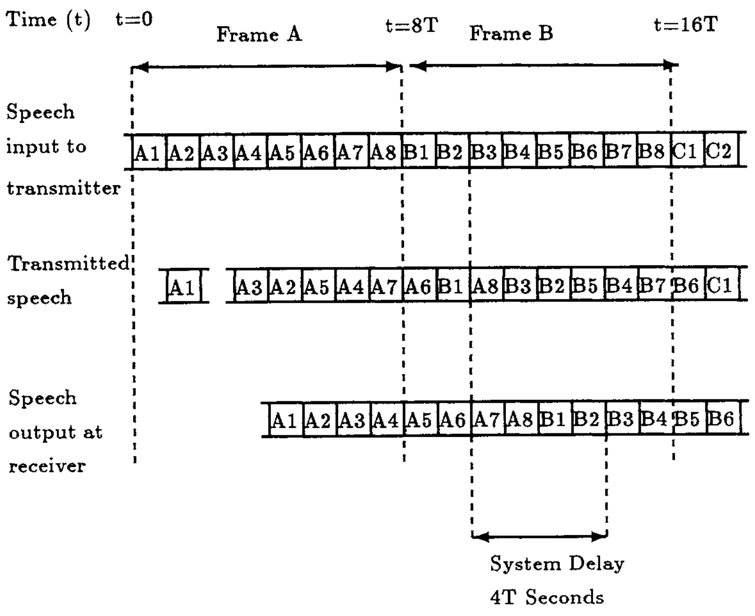
We discuss two techniques: overlapping frame sliding window scramblers and disjoint frame sliding window scramblers. Both these techniques are discussed in [26]. See also [3,27].

3.1 Overlapping frame speech scrambling

In the first technique, we divide M into sets of n consecutive segments, called frames. We pick an integer k such that $k < n$ and a permutation π from the set

$$A(n,k) := \{\pi \in S_n : i\pi \in \{i-1, i-2, \ldots, i-k\} \text{ for all } i \in \{0, \ldots, n-1\}\}$$

where S_n is the set of permutations of $\Omega := \{0, 1, \ldots, n-1\}$ and where we are regarding the elements of Ω as integers modulo n. Suppose the message starts to enter the scrambler at time 0. We then scramble the message by transmitting at time tT the segment received at time $(t-r)T$, where r is defined to be the smallest nonnegative integer such that $(t \text{ modulo } n)\pi = (t-r)$ modulo n. One can show that the total system delay will be $(k+1)T$ seconds when using this technique. An overlapping frame sliding window scrambler is shown in Figure 1, where $n = 8$ and we are using the permutation $\pi = (0541)(2763)$.



Time (t) t=0
Frame A
t=8T
Frame B
t=16T
Speech input to transmitter
A1 A2 A3 A4 A5 A6 A7 A8 B1 B2 B3 B4 B5 B6 B7 B8 C1 C2
Transmitted speech
A1 A3 A2 A5 A4 A7 A6 B1 A8 B3 B2 B5 B4 B7 B6 C1
Speech output at receiver
A1 A2 A3 A4 A5 A6 A7 A8 B1 B2 B3 B4 B5 B6
System Delay 4T Seconds

Figure 1. Overlapping frame sliding window scrambling

3.2 Disjoint frame speech scrambling

The second technique is known as disjoint frame sliding window scrambling. As before, we partition the sequence of segments into frames of length n. In contrast to the previous technique, we aim to scramble the message within each frame, rather than mixing the frames together. This allows us to use a different permutation when scrambling each frame, so increasing the security of the system. We select nonnegative integers s and t such that $s + t + 1 \leq n$. For each frame, we choose a permutation π satisfying

$$i\pi \in \begin{cases} \{0, 1, \dots, i+t\} \text{ if } i \in \{0, 1, \dots, s-1\}, \\ \{i-s, i-s+1, \dots, i+t-1, i+t\} \text{ if } i \in \{s, \dots, n-1-t\} \text{ and} \\ \{i-s, i-s+1, \dots, n-1\} \text{ if } i \in \{n-t, \dots, n-1\} \end{cases} \tag{3.1}$$

and permute the segments of the frame according to π. The total system delay in a system of this type is $(s + t + 2)T$ seconds. See Figure 2 for an example of this technique, where $n = 8$, $s = t = 1$ and where we use the permutation (01)(2)(34)(56)(7) to scramble the first frame A and the permutation (0)(12)(34)(5)(67) to permute the second frame B.

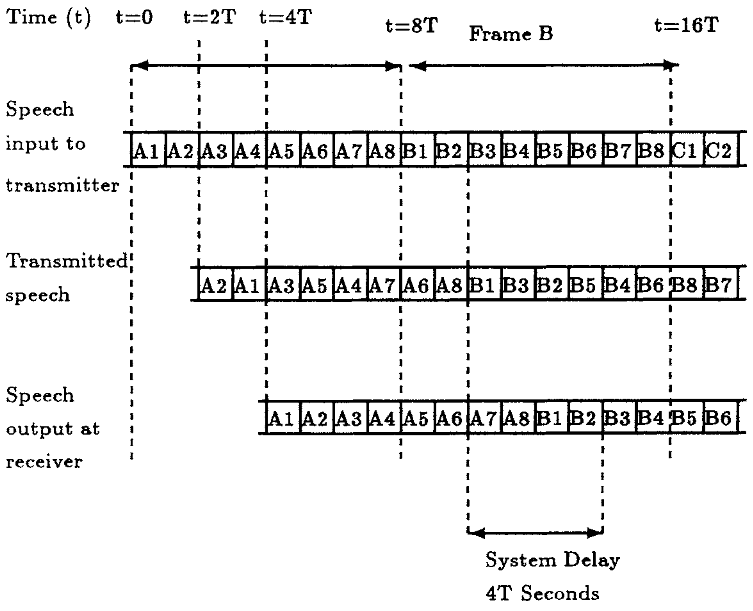

Time (t)
t=0
t=2T
t=4T
t=8T
Frame B
t=16T
Speech input to transmitter
A1 A2 A3 A4 A5 A6 A7 A8 B1 B2 B3 B4 B5 B6 B7 B8 C1 C2
Transmitted speech
A2 A1 A3 A5 A4 A7 A6 A8 B1 B3 B2 B5 B4 B6 B8 B7
Speech output at receiver
A1 A2 A3 A4 A5 A6 A7 A8 B1 B2 B3 B4 B5 B6
System Delay 4T Seconds

Figure 2. Disjoint frame sliding window scrambling

3.3 Enumeration questions

In order for our speech scrambling system to be secure when using one of the sliding window techniques above, there must be a "reasonable" number of permutations π satisfying the constraints imposed above (for otherwise an evesdropper could just try every suitable permutation until they hit upon the correct one). The two techniques above explain why designers of speech scrambling systems are interested in determining (or at least estimating) the size of the sets

$$A(n,k) := \{\pi \in S_n : i\pi \in \{i-1,\dots,i-k\} \text{ for all } i \in \{0,\dots,n-1\}\}$$

and

$$B(n,s,t) := \{\pi \in S_n : \pi \text{ satisfies Equation 3.1}\}.$$

The sets $A(n,k)$ and $B(n,s,t)$ (where $s=t$) have been studied [2,26] with the speech scrambling motivation in mind. However, the study of these sets can also be motivated from classical combinatorics. Finding the order of the set $A(n,n-1)$ is the "problème des rencontres" and of the set $A(n,n-2)$ is the "problème des ménages". So enumerating the elements of $A(n,k)$ can be regarded as generalising these well known combinatorial problems. Both enumeration problems can be phrased in terms of evaluating the permanent of an n by n matrix (see [24] or [25] for background material on permanents). Indeed, the problem of evaluating $|A(n,k)|$ has been studied in this guise by combinatorialists [23]. For the remainder of this section, we follow the approach to these enumeration questions found in [5].

We can enumerate the elements in $A(n,k)$ by first dividing them into k classes $A(n,k,0), A(n,k,1),\dots,A(n,k,k-1)$ where we define $A(n,k,r)$ to be the set

$$\{\pi \in A(n,k) : |\{i \in \{1,2,\dots,k-1\} : i\pi \in \{0,1,\dots,i-2\}\}| = r\}.$$

It is possible to establish a bijection between the elements in $A(n,k,r)$ and closed paths of length n in a certain strongly connected directed graph $\Gamma(k,r)$. We may then use the incidence matrix of $\Gamma(k,r)$ both to explicitly find $|A(n,k,r)|$ for small parameters (using a computer) and to investigate the asymptotic behaviour of $|A(n,k,r)|$ using Perron–Frobenius theory. We can show:

Theorem 1. *Let $H(k,r)$ be the incidence matrix of the graph $\Gamma(k,r)$. Define $a(n,k,r) := |A(n,k,r)|$. Then $a(n,k,r) =$ Trace $(H(k,r)^n)$. Also $a(n,k,r) \sim (\mu_{k,r})^n$ as $n \to \infty$ where $\mu_{k,r}$ is the (unique) maximum eigenvalue of $H(k,r)$.*

If we define $\mu_k := \max\{\mu_{k,r} : 0 \le r \le k-1\}$, the theorem implies that $|A(n,k)| \sim m_k(\mu_k)^n$ as $n \to \infty$ where $m_k := |\{r : \mu_{k,r} = \mu_k\}|$.

The values of $\mu_{k,r}$ for $k \le 15$ are shown in Table 1. Only the values of $\mu_{k,r}$ for $r \le \frac{k-1}{2}$ are shown, since it is not difficult to show that $\mu_{k,r} = \mu_{k,k-1-r}$. The

Table 1. The maximum eigenvalue $\mu_{k,r}$ of $H(k, r)$ for small k

k	$r =$						
	1	2	3	4	5	6	7
3	1.61803						
4	1.83929						
5	1.92756	2.33355					
6	1.96595	2.60771					
7	1.98358	2.76284	3.06177				
8	1.99196	2.85335	3.36009				
9	1.99603	2.90768	3.55755	3.79352			
10	1.99803	2.94107	3.69013	4.10597			
11	1.99902	2.96197	3.78052	4.33205	4.52677		
12	1.99951	2.97526	3.84303	4.49690	4.84856		
13	1.99976	2.98381	3.88682	4.61820	5.09514	5.26082	
14	1.99988	2.98935	3.91783	4.70829	5.28498	5.58923	
15	1.99994	2.99297	3.94000	4.77576	5.43199	5.85119	5.99534

column $r = 0$ is omitted since $\mu_{k,0} = 1$ for all k. There are several interesting questions about the behaviour of $\mu_{k,r}$ that arise from examining the table. The most obvious feature of the table is that the figures in column r seem to tend to $r + 1$ from below: it is possible to show that this is in fact the case [5]. It is also possible to show that $\mu_k \sim \frac{k}{e}$ as $k \to \infty$ (the proof of this result [5] follows from the general theory of permanents and from a result of Hwang [18]). Thus we have an estimate for the maximum μ_k of each of the rows of the table. It would be interesting to show that $\mu_{k,r-1} < \mu_{k,r}$ for all $r \leq \frac{k-1}{2}$. This would, for example, imply that $m_k = 1$ if k is odd and $m_k = 2$ if k is even. However, this remains an open problem.

Estimates of the value of $b(n, s, t) := |B(n, s, t)|$ are closely bound up with the estimation of $|A(n, k)|$. The connection is provided by a bijection which exists between $B(n, s, t)$ and closed paths in $\Gamma_{s+t+1,t}$ of length n which start and end at a particular vertex. By making use of this bijection, we can show that $b(n, s, t)$ corresponds to a particular entry on the main diagonal of the matrix $H(s + t + 1, t)^n$. This allows us to calculate the values of $b(n, s, t)$ for small parameter sets and enables us to prove that there exist constants c_1 and c_2, depending only on s and t, such that

$$c_1(\mu_{s+t+1,t})^n \leq b(n, s, t) \leq c_2(\mu_{s+t+1,t})^n$$

for all sufficiently large n. This result shows that all the constants $\mu_{k,r}$ appear in the enumeration function of a natural class of permutations. Hence it would be interesting to find good estimates of the values of each individual $\mu_{k,r}$ (rather than just the values of μ_k).

There are a great many interesting problems which remain unsolved in this area. As well as improving the asymptotic and non-asymptotic estimates for the number of permutations in $A(n,k)$ and $B(n,s,t)$, there are other interesting classes of permutations waiting to be enumerated. For example, the classes of the permutations considered here were chosen so as to make some specific types of speech scrambler efficient. Considering security issues provides new classes of permutations to examine. The concept of security is difficult to define in our situation, but by experimenting with different permutations some criteria that a permutation should satisfy emerge. Beker and Mitchell [2, Section 7] say that the most important property a secure permutation π of $\{0,\dots,n-1\}$ should satisfy is that $i\pi + 1 \neq (i+1)\pi$. Enumerating permutations with this property which lie in $A(n,k)$ or $B(n,s,t)$ is an example of a class of permutations whose enumeration would be of interest to both combinatorialists and to the designers of speech scramblers of the type we have discussed.

4 Finite fields

This section concentrates on the use of the techniques of finite field theory in the analysis of a class of ciphers known as stream ciphers. This is just one of the many areas of cryptography where finite field theory plays a role (others include the analysis of block ciphers by approximating them by linear structures and the use of polynomial interpolation in certain secret sharing schemes).

A stream cipher is a machine which accepts a small secret random number (known as the key) and rapidly produces a long pseudorandom sequence of elements from some finite field (usually $\mathbb{F}_2$). One of the main applications of such ciphers is in the encryption of digitised speech. In this application, the digitised speech can be regarded as a binary sequence $M = m_0, m_1, \dots$ which is produced at a high rate. Suppose a stream cipher produces the binary sequence $S = s_0, s_1, \dots$ when fed a certain secret key. The message M is encrypted to form the sequence $C = c_0, c_1, \dots$ by defining $c_i = m_i + s_i$ modulo 2. The sequence C is transmitted and is decrypted by anyone with knowledge of the sequence S by setting $m_i = c_i + s_i$ modulo 2. For such a system to be secure, the sequence S must be difficult to predict by an evesdropper who has no knowledge of the secret key, but who may have information about a small segment of the sequence S.

4.1 Linear complexity

One of the most important criteria for a stream cipher is the linear complexity of its output sequence S. A sequence $s_0, \dots, s_{m-1}$ of elements from a field $\mathbb{F}$ has linear complexity l if there exist elements $a_0, a_1, \dots, a_l \in \mathbb{F}$, not all zero, such that

$$\sum_{j=0}^{l} a_j s_{i+j} = 0 \tag{4.1}$$

for all $i = 0, 1, \ldots, m-1-l$ and if no non-trivial relation of the form (4.1) exists for any smaller value of l. An infinite sequence $s_0, s_1, \ldots$ has linear complexity l if (4.1) holds for some $a_0, a_1, \ldots, a_l \in \mathbb{F}$, not all zero, and for all $i \in \{0, 1, \ldots\}$ and if no non-trivial equation of the form (4.1) holds for any smaller value of l. Every finite sequence has a linear complexity and one can show that (when $\mathbb{F}$ is finite) an infinite sequence has a linear complexity if and only if it is ultimately periodic.

The notion of linear complexity is important for three reasons. Firstly, if a sequence has a low linear complexity then knowing a small segment of the sequence, together with an equation of the form (4.1), allows the rest of the sequence to be easily predicted. Secondly, a basic component (the linear feedback shift register) of many modern stream ciphers often produces sequences of low linear complexity, so it is quite possible that a poorly designed stream cipher could produce sequences which have low linear complexity. Lastly, there exist efficient algorithms which, when given a sequence S, find a recurrence of the form (4.1). This allows any evesdropper to tell if a sequence S produced by a stream cipher could fall to linear complexity attacks, if they know a short segment of S. For the remainder of this section, we concentrate on some of the algorithms which find the linear complexity of a sequence.

The algorithms we are considering fall into two broad groups: algorithms which measure the linear complexity of infinite sequences whose period is known and algorithms which measure the linear complexity of a finite sequence.

4.2 Algorithms to measure linear complexity

Suppose that S is a sequence of elements from $\mathbb{F}$ of period t. The classical method of finding the linear complexity of S is to use the Discrete Fourier Transform (DFT) of S: the weight of the transform of S is equal to the linear complexity of S. The DFT requires a primitive tth root of unity in order to operate, so can only be used when t is coprime to the characteristic of $\mathbb{F}$. The DFT can, however, be generalised to apply to sequences of any period. For a discussion of the unmodified DFT, see [31]. The Games–Chan algorithm, which operates when the sequence S is binary and has period a power of 2, is given in [12]. The general case—using various approaches—can be found in [4,17,22].

Another group of algorithms can be used when the period of the sequence S is unknown. The Berlekamp–Massey algorithm [21] is given the elements $s_0, s_1, \ldots$ in turn and returns the linear complexity of the finite sequence $s_0, s_1, \ldots, s_{i-1}$ at each stage. It can be shown that the linear complexity of the infinite sequence S of period t is equal to that of the finite sequence $s_0, s_1, \ldots, s_{i-1}$ whenever $i \geq 2t$. So if an upper bound on the period of S is known, the Berlekamp–Massey algorithm can be used to calculate the linear complexity of S. The

Berlekamp–Massey algorithm was invented to solve the decoding problem for Reed–Solomon codes [20]. It is strongly related to other algorithms [9,19,34] which solve this decoding problem. In fact, both the Berlekamp–Massey algorithm and the other decoding algorithms solve special cases of a more general problem—an interpolation problem which is related to a problem in the theory of rational approximation of analytic functions [1]. See [6] for a discussion of the general problem and a description of an algorithm which solves it.

A final algorithm, which is of a quite different nature to the previous algorithms, is the Zero Square algorithm [33]. This algorithm, when fed the elements $s_0, s_1, \ldots$ in turn, outputs the linear complexity of the i sequences $S^j := s_j, s_{j+1}, \ldots, s_{i-1}$ where $j \in \{0, 1, \ldots, i-1\}$. The complexities of such sequences are of interest since the output sequence of a stream cipher should have a low linear complexity from any starting point in order to be secure. The algorithm manages to compute these quantities efficiently by avoiding having to calculate the elements $a_0, \ldots, a_l$ in equations of the form (4.1), unlike the Berlekamp–Massey algorithm.

There are several directions that future research in this area might take. One approach is to investigate methods of improving the efficiency of algorithms to calculate the linear complexity of sequences in various situations. Another is to consider other complexity measures of sequences. For example, the k error linear complexity of [32] is very interesting (this notion has also been considered in [10] and [11]), but an algorithm only exists to measure this quantity when S is a binary sequence whose period is a power of 2. The p-adic span [16] is another interesting measure: efficient algorithms seem to exist to calculate the p-adic span, but there is currently no close analogue of the Berlekamp–Massey algorithm[2]. There are plenty of interesting questions which remain unsolved.

References

1. Antoulas, A.C. and Anderson, B.D.Q. (1986). On the scalar rational interpolation problem. *IMA J. Math. Control and Inform.*, **3**, 61–88.

2. Beker, H. and Mitchell, C. (1987). Permutations with restricted displacement. *SIAM J. Alg. Disc. Meth.*, **8**, 338–363.

3. Beker, H. and Piper, F. (1985). *Secure speech communications*, Academic Press, London.

4. Blackburn, S.R. A generalisation of the Discrete Fourier Transform: determining the minimal polynomial of a periodic sequence. *IEEE Trans. Inform. Theory.* (To appear.)

5. Blackburn, S.R. Enumerating permutations with restricted displacement. (In preparation.)

[2]Note added in proofs: Andrew Klapper has recently announced that such an algorithm has now been found.

6. Blackburn, S.R. The generalised rational interpolation problem and the solution of the Welch–Berlekamp key equation. (In preparation.)

7. Blom, R. (1983). Non-public key distribution. *Advances in Cryptology: Proc. Crypto '82*, Plenum Press, New York, 231–236.

8. Blom, R. (1985). An optimal class of symmetric key generation systems. *Advances in Cryptology: Proc. Eurocrypt '94, Lecture Notes in Computer Science*, **209**, Springer-Verlag, Berlin, 335–338.

9. Chambers, W.G., Peile, R.E., Tsie, K.Y. and Zein, N. (1993). Algorithm for solving the Welch–Berlekamp key equation with a simplified proof. *Electronics Letters*, **29**, 1620–1621.

10. Ding, C., Xiao, G. and Shan, W. (1991). The stability theory of stream ciphers. *Lecture Notes in Computer Science*, **561**, Springer-Verlag, Berlin.

11. Fell, H. (1991). Linear complexity of transformed sequences. *Proc. Eurocode '90, Lecture Notes in Computer Science*, **514**, Springer-Verlag, Berlin, 205–214.

12. Games, R.A. and Hui Chan, A. (1983). A fast algorithm for determining the complexity of a binary sequence with period 2^n. *IEEE Trans. Inform. Theory*, **IT-29**, 144–146.

13. Gong, L. and Wheeler, D.J. (1990). A matrix key-distribution scheme. *J. Cryptology*, **2**, 51–59.

14. Hughes, D.R. and Piper, F.C. (1988). *Design theory*, Cambridge University Press.

15. Jansen, C.J.A. (1986). On the key storage requirements for secure terminals. *Computers and Security*, **5**, 145–149.

16. Goresky, M. and Klapper, A. (1993). Feedback registers based on ramified extensions of the 2-adic numbers. *Proc. Eurocrypt '94*, Perugia, Italy, 211–225.

17. Günther, C.G. (1994). A Finite Field Fourier Transform for vectors of arbitrary length. *Communications and Cryptography. Two Sides of One Tapestry*, Editors: R. Blahut, D. Costello, U. Maurer and T. Mittelholzer, Kluwer, Norwell MA, 141–153.

18. Hwang, S.-G. Matrix polytope and speech security systems. (Preprint.)

19. Liu, T.-H. (1984). A new decoding algorithm for Reed–Solomon codes. *Doctoral Thesis*, University of Southern California.

20. MacWilliams, F.J. and Sloane, N.J.A. (1988). *The Theory of Error Correcting Codes*, North Holland, New York.

21. Massey, J.L. (1969). Shift register synthesis and BCH decoding. *IEEE Trans. Inform. Theory*, **IT-15**, 122–127.

22. Mathys, P. (1990). A generalisation of the Discrete Fourier Transform in finite fields. *Proc. IEEE Sym. Information Theory*, San Diego (CA), 14–19.

23. Metropolis, N., Stein, M.L. and Stein, P.R. (1969). Permanents of cyclic (0,1) matrices. *J. Combin. Theory*, **7**, 291–321.

24. Minc, H. (1984). *Permanents*, Cambridge University Press.

25. Minc, H. (1988). *Nonnegative Matrices*, John Wiley and Sons, Chichester.

26. Mitchell, C. (1989). Speech security and permanents of (0,1) matrices. *Cryptography and Coding*, Editors: H. Beker and F. Piper, Oxford, 231–240.

27. Mitchell, C.J. and Piper, F.C. (1985). A classification of time element speech scramblers. *J. Inst. Electronic and Radio Engineers*, **55**, 391–396.

28. Mitchell, C.J. and Piper, F.C. (1987). The cost of reducing key storage requirements in secure networks. *Computers and Security*, **6**, 339–341.

29. Mitchell, C.J. and Piper, F.C. (1988). Key storage in secure networks. *Discrete Applied Mathematics*, **21**, 215–228.

30. Quinn, K.A.S. (1991). Combinatorial structures with applications to information theory. *Doctoral Thesis*, University of London.

31. Rueppel, R.A. (1992). Stream ciphers. *Contemporary Cryptology. The Science of Information Integrity*, Editor: G.J. Simmons, IEEE Press, New York, 65–134.

32. Stamp, M. and Martin, C.F. (1993). An algorithm for the k-error linear complexity of binary sequences with period 2^n. *IEEE Trans. Inform. Theory*, **IT-39**, 1398–1401.

33. Stephens, N.M. (1992). The zero square algorithm for computing linear complexity profiles. *Coding and Cryptography II*, Editor: C. Mitchell, Clarendon Press, Oxford.

34. Welch, L. and Berlekamp, E.R. (1983). Error correction for algebraic block codes. *U.S. Patent 4 633 470.*

The Regular Coloration of Graphs

M.G. Everett* and S.P. Borgatti**

* *University of Greenwich, London, and* ** *University of South Carolina, USA*

Abstract

A regular coloration of a graph is an assignment of colours to the vertices which obeys the rule that if two vertices are coloured the same then their neighborhoods have the same set of colours. If the graph represents a social system then vertices which are coloured the same can be thought of as playing the same role. We investigate the concept of regular coloration and present some results which allow for the development of algorithms which can be used to analyze social network data.

1 Introduction

Graph theory has been extensively used as a model in the social sciences. The vertices of a graph represent individuals or groups of individuals. The groups can be as diverse as political parties, countries or family groups. The edges represent interpersonal or intergroup relations, for example "likes", "hates", "agrees", "communicates with", "married to", "allied with", "trades with" etc. Clearly this model can give rise to both digraphs and graphs. The area of social sciences is known as social networks and graph theory has been a principal tool in the development of both theory and techniques for data analysis. In this paper we shall examine a graph theoretic formulation of the concept of social role.

Ideas of social role have been important to social theorists since the middle of the century. Early concepts were limited to roles which had been identified by common language; these include kinship roles such as "mother", work roles such as "the manager of", occupational roles such as "teacher" and more general roles such as "leader". Social networks provide an ideal environment in which to formalize a more abstract concept of social role. Individuals are identified as playing the same role if they relate in the same way to individuals playing counterpart roles. For example the role of middle manager in an organisational structure can be determined by examining their relationships to higher management and workers.

Let $G(V, E)$ be a finite graph with vertex set V and edge set E. Self-loops and multiple edges are allowed. We shall refer to directed graphs as digraphs and use the notation $D(V, E)$.

In a directed graph we define the in-*neighbourhood* of a vertex v as the set of vertices from which v receives connections, and the out-*neighbourhood* as the set of vertices which receive connections from v; these are denoted by $N_i(v)$ and $N_O(v)$ respectively so that

$$N_i(v) = \{x : (x, v) \in E\}$$
$$N_O(v) = \{x : (v, x) \in E\}.$$

In the undirected case these two sets would be the same and we therefore use the general term ***neighbourhood of*** v, which will be denoted by $N(v)$. A *coloration* C of a digraph D is an assignment of colours to the vertices of D. If S is a subset of the vertices of a digraph then the *spectrum* of S, denoted by $C(S)$, is the set of all colours assigned to the vertices of S. If S consists of just a single vertex v, then we shall write $C(v)$ and call this the *colour* of v.

2 Regular coloration

Definition 1. *A coloration C of a digraph $D(V, E)$ is regular if and only if for all $u, v \in V$.*

$$C(u) = C(v) \Rightarrow C(N_i(u)) = C(N_i(v))$$

and

$$C(N_O(u)) = C(N_O(v)).$$

In the undirected case this reduces to the condition that

$$C(u) = C(v) \Rightarrow C(N(u)) = C(N(v)).$$

We shall now assume that all graphs and digraphs do not contain isolates (i.e. vertices with no adjacencies to other vertices). This restriction is of little consequence to the theory of regular coloration but it does simplify the statements of a number of the results.

In any graph or digraph, the coloration which assigns a different colour to every vertex is always regular. In a graph (but not necessarily a digraph) the coloration in which every vertex is coloured the same is also regular. It is clear that for any graph or digraph there may be a number of possible regular colorations. Since every regular coloration defines a partition (two vertices, being in the same colour class if they are coloured the same) then we can order the set of regular colorations by using the refinement relation on the induced partition. Let $\mathbb{R}$ be the set of all regular colorations of a digraph D. Let $C_1, C_2 \in \mathbb{R}$ and suppose $C_1 \leq C_2$, where $\leq$ is the order defined above, then for any pair of vertices u and v in D such that $C_1(u) = C_1(v)$ it follows that

$$C_2(u) = C_2(v).$$

The idea of regular coloration was first formulated algebraically by White and Reitz [1], the regular coloration formulation was by Everett and Borgatti [2], in which the following theorem was first proved for graphs.

Theorem 2. *For any digraph $D(V, E)$ then the set of all regular colorations, denoted by $\mathbb{R}(D)$, partially ordered by $\leq$ forms a lattice.*

Proof. We shall prove the existence of arbitrary joins; we note the trivial regular coloration provide us with zero so we need only consider non-empty subsets of $\mathbb{R}(D)$. Let $R \in \mathbb{R}(D)$, then each regular coloration induces an equivalence relation $\equiv_R$ on V. If I is a non-empty subset of $\mathbb{R}(D)$, then we can identify a family of induced equivalence relations $\equiv_i$, for each $i \in I$. Define a new relation $\equiv$ on V by $v_k \equiv v_m$ if there exists a sequence $z_0, z_1, \ldots, z_n$ with $v_k = z_0, v_m = z_n$, such that for all j in the range $1 \leq j \leq n$, $\exists i(j) \in i$ such that $z_{j-1} \equiv_{i(j)} z_j$. In other words, in our new coloration, two vertices, v_k and v_m, are in the same colour class if we can find a sequence of vertices beginning with v_k and ending with v_m such that every successive pair in the sequence is in the same colour class for some regular coloration. We shall show that $\equiv$ induces a regular coloration on D which is equal to $\vee I$. The construction of $\equiv$ is the same as that used in the construction of the join for the lattice of equivalence relations; consequently, it is well known that $\equiv$ is an equivalence relation and a supremum with respect to the refinement ordering. We need only show that it induces a regular coloration.

Suppose $v_k \equiv v_j$, with corresponding sequence $z_0, z_1, \ldots, z_n$, and further suppose that $x \in N_0(v_k)$, hence $C(x) \in C(N_0(v_k))$. Now $z_0 \equiv_{i(1)} z_1$ and since $z_0 = v_k$, then $x \in N_0(z_0)$. It follows that $C_{i(1)}(x) \in C_{i(1)}(N_0(z_1))$ and therefore $\exists d_1 \in N_0(z_1)$ with $C_{i(1)}(x) = C_{i(1)}(d_i)$ and hence $x \equiv_{i(1)} d_1$. Similarly since $z_1 \equiv_{i(2)} z_2$ and $d_1 \in N_0(z_1)$ we can repeat the above argument and find a d_2 such that $d_2 \equiv_{i(2)} d_1$. Continuing inductively we construct a sequence $x, d_1, d_2 \ldots, d_n$, where $d_n \in N_0(z_n) = N_0(v_j)$ with each pair of the sequence in the same colour class for some member of I. It therefore follows that $x \equiv d_n$ so that $C(x) \in C(N_0(v_j))$ and hence $C(N_0(v_k)) \subset C(N_0(v_j))$. Similarly, $C(N_0(v_j)) \subset C(N_0(v_k))$, $C(N_i(v_k)) \subset C(N_i(v_j))$, $C(N_i(v_j)) \subset C(N_i(v_k))$ and the result follows.

It is not the case that $\mathbb{R}(D)$ is simply a sublattice of $\xi(V)$, the lattice of all colorations (i.e. all equivalence relations) on V. Whilst the joins are the same the meets are not. Consider the following two colorations of $2P_3$. For the first coloration colour each vertex of one component P_3 a different colour and repeat the coloration for the second component. For the second coloration, simply interchange the colours of the two endpoints of one of the components of the graph. The equivalence relation meet of these two colorations is not a regular coloration.

Definition 3. *A regular coloration which uses k colours is called a k-regular coloration.*

Clearly a bipartite graph has a 2-regular coloration. As previously stated any graph has a 1-regular coloration. The following theorem gives conditions under which a digraph has a 1-regular coloration. (Borgatti and Everett [3]).

Theorem 4. *A digraph $D(V, E)$ has a 1-regular coloration if and only if it contains no sources or sinks.*

Proof. Suppose D contains no sources or sinks then every vertex of D has non-empty in-neighbourhoods and out-neighbourhoods. Hence if every vertex is the same colour the coloration will be regular.

Conversely, suppose that assigning every vertex the same colour produces a regular coloration. If D contains a source then by definition the in-neighbourhood of this vertex will be empty. Since the digraph must contain a vertex which is not a source then this vertex must have a non-empty in-neighbourhood. Both neighbourhoods cannot be coloured the same which contradicts the regularity of the coloration.

Definition 5. *The image digraph $D'(C(V), E')$ of a coloration C of a digraph $D(V, E)$ has $C(V)$ as its vertices. There is an edge from vertex A to B if there exists an edge in D from a vertex coloured A to a vertex coloured B.*

Definition 6. *Let C be a coloration of a digraph $D(V, E)$. Given a colour $K \in C$ then the K outdegree of a vertex $v \in V$, denoted by ${}^K\rho_0(v)$ is the number of edges initiating from v and terminating at a vertex coloured K. We define K indegree, denoted by ${}^K\rho_i(v)$ similarly. The coloration C is a divisor coloration if, and only if, for all $u, v \in V$ and every $K \in C(V)$.*

$$C(u) = C(v) \Rightarrow^K \rho_0(u) =^K \rho_0(v)$$

and

$$^K\rho_i(u) =^K \rho_i(v)$$

Clearly any divisor coloration is a regular coloration. The term divisor coloration is a consequence of a result of Sachs [4]. He showed that for a divisor coloration if we place the value of ${}^K\rho_0(u)$ on the edge $(C(u), K)$ in the image digraph then the characteristic polynomial of the image digraph is a factor of the characteristic polynomial of the original digraph. One important example of a divisor coloration is colouring each orbit of any subgroup of the automorphism group of a digraph a different colour. A proof of this, together with further details on divisor coloration is contained in Cvetkovic, Doob and Sachs [5]. A consequence of this result is the following theorem.

Theorem 7. *Colouring each orbit of any subgroup of the automorphism group of a digraph a different colour is a regular coloration.*

The next theorem gives us a useful characterisation which is exploited in some of the proofs of other theorems in the paper.

Theorem 8. *Let $D(V,E)$ be a digraph with coloration C, then C is a regular coloration if and only if for every $v \in V$, $C(N_0(v)) = N_0(C(v))$ and $C(N_i(v)) = N_i(C(v))$. Note the two right hand sides refer to the neighbourhoods of the image digraph under C.*

Proof. The construction of the image means that for all $v \in V$, $C(N_0(v)) \subset N_0(C(v))$. Suppose that C is a regular coloration and $\exists v$ such that $N_0(C(v)) \not\subset C(N_0(v))$. That is, $\exists$ a colour A such that $A \in N_0(C(v))$ but $A \notin C(N_0(v))$. Since $A \in N_0(V(v))$ then $\exists y \in V$ such that $C(y) = C(v)$ and $A \in C(N_0(y))$ contradicting the regularity of C.

Now suppose that for all $v \in V$, $C(N_0(v)) = N_0(C(v))$. If $C(x) = C(y)$ then $N_0(C(x)) = N_0(C(y))$ and hence $C(N_0(x)) = C(N_0(y))$. We can repeat the above argument by replacing N_0 by N_i and the result follows.

Any digraph $D(V,E)$ is simply a binary relation E on the vertex set V. We can use E to generate a semigroup S under the operation of relational composition. We call this the semigroup associated with D. The following theorem was first proved by White and Reitz [1].

Theorem 9. *Let D' be the image digraph of a regular coloration C of a digraph D. Let S and S' be the associated semigroups of D and D' respectively then S is a homomorphic image of S.*

Proof. Let E and E' be the relations induced by D and D'. Suppose $A(E^i)'B$ so that $\exists u, v \in V(D)$ such that uE^iv with $C(u) = A$ and $C(v) = B$. Since uE^iv then $\exists z_1, z_2 \ldots z_{i-1}$ with $uEz_1, z_1Ez_2, \ldots z_{i-1}Ev$. By the construction of the image we have $C(u)E'C(z_1)$, $C(z_1)E'C(z_2) \ldots C(z_{i-1})E'C(v)$ and hence $A(E')^iB$.

Conversely, if $A(E')^iB$ then $\exists X_1, X_2, \ldots X_{i-1}$ such that $AE'X_1, X_1E'X_2, \ldots X_{i-1}E'B$. Since $AE'X_1$ then $\exists u \in V(D)$ such that $C(u) = A$ and $X_1 \in N(C(u))$. It follows from Theorem 8 that $C(z_1) = X_1$ so that $\exists z_1 \in V$ such that $C(z_1) = X_1$ and uEz_1. We can continue in this way to form a sequence $u, z_1, z_2, \ldots, z_{i-1}, v \in V$ such that $uEz_1, z_1Ez_2, \ldots z_{i-1}Ev$ with $C(z_j) = X_j$ $j = 1 \ldots i-1$, $C(u) = A$ and $C(v) = B$ so that $A(E^i)B$. It follows that $(E^i)' = (E)^i$. Let $Q \in S$ then $Q = E^p$ and define $\phi(Q) = (E^p)'$, it easily follows that ϕ is a homomorphism.

Corollary. *If in addition to the theorem the coloration is such that for all $u, v \in V$ then $C(u)E'c(v)$ implies uEv then S and S' are isomorphic.*

The conditions of the corollary are much stronger than required and it would be useful to weaken them to some extent, but no reasonable weakening is known to the authors.

Definition 10. *If the lattice of all regular colorations of a graph or digraph is trivial (i.e. consists of 1 or 2 elements) then we call the graph or digraph role primitive.*

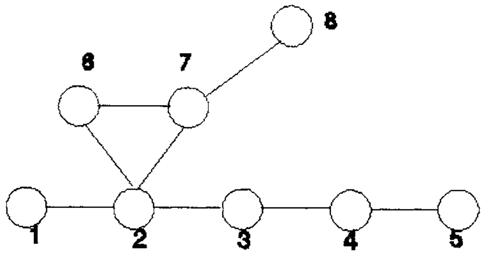


Figure 1. A role primitive graph

Clearly we are interested in graphs and digraphs with 3 or more vertices. It is easy to construct role primitive digraphs (with more than 3 vertices) but slightly more difficult for graphs. Let H be the 8 vertex graph as shown in Figure 1.

Theorem 11. *The graph H is role primitive.*

Proof. Suppose that $C(5) = A$. It follows that $C(4)$ must be a different colour, B say. Now if $C(3) = A$ then the only way we could complete the regular coloration is if the graph was bipartite. Alternatively if $C(3) = B$ then it would not be possible to colour 1 in such a way that the coloration was regular. We therefore assume $C(3) = C$. Clearly $C(2)$ cannot be coloured A, suppose $C(2) = B$. Since B's are only adjacent to A's and C's then $C(7)$ is A or C. However A's and C's are only adjacent to B's and it follows that 8 must be coloured B. This contradicts regular coloration since B's cannot be pendants so that $C(2) \neq B$. If $C(2) = C$ then 1 cannot be coloured in such a way as to produce a regular coloration so 2 must be coloured with a new colour D. If $C(2) = D$ then $C(1)$ must also be a different colour E. Since vertex 8 is a pendant it can only be coloured A, E or a new colour. If $C(8) = A$ then $C(7) = B$ and we cannot complete a regular coloration. Alternatively if $C(8) = E$ then $C(7) = D$ in which case $C(6)$ must be C but this is non-regular. It follows that $C(8) = F$ in which case $C(7)$ would be forced to be D but this cannot give a regular coloration. Hence 7 and therefore 6 both require new colours. We conclude that H is role primitive.

A computer search can be used to verify that there are no graphs with fewer than 8 vertices which are role primitive. The following theorem allows us to restrict our search to identity graphs.

Theorem 12. *Every role primitive graph with 3 or more vertices is an identity graph.*

Proof. Since the orbits of any subgraph of the automorphism group of the graph G can be used to form a regular coloration then $Aut(G)$ is either the identity or acts transitively. It follows that if $Aut(G)$ is transitive the stabilizers are trivial and so $Aut(G)$ acts regularly. Since no subgroup of a regular group can be transitive $Aut(G)$ must have prime order and so be Abelian. The only Abelian automorphion groups which can act regularly on the vertices of a graph are the elementary Abelian 2–groups. It follows that $Aut(G) = Z_2$ and Burnsides theorem then contradicts the fact that G has 3 or more vertices.

Note that the above theorem is false for digraphs, any directed cycle of prime length acts as a counter–example. The next result examines regular coloration in terms of standard graph operations.

Theorem 13. *Let D and E be digraphs.*

1. *If D has a k–regular coloration and E has an l–regular coloration then $D \cup E$ and $D + E$ have a $k + l$–regular coloration.*
2. *If $D \cup E$ has an m–regular coloration then $D \times E$ has an m^2 regular coloration.*
3. *If $D \cup E$ is coloured so that D, E and $D \cup E$ are m–regular colorations $D \times E$ has an ${}^1/_2 m(m + 1)$ regular coloration.*

Proof.

1. Simply colour each vertex of D and E as for the k and l regular colorations making sure that E is not coloured using any of the colours of D.
2. Let C be an m–regular coloration of $D \cup E$. Colour each vertex $u = (u_1, u_2)$ of $D \times E$ by $(C(u_1), C(u_2))$.

$$\begin{array}{rcl} C(N_i(u)) & = & \{(C(u_1), A) \ : \ A \in C(N_i(u_2))\} \\ & \cup & \{(B, C(u_2)) \ : \ B \in C(N_i(u_1))\} \\ & = & \{(C(u_1), A) \ : \ A \in N_i(C(u_2))\} \\ & \cup & \{(B, C(u_2)) \ : \ B \in N_i(C(u_1))\} \\ & = & N_i(C(u)) \end{array}$$

   The out–degree is similar.

3. Follows from noting that the proof in 2 remains valid if we replace the ordered pair $(C(u_1), C(u_2))$ by the un–ordered pair $\{C(u_1), C(u_2)\}$.

Definition 14. *A coloration of a digraph D is* **ecological** *if $C(N_i(u)) = C(N_i(v)$ and $C(N_O(u)) = C(N_O(v)) \Rightarrow C(u) = C(v)$.*

A coloration which is ecological and regular is called a perfect *coloration.*

Ecological colorations are precisely the converse condition of regular colorations. It can be shown that the class of all ecological colorations forms a lattice (simply use the meet as in the lattice of all equivalence relations). The following theorem is a useful characterization of perfect colorations.

Theorem 15. *A regular coloration of a digraph D is a perfect coloration if and only if the image digraph contains no pairs of vertices with identical in and out neighbourhoods.*

Proof. Suppose that D has a perfect coloration and that in the image graph the distinct vertices $C(u)$ and $C(v)$ have identical neighbourhoods. That is $N_i(C(u)) = N_i(C(v))$, but since the coloration is regular then $C(N_i(u)) = C(N_i(v))$; similarly $C(N_0(u)) = C(N_0(v))$. Since the coloration is also ecological it follows that $C(u) = C(v)$, a contradiction. Conversely suppose that D has a regular coloration and that the image graph does not contain vertices with the same neighbourhoods. If $C(N_i(u)) = C(N_i(v))$ then $N_i(C(u)) = N_i(C(v))$ and similarly if $C(N_0(u)) = C(N_0(v))$ then $N_0(C(u)) = N_0(C(v))$. It follows that the neighbourhoods are the same so that $C(u) = C(v)$.

The following corollaries can be derived from the above theorem.

Corollary. *The class of all perfect colorations forms a lattice under the refinement relation.*

Corollary. *The maximal element of the regular coloration lattice is perfect.*

3 Algorithmic considerations

To be of use in the social sciences it is necessary that a regular coloration can be computed for a given data set. Noise in network data offers means that it is desirable to have robust approximate regular coloration algorithms. In this section we examine two methods—one based upon combinatorial optimisation and one direct method for computing regular coloration.

Definition 16. *Let D be a digraph with coloration C and adjacency matrix A. Then the partitioning of A by which the rows and columns are partitioned into the colour classes is called* **blocking** *of A. The submatrices corresponding to a blocking are called* **blocks**.

Theorem 17. *A coloration of a digraph D is regular if and only if the blocks of the corresponding adjacency matrix are either made up of zeros or have a 1 in every row and every column.*

Proof. Simple application of the definition of regular coloration.

Given a coloration then Theorem 17 allows us to measure to what extent the coloration is regular. We simply compute the Hamming distance between each block and a block which satisfies the regular coloration condition. The sum of all these distances gives a cost function which can be used as the basis of a combinatorial optimization method. The software package UCINET (Borgatti et al. [6]) implements a Tabu Search routine, the authors have also experimented with a genetic algorithm which has proved more efficient.

The following n^3 algorithm finds the maximal element of the regular coloration lattice.

1. Colour all vertices the same.
2. For each colour pick a representative colour.
3. For each representative: check the regular coloration condition against all vertices in the same colour class. Re–colour all vertices which fail the condition with a new colour. If no re–colouring occurs stop: otherwise go to 2.

Of course, by virtue of Theorem 4 this algorithm is only of use on digraphs which contain sources or sinks.

We note however, that this algorithm and Theorem 2 can be applied, with a small modification to divisor colorations. Therefore, an application area in the Social Sciences has thrown some light onto an area of pure mathematics. There are, of course, a number of problems still open in this area. In particular the following are of interest to social networks.

- What is the meet operation in the lattice of regular colorations?
- What is a characterization of any k–regular colorations other than $k = 1$ or n. In particular can we characterize 2–regular colorations?
- Can we give conditions under which the semigroups of the original graph and the image graph are isomorphic?
- How common are role primitive graphs?
- Can we develop a theory of "nearly" regular coloration which involves polynomial time algorithms?

References

1. White, D.R. and Reitz, K.P. (1983). Graph and semigroup homomorphisms on networks of relations. *Social Networks*, **5**, 193–235.
2. Everett, M.G. and Borgatti, S.P. (1991). Role colouring a graph. *Math. Social Sci.*, **21**, 183–188.

3. Borgatti, S.P. and Everett, M.G. (1989). The class of all regular equivalences: algebraic structure and computation. *Social Networks*, **11**, 65–88.

4. Sachs, H. (1966). Uber teiler, faktoren und charakteristische polynome von graphen. *Teil I Wiss. Z.TH Ilmenau*, **12**, 7–12.

5. Cvetkovic, D.M., Doob, M. and Sachs, H. (1979). *Spectra of graphs*, Academic Press, New York.

6. Borgatti, S.P., Everett, M.G. and Freeman, L.C. (1992). *L.C. UCINET IV*, Analytic Technologies, Columbia.

The Use of Combinatorial Structures in Communication Signal Design

S.W. Golomb

Department of Electrical Engineering, Communication Sciences Institute, University of Southern California, USA

1 Favourable signal sets

For many kinds of communication channels, the *optimum detection* system is a **correlation detector**, which computes the cross-correlation between the actual (noisy) received signal and locally generated models of the various waveforms which might have been sent. The waveform with the largest correlation is the *maximum likelihood estimate* of what was sent. It is this fact which motivates the search for sets of signals (waveforms) which are as mutually uncorrelated as possible for use as the "codewords" in reliable communication systems, and for signals which are as uncorrelated as possible with all non-zero time-shifts of themselves for use in radar, sonar, and synchronisation applications.

Typically, modern communication systems use binary (or at most q-level) modulation on some kind of carrier signal to obtain the transmitted signal. The rows of a **Hadamard matrix** can be used to provide mutually uncorrelated (i.e. orthogonal) waveforms for use in communications (see Diagram 1). The Hadamard matrix can be modified in any of a number of ways to provide additional desired characteristics of the waveforms. (See Diagrams 2–4.)

HADAMARD MATRIX ⇒ ORTHOGONAL CODE

+	+	+	+
+	+	−	−
+	−	+	−
+	−	−	+

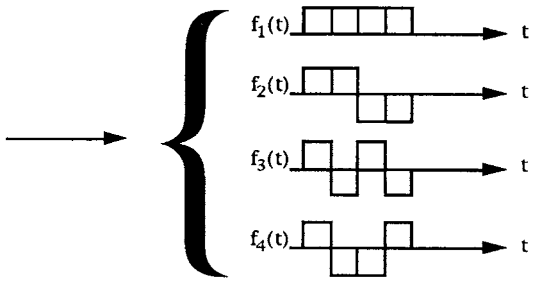


Diagram 1. The correspondence between the rows of a Hadamard matrix and a set of orthogonal signals

DOUBLED H-MATRIX ⇒ BI-ORTHOGONAL CODE

$$\begin{array}{|c|c|c|c|}\hline + & + & + & + \\ \hline + & + & - & - \\ \hline + & - & + & - \\ \hline + & - & - & + \\ \hline - & - & - & - \\ \hline - & - & + & + \\ \hline - & + & - & + \\ \hline - & + & + & - \\ \hline \end{array} \longrightarrow$$

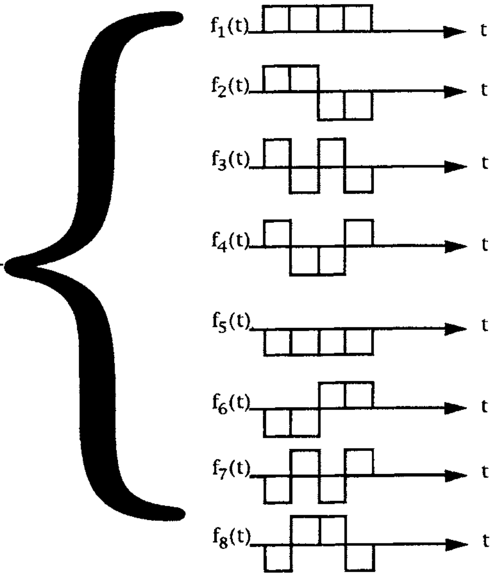


Diagram 2. A Hadamard matrix is $4t \times 4t$. (Do they exist for all $t \in Z^+$?) When $4t = 2^n$, the bi-orthogonal code is called a "Reed–Muller Code"

HADAMARD MATRIX ⇒ ORTHOGONAL CODE

$$\begin{array}{|c|c|c|c|}\hline + & + & + & + \\ \hline + & + & - & - \\ \hline + & - & + & - \\ \hline + & - & - & + \\ \hline \end{array} \longrightarrow$$

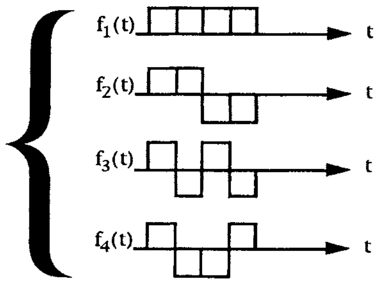


Diagram 3. (Normalised) correlation: $\int_0^1 f_i(t) f_j(t) dt = \delta_{ij}$

Reduced HADAMARD MATRIX ⇒ SIMPLEX CODE

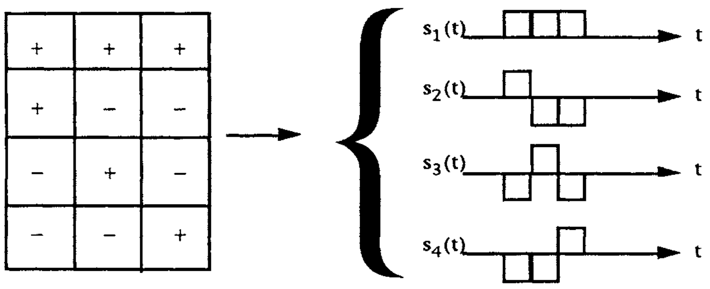


Diagram 4. (Normalised) correlation: $\int_0^1 s_i(t)s_j(t)dt = \left\{ \begin{array}{l} 1 \text{ if } i=j \\ -1/(n-1) \text{ if } i \neq j \end{array} \right\}$

2 The simplex bound

Theorem 1. *Given n "signals" (= functions) $h_i(t)$ on [0,1], with $(h_i \cdot h_j) = \int_0^1 h_i(t)h_j(t)dt$ such that $(h_i \cdot h_i) = 1$ for all i, $1 \le i \le n$, then*

$$\min_{\text{all ``codes''}} \max_{i \neq j}(h_i \cdot h_j) \ge -1/(n-1).$$

Proof.

$$\begin{aligned} \max_{i\neq j}(h_i \cdot h_j) &\ge \text{average}_{i \neq j}(h_i \cdot h_j) \\ &= \tfrac{1}{n(n-1)} \left\{ \textstyle\sum_{i=1}^n \sum_{j=1}^n (h_i \cdot h_j) - \sum_{i=1}^n (h_i \cdot h_i) \right\} \\ &= \tfrac{1}{n(n-1)} \left\{ ((\textstyle\sum_{i=1}^n h_i) \cdot (\sum_{j=1}^n h_j)) - n \right\} \\ &= \tfrac{1}{n(n-1)} \left\{ |\textstyle\sum_{i=1}^n h_i|^2 - n \right\} \ge \tfrac{1}{n(n-1)}(0-n) \\ &= \tfrac{-1}{n-1}. \end{aligned}$$

Corollary. If the signals $\{h_i(t)\}$ are "binary", i.e. restricted to the values +1 and −1, then

$$\min_{\text{all ``codes''}} |\sum_{i=1}^n h_i(t)|^2 \ge \left\{ \begin{array}{l} 0 \text{ if } n \text{ even} \\ 1 \text{ if } n \text{ odd} \end{array} \right\}.$$

Thus,

$$\min_{\text{all "codes"}} \max_{i \neq j}(h_i \cdot h_j) \leq \left\{ \begin{array}{l} -1/(n-1),\ n \text{ even} \\ -1/n,\ n \text{ odd} \end{array} \right\}.$$

Note. If $4t \times 4t$ Hadamard matrices exist for all $t \in Z^+$, then the simplex bound $-1/(n-1)$ can be attained with binary signals for all even $n \in Z^+$.

3 Signals for range radar and sonar

For applications to range radar and sonar, sequences are sought which have a uniformly low autocorrelation value for all non-zero time shifts of the sequence. The problem of finding binary sequences with two-level periodic autocorrelation is mathematically equivalent to finding cyclic (v, k, λ) designs. For an out-of-phase periodic auto-correlation value near zero, cyclic Hadamard designs $(v = 4t - 1,\ k = 2t - 1,\ \lambda = t - 1)$ are the appropriate choice.

In addition to periodic binary modulation on a continuous wave (CW) carrier, a number of other radar and sonar systems are also in use. Instead of binary modulation, q-phase modulation (corresponding to sequences of complex numbers from the unit circle) may be used, for which the sequences of Frank and of Chu have specially favourable properties in the periodic case; and in the non-periodic [so-called *finite*] correlation case, Barker sequences (using *binary* modulation), generalised Barker sequences (using *q-phase* modulation), the aperiodic versions of Frank's and Chu's sequences, and Huffman's "impulse-equivalent pulse trains" (corresponding to *amplitude modulation*, i.e. the use of sequences of real numbers to generate the waveform) have all been studied. For *pulse radar* (as contrasted with CW radar), the optimum pulse patterns correspond to combinatorial objects called *minimum spanning rulers.*

4 Cyclic Hadamard difference sets

A *cyclic Hadamard difference* set is a cyclic (v, k, λ) difference set where $v = 4t - 1$, $k = 2t - 1$, and $\lambda = t - 1$. These correspond to "cyclic Hadamard matrices", where the rows are cyclic shifts of one another after the top row and left column of the matrix have been removed. For example:

$$H_4 = \begin{bmatrix} +1 & +1 & +1 & +1 \\ +1 & +1 & -1 & -1 \\ +1 & -1 & +1 & -1 \\ +1 & -1 & -1 & +1 \end{bmatrix}$$

$$H_8 = \begin{bmatrix} + & + & + & + & + & + & + & + \\ + & - & - & - & + & - & + & + \\ + & + & - & - & - & + & - & + \\ + & + & + & - & - & - & + & - \\ + & - & + & + & - & - & - & + \\ + & + & - & + & + & - & - & - \\ + & - & + & - & + & + & - & - \\ + & - & - & + & - & + & + & - \end{bmatrix}$$

and

$$H_{12} = \begin{bmatrix} + & + & + & + & + & + & + & + & + & + & + & + \\ + & - & + & - & + & + & + & - & - & - & + & - \\ + & - & - & + & - & + & + & + & - & - & - & + \\ + & + & - & - & + & - & + & + & + & - & - & - \\ + & - & + & - & - & + & - & + & + & + & - & - \\ + & - & - & + & - & - & + & - & + & + & + & - \\ + & - & - & - & + & - & - & + & - & + & + & + \\ + & + & - & - & - & + & - & - & + & - & + & + \\ + & + & + & - & - & - & + & - & - & + & - & + \\ + & + & + & + & - & - & - & + & - & - & + & - \\ + & - & + & + & + & - & - & - & + & - & - & + \\ + & + & - & + & + & + & - & - & - & + & - & - \end{bmatrix}$$

Examples of this type are called "cyclic Hadamard matrices", and are in one-to-one correspondence with "cyclic Hadamard difference sets".

All known examples of cyclic Hadamard difference sets have $v = 4t - 1$ of one of three types

1. $v = 4t - 1$ *prime.*
2. $v = 4t - 1 = u(u + 2)$, a *product of twin primes.*
3. $v = 2^k - 1$, *one less than a power of 2.*

We recently conducted a search for $v \leq 10,000$. Of the 2500 values of v on this range, all examples found had v of one of these three types, and only 17 other values of v remain to be completely checked.

The *general* constructions which give these designs are

1. For $v = 4t - 1$ *prime*, use the Legendre symbol $a_k = \left(\frac{k}{v}\right)$ for the sequence for $1 \leq k \leq v - 1$, but take $a_v = +1$ (instead of using $\left(\frac{0}{v}\right) = 0$). ["Legendre sequence" = "Quadratic residue sequence"].
2. For $v = 4t - 1 = u(u + 2)$ a product of twin primes, set $a_k = \left(\frac{k}{v}\right)$, the Jacobi symbol, when $(k, v) = 1$, but take $a_k = +1$ when $(k, v) = u$, and $a_k = -1$ when $(k, v) = u + 2$ and when $k = 0$. ["Twin-prime sequence" = "Stanton–Sprott sequence"].

3. For $v = 2^k - 1$, use the "m-sequence construction", i.e. use a "maximum length linear binary shift register sequence of degree k".

4.1 Other constructions for cyclic Hadamard sequences

1. When $v = 4t - 1$ is prime, if also $v = 4a^2 + 27$ (for example $v = 31, 43, 127, 223, 283, \ldots$) there is M. Hall, Jr.'s "sextic residue sequence" construction. However, these periods form a subset of those for which the Legendre sequence construction provides examples.
2. When $v = 2^k - 1$, with k composite, the Gordon–Mills–Welch (GMW) construction provides an alternative to the m-sequence construction.
3. Exhaustive searches have been conducted for all cyclic Hadamard difference sets when $v = 2^7 - 1 = 127$, $v = 2^8 - 1 = 255$, and $v = 2^9 - 1 = 511$. In each case, there were at least two "new" examples (three new examples at $v = 127$) not belonging to any of the previously described families. These may be examples of new families of cyclic Hadamard matrices which are still awaiting discovery.

5 The applications to range radar and sonar

To determine the range (i.e. the distance) to a target by radar or sonar, a signal is transmitted, reflected by the target, and detected at a receiver which is usually, though not always, located at or near the transmitter. The range is then proportional to the round-trip time of the signal.

One strategy is to send a very brief pulse with very high intensity (amplitude), and measure the elapsed time until an echo is detected. Quite often, however, the transmitter has a peak power limitation which makes it impossible to send a single pulse signal of sufficient intensity that its echo can be detected with high confidence. Broadening the pulse may allow more total energy to be transmitted, but at the expense of sharpness in the measurement of the round-trip time, and hence of the range being determined. (See Figure 1.)

A strategy to get more energy into the signal, and yet to retain the ability to make sharp range determinations, is to use **coded waveforms**. The type of coded waveform depends on the kind of signal the radar (or sonar) is able to generate.

A **pulse radar** (or sonar) generates pulses of energy of fixed amplitude and duration. However, the spaces between the pulses may be varied in some coded fashion. We will return to the coding problem for these pulse patterns later, under the heading of a family of combinatorial designs called **spanning rulers**.

A "continuous wave" (CW) radar or sonar transmits a sine wave signal $f(t) = A \sin(\omega t + \phi)$ which may be modulated in amplitude A, frequency ω, or phase ϕ. Quite commonly, phase modulation is used. In the case of binary phase modulation, the phase angle ϕ changes at intervals specified by an appropriate coding scheme between the values 0° and 180°, which has the same effect as

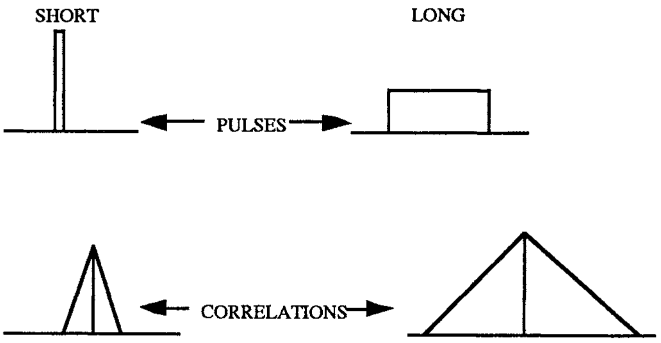


Figure 1. The shorter the pulse, the sharper the autocorrelation function

binary amplitude modulation, with the amplitude changing abruptly between $+A$ and $-A$ at the same intervals. We will see that the optimum coding scheme for *bi*-phase modulated CW radar (or sonar) corresponds to using the successive terms of a **cyclic Hadamard sequence**, of the type already described.

Another type of radar (or sonar) modulation, called **frequency hopping**, involves transmitting a sequence of pulses at different frequencies, where the sequence follows some type of coded pattern. A range/doppler radar (or sonar) is one which is used to determine both the ***distance*** (**range**) and the velocity relative to the observer (**doppler shift**) of the target. Frequency hopping patterns based on a family of combinatorial objects called **Costas Arrays** provide an optimum coding scheme for the simultaneous determination of distance and velocity.

It is possible to design a range radar (or sonar) system based on a uniformly spaced sequence of amplitude-modulated pulses. This idea, under the name of "impulse-equivalent pulse trains", was cleverly explored by David A. Huffman, but it has not led to practical systems.

Phase modulation with more than two phases is frequently used in radar signals. There is also a significant distinction between CW radars that "stay on the air" by repeating their patterns periodically, and those that "turn off" after completing their signal pattern.

As early as 1953, R.H. Barker explored the question of ± 1 modulation on a finite radar signal, and found excellent binary "Barker sequences" of lengths 2, 3, 4, 5, 7, 11 and 13. Several years later (1961) Turyn and Storer proved that no *odd*-length Barker sequences exist with length $L > 13$, and it is generally believed that no *even*-length Barker sequences exist with length $L > 4$. This is an unsolved combinatorial problem which has attracted considerable attention. It is equivalent to finding a "totally cyclic" Hadamard matrix larger than this 4×4 example

$$\begin{array}{cccc} + & + & + & - \\ + & + & - & + \\ + & - & + & + \\ - & + & + & + \end{array}$$

Barker's constraint on his sequences $\{a_1, a_2, \ldots, a_L\}$ is that the "unnormalized correlation" $C(\tau) = \sum_{i=1}^{L-\tau} a_i a_{i+\tau}$ satisfy $|C(\tau)| \leq 1$, for $\tau = \pm 1, \pm 2, \ldots, \pm(L-1)$. In 1965, Golomb and Scholtz introduced the notion of "generalised Barker sequences", where now the terms of the sequence $\{a_1, a_2, \ldots, a_L\}$ are numbers on the unit circle in the complex plane, with $C(\tau) = \sum_{i=1}^{L-\tau} a_i a_{i+\tau}^*$ and $|C(\tau)| \leq 1$ for $\tau = \pm 1, \pm 2, \ldots, \pm(L-1)$. Examples of "generalised Barker sequences" have now been found for all lengths $L < 20$ (N. Zhang and S. Golomb), and for several additional values with $21 \leq L \leq 25$ (Bömer and Antweiler), but it seems very unlikely that examples of arbitrarily great length exist.

Much longer "polyphase" signals, with weaker correlation constraints, have been introduced and studied for both the periodic case (Frank and Chu), and for the "finite" case (Frank, Zhang and Golomb et al.)

What all of these radar and sonar signals have in common is a highly favourable "auto-correlation function" $C(\tau) = \sum_i a_i a_{i+\tau}^*$, where, for $\tau = 0$, $C(0) = \sum_i |a_i|^2$ = the total energy of the signal, while for all $\tau \neq 0, |C(\tau)|$ is small compared to the length, or period, of the sequence.

The brute force approach is to undertake ever more ambitious and time-consuming attacks on these problems by intensive computer search. It is unlikely that we will ever abandon this method of finding good specific examples of ever greater lengths (or periods). However, there is one general approach to the study of the autocorrelation of sequences which applies to all of these specific cases, and yields valuable information about each of them.

6 The polynomial model

With the sequence $\{a_0, a_1, a_2, \ldots, a_{n-1}\}$ of length n, we associate the degree-$(n-1)$ polynomial $f(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$. We then observe that

$$f(x) f^*\left(\frac{1}{x}\right) = \sum_{\tau=-(n-1)}^{n-1} C(\tau) x^\tau$$

where as before,

$$C(\tau) = \sum_{i=0}^{n-1-\tau} a_i a_{i+\tau}^* \text{ if } \tau \geq 0, \text{ and } C(-\tau) = C^*(\tau).$$

Different specialisations of the $f(x) f^*\left(\frac{1}{x}\right)$ identity above have led to important results in several of the applications areas we have discussed. For the case of cyclic difference sets modulo v, this identity is the first step in the demonstration of M. Hall, Jr.'s "multiplier theorem" for difference sets. For pulse radar sequences, and the spanning ruler formulation, this identity is a crucial step in the study of "homometric rulers". Huffman used this identity in his work on "impulse-equivalent pulse trains", and it was used by Golomb and Scholtz to help identify the group of "Barker preserving transformations" on a sequence.

Recapitulation

1. Barker sequences.
2. Generalised Barker sequences.
3. Frank sequences.
4. Chu/Golomb–Zhang sequences.
5. Huffman's "Impulse-equivalent pulse trains".

Hadamard matrices (constructions)

1. Orthogonal codes.
2. Bi-orthogonal codes.
3. Reed–Muller codes.
4. Simplex codes.

Spanning rulers (applications)

1. Pulse radar patterns.
2. Antenna positioning for radio astronomy.
3. X-ray diffraction crystallography.
4. Tap combinations for convolutional codes.

2-level periodic correlation (constructions)

1. Shift register (PN, or m-) sequences.
2. Quadratic residue (Legendre) sequences.
3. Twin prime (Jacobi-symbol) sequences.
4. Cyclic Hadamard design sequences.

5. Cyclic (v, k, λ)-design sequences.

$$\left.\begin{array}{l}\text{Costas arrays}\\ \text{Tuscan squares}\end{array}\right\}\text{Frequency-hopping patterns.}$$

Some general methods

1. Lower bounds on mutual correlation
   (a) Simplex bound.
   (b) Bounds of Sidel'nikov, Welch, Levenshtain, etc.
2. The polynomial model (degree $L = n - 1$)

$$\{a_k\}_{k=0}^{L} \to \sum_{k=0}^{L} a_k x^k = f(x).$$

Then $F(x) = f(x)f^*\left(\frac{1}{x}\right) = \sum_{\tau=-L}^{L} C(\tau)x^\tau$, where $C(\tau) = \sum_{k=0}^{L-\tau} a_k^* a_{k-\tau}$.

3. Applications
   (a) $|C(\tau)|$ is unchanged by $a_k \to \rho\eta^k a_k$ and by $a_k = \rho\eta^k a_{L-k}$ for any complex numbers ρ and η with $|\rho| = |\eta| = 1$.
   (b) Information is gained by setting
      i. $f(x) = 0$,
      ii. $F(x) = 0$,
      iii. $x = e^{2\pi i/m} = \alpha$, or
      iv. $x = \alpha \in GF(q)$ with $\alpha^m = 1$.
   (c) (v, k, λ)-[cyclic]-difference sets, including [cyclic] Hadamard matrices.
   (d) Barker gave examples having the following lengths

L	Sequence
1	$+1$
2	$+1, +1$
3	$+1, +1, -1$
4	$+1, +1, -1, +1$
5	$+1, +1, +1, -1, +1$
7	$+1, +1, +1, -1, -1, +1, -1$
11	$+1, +1, +1, -1, -1, -1, +1, -1, -1, +1, -1$
13	$+1, +1, +1, +1, +1, -1, -1, +1, +1, -1, +1, -1, +1$

Turyn and Storer showed that there are no other "Barker Sequences" for *odd* lengths $L > 13$. It is still unproven that even length Barker Sequences with $L > 4$ do not exist, though this is generally believed.

7 Examples of generalised Barker sequences*

Length	# (Phases)	Sequences	Correlation $K(\tau)$ for $\tau =$								
L	ϕ	$\{a_1, a_2, \ldots, a_L\}$	0	1	2	3	4	5	6	7	8
2	2B.,C.	1,-1	2,	-1							
3	2B.	1,1,-1	3,	0,	1						
4	2B.,F.	1,1,1,-1	4,	1,	0,	-1					
5	2B.	1,1,1,-1,1	5,	0,	1,	0,	1				
6	6C.	1,ϵ,-1,1,-ϵ,-1	6,	-1,	-1,	1,	-1,	-1			
7	2B.	1,1,1,-1,-1,1-1	7,	0,	-1,	0,	-1,	0,	-1		
9	3F.	1,1,1,1,ω,$\overline{\omega}$,1, $\overline{\omega}$,ω	9,	ϵ,	ϵ^2,	0,	ϵ^5,	ϵ^4,	0,	-1,	ϵ^2

- $*|K(\tau)| \leq 1$ for $|\tau| \geq 1$. $\epsilon = e^{2\pi i/6}, \omega = \epsilon^2, \overline{\omega} = \omega^2 = \epsilon^4$.
- "B." *BARKER SEQUENCE.* $\phi = 2$.
- "F." *FRANK SEQUENCE.* $L = \phi^2$.
- "C." *CHU SEQUENCE.* $\phi = L$. (*Not* generalised Barker for $\phi = L > 9$.)

Frank sequences

Let $\alpha = e^{2\pi i/n}$. {Sequence length $L = n^2$}.

$$\left\{ \underbrace{\alpha^0, \alpha^0, \ldots, \alpha^0}_{n \text{ terms}}, \underbrace{\alpha^0, \alpha^1, \alpha^2, \ldots, \alpha^{n-1}}_{n \text{ terms}}, \right.$$

$$\left. \underbrace{\alpha^0, \alpha^2, \alpha^4, \ldots, \alpha^{2(n-1)}}_{n \text{ terms}}, \cdots, \underbrace{\alpha^0, \alpha^{n-1}, \alpha^{2(n-1)}, \ldots, \alpha^{(n-1)^2}}_{n \text{ terms}} \right\}$$

Chu sequences

Let $\alpha = e^{2\pi i/L}$. Sequence of length L is: $\left\{\alpha^0, \alpha^1, \alpha^3, \alpha^6, \ldots, \alpha^{\binom{L}{2}}\right\} = \left\{\alpha^{\binom{1}{2}}, \alpha^{\binom{2}{2}}, \alpha^{\binom{3}{2}}, \ldots, \alpha^{\binom{L}{2}}\right\}$.

For both Frank and Chu sequences, the **periodic** correlation $C(\tau) = 0$ for all $\tau \not\equiv 0 \pmod{L}$.

For both Frank and Chu sequences, the **aperiodic** (= **finite**) *unnormalised* correlation $K(\tau)$ satisfies

$$\max_{|\tau|\geq 1} |K(\tau)| < c \cdot L^{1/2},$$

so that the *normalised* auto-correlation $\frac{K(\tau)}{L}$ satisfies

$$\max_{|\tau|\geq 1} \left|\frac{K(\tau)}{L}\right| = O(L^{-1/2}) \text{ as } L \to \infty.$$

Bounds proved by

	Periodic Correlation	Finite Correlation
Frank sequences	Frank	Turyn
Chu sequences	Chu	Golomb & Zhang

8 Pulse patterns and "optimal rulers"

A pulse radar can send one or more pulses of RF energy toward a target. It is convenient and realistic to assume that all pulses have the same amplitude and duration. The signal design problem is to devise *patterns* of these identical pulses with an autocorrelation function as impulse-like as possible.

Equivalent problem: for each positive integer n, what is the shortest length $L = L(n)$ for which there is a sequence $\{a_1, a_2, \dots, a_n\}$ with $0 = a_1 < a_2 < \cdots < a_n = L$, such that the $\binom{n}{2}$ differences $\{a_j - a_i\}$ with $1 \le i < j \le n$ are all *distinct*?

We envision brief *pulses* at each time position $0 = a_1, a_2, a_3, \dots, a_n = L$. The distinctness of all $a_j - a_i$ $(i < j)$ guarantees that for each τ, the unnormalised correlation at $\tau \neq 0$ is at most 1.

The sequence model is also described in terms of a certain class of *rulers* (the measuring devices, not the monarchs or autocrats).

A ruler of length L has only n marks on it, at integer positions $a_1, a_2, \dots,$ a_n, where $a_1 = 0$ and $a_n = L$ are the two endpoints of the ruler. If every integer distance d, $1 \le d \le L$, can be measured in one and only one way

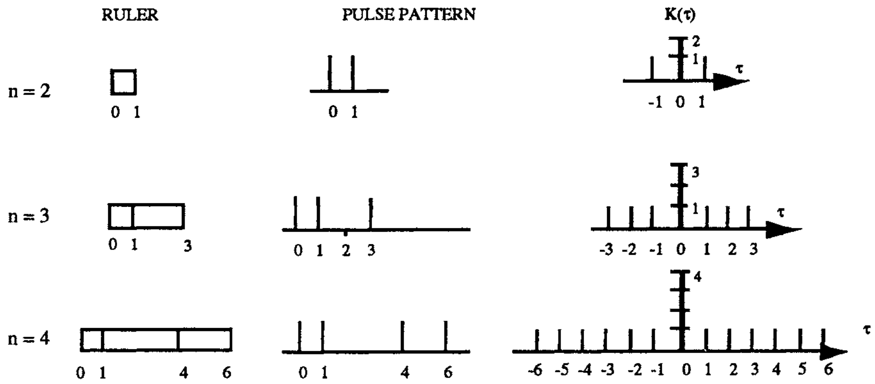


Figure 2. Perfect rulers for $n = 2$, $n = 3$ and $n = 4$, with the corresponding pulse radar patterns and their unnormalised autocorrelation functions

as a distance between two of the n marks, then the ruler is called a *perfect ruler*. For a perfect ruler, $L = \binom{n}{2}$, since there are exactly $\binom{n}{2}$ distances between the n marks, and these must be some permutation of $\{1, 2, 3, \ldots, L\}$. In Figure 2, we see perfect rulers for $n = 2, 3$, and 4, with the corresponding pulse radar patterns and their autocorrelation functions $K(\tau)$.

Unfortunately, for $n > 4$, there are no perfect rulers.

Theorem 2. *For $n > 4$, no perfect rulers exist.*

There are two obvious ways to relax the requirements on a perfect ruler to get objects which exist for all n. A *covering ruler* with n marks and length L measures every distance from 1 to L, as a distance between two marks on the ruler, in *at least one way*; while a *spanning ruler* with n marks and length L measures every distance from 1 to L, as a distance between two marks on the ruler, in *at most one way*. The interesting combinatorial problems are to determine the *longest covering ruler* with n marks, and the *shortest spanning ruler* with n marks, for each positive integer n. Both of these problems have long histories in the combinatorial literature. However, the application to pulse radar involves only finding the *shortest spanning ruler* for each n. (Martin Gardner termed these objects "Golomb rulers", a name which seems subsequently to have been widely adopted. There are also important papers which refer only to the "difference triangles", and not to the rulers themselves.)

The behavior of $L(n)$ as a function of n, for the shortest spanning ruler, is quite erratic in detail, although it is known that asymptotically $L(n) \sim n^2$ as $n \to \infty$. The value of $L(n)$ has been determined by exhaustive computer search for all $n \leq 19$. (For the values of n with $14 \leq n \leq 16$, this search was first performed by J. Shearer of IBM. The values of $L(17)$ and $L(18)$ were determined in 1993 by W. Olin Sibert of Lexington, MA, and the value of $L(19)$ was found in 1994 by a group at Duke University consisting of A. Dollas, T. Rankin, and D. McCracken.) In addition to left-right reversal of the ruler, these rulers are not unique for several of the smaller values of n. One example of a spanning ruler of length $L(n)$, for each $n \leq 16$, is shown in Table 1.

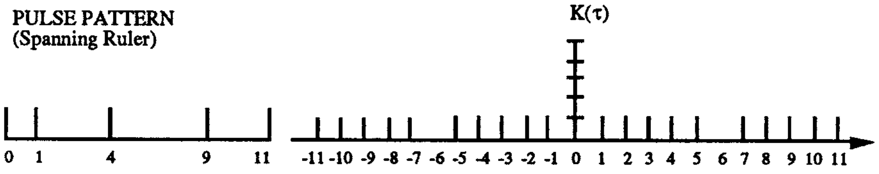


Figure 3. Radar pulse pattern, and its autocorrelation, for $n = 5$ pulses

Table 1. Table of the shortest spanning rulers

n	L(n)	m	Sequence of Marks
2	1	1	0,1
3	3	1	0,1,3
4	6	1	0,1,4,6
5	11	2	0,1,4,9,11
6	17	4	0,1,4,10,12,17
7	25	5	0,1,4,10,18,23,25
8	34	1	0,1,4,9,15,22,32,34
9	44	1	0,1,5,12,25,27,35,41,44
10	55	1	0,1,6,10,23,26,34,41,53,55
11	72	2	0,1,4,13,28,33,47,54,64,70,72
12	85	1	0,2,6,24,29,40,43,55,68,75,76,85
13	106	1	0,2,5,25,37,43,59,70,85,89,98,99,106
14	127	1	0,5,28,38,41,49,50,68,75,92,107,121,123,127
15	151	1	0,6,7,15,28,40,51,75,89,92,94,121,131,147,151
16	177	1	0,1,4,11,26,32,56,68,76,115,117,134,150,163,168,177

Note: The quantity "m" is the number of inequivalent rulers of length $L(n)$ which are shortest spanning rulers with n marks. Only one of each set of m rulers is listed explicitly in this table.

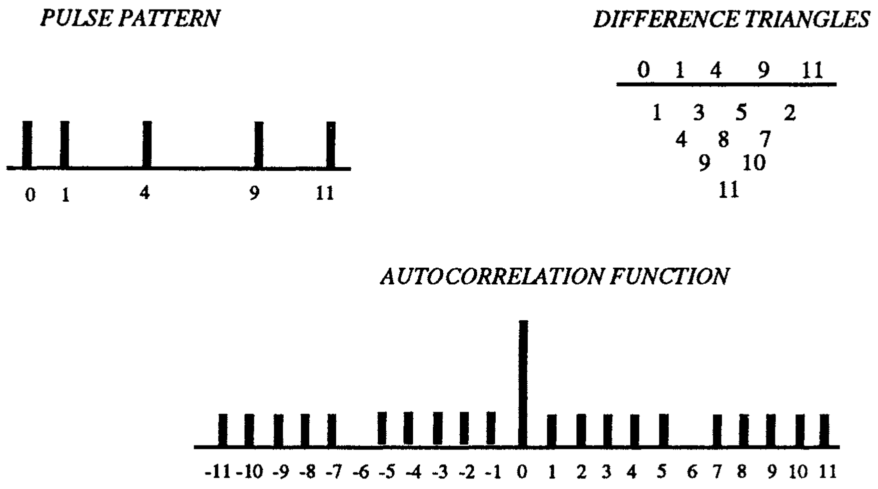


Diagram 5. The difference triangle corresponding to the five-mark spanning ruler in Figure 3, and auto-correlation

In Figure 3, we see the pulse pattern and correlation corresponding to the sequence listed for $n = 5$. The difference triangle, and the polynomial model for this example, are shown in Diagram 5.

Additional shortest spanning rulers

$n = 17$, $L(17) = 199$ *W. Olin Sibert* (1993)

$0, 8, 31, 34, 40, 61, 77, 99, 118, 119, 132, 143, 147, 182, 192, 194, 199$

$n = 18$, $L(18) = 216$ *W. Olin Sibert* (1993)

$0, 2, 10, 22, 53, 56, 82, 83, 89, 98, 130, 148, 153, 167, 188, 192, 205, 216$

$n = 19$, $L(19) = 246$ *A. Dollas, T. Rankin, D. McCracken* (1994)

$0, 1, 6, 25, 32, 72, 100, 108, 120, 130, 153, 169, 187, 190, 204, 231, 233, 242, 246$

with polynomial

$$f(x) = 1 + x + x^4 + x^9 + x^{11},$$

given by

$$f(x) \cdot f\left(\frac{1}{x}\right) = \sum_{\tau=-11}^{11} K(\tau)x^\tau.$$

9 Sophie Piccard's "Theorem"

Homometric rulers are rulers which measure the same set of distances.

In our terminology, S. Piccard's "Theorem" (1939) asserts

"If two *spanning rulers* of length L, with n marks, are *homometric*, they are either identical, or are mirror images of each other."

The counter-example shown in Diagram 6, with $n = 6$ and $L = 17$, was found (c. 1974) by Gary Bloom. [These were two of the four *minimum spanning rulers* ($n = 6$) from a table by John Leech, c. 1952.]

We found two infinite 2-parameter families of counter-examples for the case of rulers with $n = 6$ marks.

We now believe that there are *no* counter-examples with $n > 6$ marks. (We *proved* there are none for $n < 6$.)

G. Yovanof proved that if a counter-example is of the form $f_1(x) = \phi(x)g_1(x)$, $f_2(x) = \phi(x)g_2(x), f_1(x)f_1\left(\frac{1}{x}\right) = f_2(x)f_2\left(\frac{1}{x}\right)$ (as it *must* be) *and* $\phi(x), g_1(x)$ have at most one negative term between them, *then* $n = 6$.

HOMOMETRIC RULERS

```
 0  1  4  10  12  17                            0  1  8  11  13  17
 ___________________                            ___________________
  1  3  6   2   5                                 1  7  3   2   4
    4  9  8   7                                     8  10  5   6
     10 11 13          EACH SKIPS                    11  12  9
       12  16           14  &  15                      13  16
         17                                              17
```

Diagram 6. Non-identical homometric rulers, whose polynomial must be related as shown in this example

$$f_1(x) = 1 + x + x^4 + x^{10} + x^{12} + x^{17},$$

$$f_2(x) = 1 + x + x^8 + x^{11} + x^{13} + x^{17}.$$

$$f_1(x) = \phi(x)g_1(x), \quad f_2(x) = \phi(x)g_2(x)$$

where

$$g_2(x) = x^d g_1\left(\frac{1}{x}\right)$$

$$f_1(x) = (1 + x + x^6)(1 + x^4 - x^5 + x^{11});$$

$$f_2(x) = (1 + x + x^6)(1 - x^6 + x^7 + x^{11}).$$

9.1 Two-dimensional pulse patterns

J.P. Costas (see [21]) proposed the following problem: we wish to design an $n \times n$ frequency hop pattern, for radar or sonar, using n consecutive time intervals $t_1, t_2, \ldots t_n$, and n consecutive frequencies $f_1, f_2, \ldots, f_n$, where some permutation of the n frequencies is assigned to the n consecutive time slots. Moreover, this should be done in such a way that, if two frequencies f_i and f_{i+r} occur at the two times t_j and t_{j+s}, then there is no $i', i' \neq i$, where the two frequencies $f_{i'}$ and $f_{i'+r}$ occur at times $t_{j'}$ and $t_{j'+s}$. This constraint corresponds to an ideal, or "thumb-tack" ambiguity function for the frequency hop pattern. [A "thumb-tack" (US) is a "drawing pin" (UK).]

We may represent the frequency hop pattern by an $n \times n$ permutation matrix (a_{ij}), where $a_{ij} = 1$ if and only if frequency f_i is used at time t_j. (Otherwise $a_{ij} = 0$.) The extra condition is that the $\binom{n}{2}$ "vectors" connecting the n positions in the matrix where 1s are located are all distinct *as vectors*: no two vectors are the same in both magnitude and slope. One may visualise a *dot* at each position where $a_{ij} = 1$. When the pattern is shifted in both time (horizontally) and frequency (vertically), any dot can be brought into coincidence with any other dot. However, the extra "Costas" condition is that no such shift (other than the identity, which is no shift at all) will bring *two* dots into coincidence with two other dots.

Costas succeeded, initially, in finding examples, by exhaustive computer search, only for $n \leq 12$. However, several systematic constructions for these "Costas Arrays" are now known, giving examples for arbitrarily large values of n. All of these systematic constructions are based on the existence of primitive roots in finite fields. Three such constructions are:

1. **The Welch Construction**, for $n = p - 1$ and $p - 2$, p prime.

   Let g be a primitive root modulo p. The "dots" of the permutation matrix occur at the locations (i, g^i) for $1 \leq i \leq p - 1$, giving a Costas Array of order $n = p - 1$.

   Since $g^{p-1} \equiv 1 \pmod{p}$, there is a dot at $(p-1, g^{p-1})$, which is at a corner of the matrix. Removing the row and column of this dot leaves a Costas Array of order $n = p - 2$.

2. **The Lempel Construction**, for $n = q - 2$, q a prime power.

   Let α be a primitive element in $GF(q)$. The "dots" of the permutation matrix occur at the locations (i, j) whenever $\alpha^i + \alpha^j = 1$ in $GF(q), 1 \leq i, j \leq n - 2$. (This always produces a *symmetric* matrix.)

3. **The Golomb Construction**, for $n = q - 2$ and $q - 3$, q a prime power.

   Let α and β be any two primitive elements in $GF(q)$. The "dots" of the permutation matrix occur at the locations (i, j) whenever $\alpha^i + \beta^j = 1$ in $GF(q), 1 \leq i, j \leq n - 2$. (The special case when $\alpha = \beta$ is the Lempel Construction.)

Table 2. The number of Costas Arrays for $n \leq 23$; where $C(n)$ is the total number, $c(n)$ is the reduced number, and $s(n)$ is the number of symmetric Costas Arrays of order n

n	$C(n)$	$c(n)$	$s(n)$
2	2	1	1
3	4	1	1
4	12	2	1
5	40	6	2
6	116	17	5
7	200	30	10
8	444	60	9
9	760	100	10
10	2160	277	14
11	4368	555	18
12	7852	990	17
13	12828	1616	25
14	17252	2168	23
15	19612	2467	31
16	21104	2648	20
17	18276	2294	19
18	15096	1892	10
19	10240	1283	6
20	6464	810	4
21	3536	446	8
22	2052	259	5
23	872	114	10

It has been shown that for all $q > 2$, the field $GF(q)$ contains primitive elements α and β (not necessarily distinct) with $\alpha + \beta = 1$. Using such α and β in the Golomb Construction, since $\alpha^1 + \beta^1 = 1$, we have (1,1) as the location of a "dot" in the construction. Removing the top row and left column of the matrix leaves a $(q-3) \times (q-3)$ Costas Array.

For proofs that these three constructions must yield Costas Arrays, see [22]. For additional variants, and the way they yield examples of Costas Arrays for many values of $n \leq 360$, see [23]. Since 1984, the smallest values of n for which no examples of Costas Arrays are known are $n = 32$ and $n = 33$.

References

1. Shannon, C. A mathematical theory of communication. *Bell System Technical J.*, **27**, Part I: 379–423, Part II: 623–656.

2. Gallager, R. (1968). *Information Theory and Reliable Communication*, Wiley, New York.
3. Golomb, S.W., Peile, R.E. and Scholtz, R.A. (1994). *Basic Concepts in Information Theory and Coding*, Plenum Press, New York.
4. Selin, I. (1965). *Detection Theory*, Princeton University Press, New Jersey.
5. Fano, R. (1961). *Transmission of Information*, M.I.T. Press, Massachusetts.
6. Black, H.S. (1953). *Modulation Theory*, Van Nostrand, New York.
7. Baumert, L.D. (1971). *Cyclic Difference Sets*, *Lecture Notes in Maths.*, **182**, Springer-Verlag, Germany.
8. Hall, M., Jr. (1986). *Combinatorial Theory*, 2nd Edition, Wiley-Interscience, New York.
9. Turyn, R. (1968). Sequences with small correlation. *Error Correcting Codes*, Editor: H.B. Mann, Wiley, New York, 195–228.
10. Gordon, B., Mills, W.H. and Welch, L.R. (1962). Some new difference sets. *Canadian J. Maths.*, **14**, 614–625.
11. Golomb, S.W. (1967 and 1982). *Shift Register Sequences*, Holden-Day, Inc., San Francisco, and (Revised Edition) Aegean Park Press, Laguna Hills, USA.
12. Golomb, S.W. (1960). The twin prime constant. *Amer. Math. Monthly*, **67**, 767–769.
13. Barker, R.H. (1953). Group synchronization of binary digital systems. *Communication Theory, Proc. 2nd Sym. Information Theory*, London, 273–287.
14. Turyn, R. and Storer, J. (1961). On binary sequences. *Proc. Amer. Math. Soc.*, **12**, 394–399.
15. Golomb, S.W. and Scholtz, R.A. (1965). Generalized Barker sequences. *IEEE Transactions on Information Theory*, **IT-11**, 533–537.
16. Zhang, N. and Golomb, S.W. (1989). Sixty-phase generalized Barker sequences. *IEEE Transactions on Information Theory*, **IT-35**, 911–912.
17. Frank, R.L. (1963). Polyphase codes with good nonperiodic correlation properties. *IEEE Transactions on Information Theory*, **IT-9**, 43–45.
18. Huffman, D.A. (1962). The generation of impulse-equivalent pulse trains. *IRE Transactions on Information Theory*, **IT-8**, S10–S16.
19. Gardner, M. (1983). *Wheels, Life and Other Mathematical Amusements*, W.H. Freeman and Company, New York, 152–165.
20. Bloom, G.S. and Golomb, S.W. (1977). Applications of numbered, undirected graphs. *Proc. IEEE*, **65**, 562–571.
21. Costas, J.P. (1984). A study of a class of detection waveforms having nearly ideal range—doppler ambiguity properties. *Proc. IEEE*, **72**, 996–1009.

22. Golomb, S.W. (1984). Algebraic constructions for Costas arrays. *J. Combinatorial Theory (A)*, **37**, 13–21.

23. Golomb, S.W. and Taylor, H. (1984). Constructions and properties of Costas arrays. *Proc. IEEE*, **72**, 1143–1163.

24. Golomb, S.W. (1992). Two-valued sequences with perfect periodic autocorrelation. *IEEE Transactions on Aerospace and Electronic Systems*, **28**, 383–386.

25. Golomb, S.W. and Taylor, H. (1985). Tuscan squares—a new family of combinatorial designs. *Ars Combinatorica*, **20-B**, 115–132.

26. Cheng, U. (1983). Exhaustive Construction of (225, 127, 63)-cyclic difference sets. *J. Combinatorial Theory (A)*, **35**, 115–125.

A Constructive Algorithm for Neural Network Design with Application to Channel Equalization

Catherine Z.W. Hassell Sweatman*, Gavin J. Gibson and Bernard Mulgrew***

**Department of Electrical Engineering, **Department of Biomathematics and Statistics, University of Edinburgh, Scotland*[1]

1 Introduction

We describe a deterministic algorithm for designing a multilayer perceptron (MLP) [11] for the solution of a two–state classification problem. This algorithm is illustrated in the context of channel equalization [4], but is more generally applicable.

We consider MLPs constructed from McCulloch–Pitts units [5]. A McCulloch–Pitts unit or node (see Figure 1) accepts a real–valued input $\underline{y} \in \mathbb{R}^m$ and calculates as its output the quantity $f(\underline{y}^T \underline{w} - \theta)$, where f is the Heaviside step function and $\underline{w} \in \mathbb{R}^m$ and $\theta \in \mathbb{R}$ are the weight vector and threshold of the node, respectively.

Attention is restricted to MLPs with one hidden layer of nodes and a single output node (see [3] and Figure 1). The outputs of the nodes in the first (hidden) layer are the inputs to the output node.

2 Channel equalization

The algorithm presented was motivated by the problem of reconstructing digital signals which have been passed through a dispersive channel and corrupted with additive noise as depicted in Figure 2. Explicitly, a random sequence $\{x_i\}$, $x_i \in \{-1, 1\}$, is passed through a real linear dispersive channel of finite impulse response with response function $a(z) = a_0 + a_1 z^{-1} + ... + a_k z^{-k}$, where the coefficients a_j are real, $0 \leq j \leq k$, and a_0 and a_k are non–zero; producing a sequence of outputs, $\{y_i\}$, where $y_i = \Sigma_{j=0}^{k} a_j x_{i-j}$. A term, σ_i, which represents additive noise, is then added to each y_i to produce an observation sequence $\{\hat{y}_i\}$. The task of the equalizer is to use the information represented by the observed channel outputs $\hat{y}_i$, $\hat{y}_{i-1}$, $\ldots$, $\hat{y}_{i-m+1}$, to produce an estimate of the input symbol x_{i-d}, where the integer $d \geq 0$ is the *delay* and $m \geq d$. The integer m is the *order* of the equalizer. The input samples x_i are chosen from $\{-1, 1\}$ with equal probability and are assumed to be independent of one another.

[1]Supported by EPSRC Grant GR/J34248

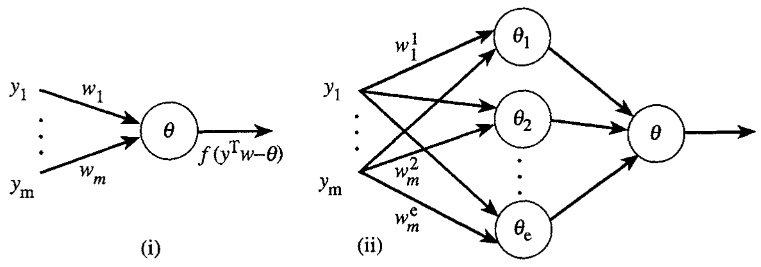


Figure 1. i) A single node, ii) MLP with one hidden layer

In the absence of noise the problem is to separate two sets of points in $\mathbb{R}^m$. These sets of points are the images of the sets

$$X_1 = \{(x_0, \dots, x_{-k-m+1})^T \ : \ x_i \in \{-1, 1\}, \ -k-m+1 \le i \le 0, \ x_{-d} = 1\}$$

and

$$X_{-1} = \{(x_0, \dots, x_{-k-m+1})^T \ : \ x_i \in \{-1, 1\}, \ -k-m+1 \le i \le 0, \ x_{-d} = -1\}$$

under the linear transformation represented by the $m \times (m+k)$ matrix

$$A = \begin{pmatrix} a_0 & a_1 & a_2 & \dots & a_k & 0 & 0 & \dots & 0 \\ 0 & a_0 & a_1 & \dots & a_{k-1} & a_k & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots & 0 & a_0 & a_1 & \dots & a_k \end{pmatrix}$$

Let $P_{(m,d)}(1) = AX_1$ and $P_{(m,d)}(-1) = AX_{-1}$ as defined in ([4]).

When noise is present, the observed channel outputs $\underline{\hat{y}} = (\hat{y}_i, \dots, \hat{y}_{i-m+1})^T$ represent elements of $P_{(m,d)}(\pm 1)$ which are corrupted in each component independently by noise. For low values of the noise variance, each $\underline{\hat{y}}$ is very close to an element of $P_{(m,d)}(1)$ or $P_{(m,d)}(-1)$. The equalizer must represent some function

$$\begin{array}{cccc} g & : & \mathbb{R}^m & \rightarrow & \{-1, 1\} \\ & & \underline{\hat{y}} & \mapsto & \hat{x} \end{array}$$

such that $g(A(x_i, x_{i-1}, \dots, x_{i-(k+m-1)})^T) = x_{i-d}$.

Previous attempts to use the MLP as a channel equalizer are discussed by [4]. In general, $P_{(m,d)}(1)$ and $P_{(m,d)}(-1)$ cannot be separated by a single hyperplane, that is, are not linearly separable, and so a linear transversal equalizer (represented by a single node) will fail to separate them. It has been shown that for $d = 0$, the sets will be linearly separable, for all sufficiently large m, if and only if the channel is minimum phase; that is, if all the roots of $z^k a(z)$ lie strictly within the unit circle in the complex plane [4]. Even if the two sets are

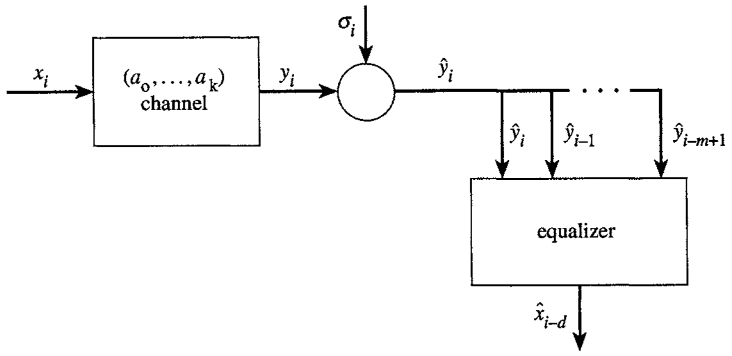


Figure 2. Transmission system

linearly separable, the optimal decision boundary (see Section 4.2.2) is generally non–linear.

Motivated by these difficulties, the MLP was considered. Simulations have shown [4] that a MLP with two hidden layers trained by back propagation [8] can approximate the decision boundary of an optimal equalizer better than can a linear transversal equalizer. In this study, the architecture was chosen by experiment. The training was slow and the decision boundaries obtained were sub–optimal in general. The convergence was often too slow to be of use in the case of time–dependent response function coefficients. In this paper we adopt an alternative approach.

There are basically two strategies for channel equalization:

- estimating the channel characteristics from the data and constructing the data equalizer; or
- estimating the equalizer directly from the data.

The former strategy is usually preferred in the case of time–dependent channel characteristics, or when speed is required, as fewer parameters need to be estimated. Methods for estimating the channel are described and assessed in [6].

These observations led us to develop and propose a new algorithm we call the Slab Algorithm to construct a solution in the form of a MLP with one hidden layer and a single output node, assuming knowledge of the channel characteristics.

3 The Slab Algorithm

Let U and V be the cells of a partition of a finite set of points in $\mathbb{R}^m$. Our aim is to construct a classifier that separates U from V. Let $U_0 = U$ and $V_0 = V$. Iterate the following steps, $p \geq 1$.

1. Find a slab
$$S_p = \{\underline{y} \in \mathbb{R}^m \ : \ a_p \leq \underline{y}^T \underline{w}^p \leq b_p\}$$
where a_p, $b_p \in \mathbb{R}$, $a_p \leq b_p$, $\underline{w}^p \in \mathbb{R}^m$ and $\underline{w}^p \neq \underline{0}$, such that

   (a)
$$U_{p-1} \subseteq \{\underline{y} \in \mathbb{R}^m \ : \ a_p \leq \underline{y}^T \underline{w}^p\},$$
$$V_{p-1} \subseteq \{\underline{y} \in \mathbb{R}^m \ : \ \underline{y}^T \underline{w}^p \leq b_p\}$$
   and

   (b) the width of the slab is minimal (in the sense defined below).

2. Let $U_p = U_{p-1} \cap S_p$ and let $V_p = V_{p-1} \cap S_p$. If $U_p = V_p = \emptyset$ then stop. Otherwise, return to step 1, increasing p by one.

We use standard linear programming techniques to identify the slabs. By "find a slab of minimal width" we mean

- if U_{p-1} and V_{p-1} are linearly separable then S_p must be a separating hyperplane such that $(U_{p-1} \cup V_{p-1}) \cap S_p = \emptyset$,
- otherwise, attempt to minimize the number of elements in $(U_{p-1} \cup V_{p-1}) \cap S_p$.

Knowledge of the slabs S_p, $p \geq 1$, enables a separating network to be specified. If U and V are linearly separable, the Slab Algorithm yields a single separating hyperplane, S_1. A network comprising a single node with weight vector $\underline{w}^1$ and threshold $a_1 = b_1$ will separate U from V.

Otherwise, let $S_1, \ldots, S_{q+1}$, $q \geq 1$, be the slabs calculated. For p, $1 \leq p \leq q$, S_p specifies a pair of hidden layer nodes. The hidden layer nodes are assigned weights $w_1^p, \ldots, w_m^p$ with threshold a_p and weights $-w_1^p, \ldots, -w_m^p$ with threshold $-b_p$. The weights for the output node in the second layer which multiply the outputs from the two nodes corresponding to S_p are $1/2^p$ and $-1/2^p$, respectively. These two nodes combine to contribute $1/2^p$ to the output node from points in $U_{p-1} \backslash S_p$, $-1/2^p$ to the output node from points in $V_{p-1} \backslash S_p$ and zero from points in $(U_{p-1} \cup V_{p-1}) \cap S_p$, $1 \leq p \leq q$. The last node in the hidden layer is specified by the separating hyperplane

$$S_{q+1} = \{\underline{y} \in \mathbb{R}^m \ : \ \underline{y}^T \underline{w}^{q+1} = a_{q+1} = b_{q+1}\}$$

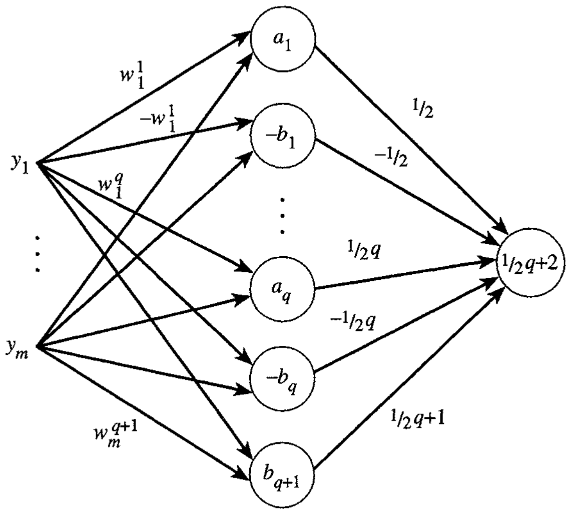


Figure 3. MLP constructed by the Slab Algorithm

such that

$$U_q \subseteq \{\underline{y} \in \mathbb{R}^m \; : \; \underline{y}^T \underline{w}^{q+1} > b_{q+1}\}$$

and

$$V_q \subseteq \{\underline{y} \in \mathbb{R}^m \; : \; \underline{y}^T \underline{w}^{q+1} < b_{q+1}\}.$$

The last hidden layer node has weights $w_1^{q+1}, \ldots, w_m^{q+1}$ with threshold b_{q+1}. The corresponding weight for the output node in the second layer is $1/2^{(q+1)}$. This output node is assigned the threshold $1/2^{(q+2)}$ (see Figure 3). We make use of the geometric series

$$1/2 + 1/4 + 1/8 + \ldots.$$

Note that

$$1/2 \pm 1/4 \pm 1/8 \pm \ldots \pm 1/2^{q+1} > 1/2^{(q+2)} > 0.$$

Hence, when $q \geq 1$ and $1 \leq p \leq q$, points in $U_{p-1} \backslash S_p$ and $V_{p-1} \backslash S_p$ are well classified by the pair of nodes corresponding to S_p and will not be misclassified by the addition of further hidden layer nodes.

The number of nodes generated by this process is guaranteed finite because we ensure that at least one point in $U_{p-1} \cup V_{p-1}$ is excluded from the slab S_p at each step.

It is possible to replace the p-th pair of hidden layer nodes, $1 \leq p \leq q$, when $q \geq 1$, by a single hidden layer node with two thresholds, a_p and b_p and node activation function

$$f \; : \; \mathbb{R} \; \rightarrow \; \{-1, 0, 1\}$$

defined by $f(x) = -1$ if $x < a_p$, $f(x) = 0$ if $a_p \leq x \leq b_p$ and $f(x) = 1$ if $x > b_p$. The outputs from these nodes (inputs to the output node) are multiplied by the weights $1/2$, $1/4$, $1/8, \ldots, 1/2^q$, respectively.

The slabs S_p, $1 \leq p \leq q+1$, $q \geq 0$, are determined by linear programming as follows.

1. We first try to separate U_{p-1} from V_{p-1} with a hyperplane

$$S_p = \{\underline{y} \in \mathbb{R}^m \ : \ a_p = \underline{y}^T \underline{w}^p = b_p\}$$

such that

$$U_{p-1} \subseteq \{\underline{y} \in \mathbb{R}^m \ : \ \underline{y}^T \underline{w}^p > a_p\}$$

and

$$V_{p-1} \subseteq \{\underline{y} \in \mathbb{R}^m \ : \ \underline{y}^T \underline{w}^p < a_p\}.$$

Linear programming constraints are expressed in the form of linear inequalities or equations, but not as strict inequalities. Hence we seek a separating slab of non-zero thickness

$$S'_p = \{\underline{y} \in \mathbb{R}^m \ : \ c_p \leq \underline{y}^T \underline{w}^p \leq d_p\}$$

where $c_p, d_p \in \mathbb{R}$, $c_p + 1 = d_p$, $\underline{w}^p \in \mathbb{R}^m$, $\underline{w}^p \neq \underline{0}$, such that

$$U_{p-1} \subseteq \{\underline{y} \in \mathbb{R}^m \ : \ \underline{y}^T \underline{w}^p \geq d_p\}$$

and

$$V_{p-1} \subseteq \{\underline{y} \in \mathbb{R}^m \ : \ \underline{y}^T \underline{w}^p \leq c_p\}.$$

The constraints which must be satisfied are

(a) $\underline{u}^T \underline{w}^p \geq d_p$ if $\underline{u} \in U_{p-1}$

(b) $\underline{v}^T \underline{w}^p \leq c_p$ if $\underline{v} \in V_{p-1}$ and

(c) $0 \leq d_p - c_p \leq 1$

while minimizing $c_p - d_p$. Begin by setting $\underline{w}^p = \underline{0}$ and $c_p = d_p = 0$. If U_{p-1} and V_{p-1} are linearly separable then, by linear programming, we can find a separating slab S'_p such that $c_p + 1 = d_p$ and $\underline{w}^p \neq \underline{0}$. Set $S_p = \{\underline{y} \in \mathbb{R}^m \ : \ \underline{y}^T \underline{w}^p = 1/2 \times (c_p + d_p)\}$.

No more slabs are required.

2. Failure to find a suitable $\underline{w}^p$, c_p and d_p as described above indicates that U_{p-1} and V_{p-1} are not linearly separable. In this case, try to find a slab

$$S'_p = \{\underline{y} \in \mathbb{R}^m \ : \ a'_p \leq \underline{y}^T \underline{w}^p \leq b'_p\}$$

where a'_p, $b'_p \in \mathbb{R}$, $a'_p + 1 = b'_p$, $\underline{w}^p \in \mathbb{R}^m$, $\underline{w}^p \neq \underline{0}$,

$$U_{p-1} \subseteq \{\underline{y} \in \mathbb{R}^m \ : \ \underline{y}^T \underline{w}^p \geq a'_p\}$$

and

$$V_{p-1} \subseteq \{\underline{y} \in \mathbb{R}^m \ : \ \underline{y}^T \underline{w}^p \leq b'_p\},$$

containing as few elements as possible in $U_{p-1} \cup V_{p-1}$.

The constraints to satisfy are

(a) $\underline{u}^T \underline{w}^p \geq a'_p$ if $\underline{u} \in U_{p-1}$

(b) $\underline{v}^T \underline{w}^p \leq b'_p$ if $\underline{v} \in V_{p-1}$ and

(c) $b'_p - a'_p = 1$

while minimizing

$$-\Sigma_{\underline{u} \in U_{p-1}} (\underline{u}^T \underline{w}^p - b'_p) + \Sigma_{\underline{v} \in V_{p-1}} (\underline{v}^T \underline{w}^p - a'_p).$$

Begin by setting $\underline{w}^p = \underline{0}$ and $a'_p = b'_p = 0$.

This method yields a suitable slab

$$S'_p = \{\underline{y} \in \mathbb{R}^m \ : \ a'_p \leq \underline{y}^T \underline{w}^p \leq b'_p\}$$

whose upper bounding hyperplane

$$\{\underline{y} \in \mathbb{R}^m \ : \ \underline{y}^T \underline{w}^p = b'_p\}$$

contains elements in V_{p-1} and whose lower bounding hyperplane

$$\{\underline{y} \in \mathbb{R}^m \ : \ \underline{y}^T \underline{w}^p = a'_p\}$$

contains elements in U_{p-1}. With our noisy channel equalization application in mind, we shift the slab boundaries outwards. If $U_{p-1} \backslash S_p \neq \emptyset$, let

$$b''_p = \min_{\underline{u} \in (U_{p-1} \backslash S_p)} \{\underline{u}^T \underline{w}^p\}$$

and let $b_p = 1/2 \times (b'_p + b''_p)$. Else, let $b_p = b'_p + 1/2$. If $V_{p-1} \backslash S_p \neq \emptyset$, let

$$a''_p = \max_{\underline{v} \in (V_{p-1} \backslash S_p)} \{\underline{v}^T \underline{w}^p\}$$

and let $a_p = 1/2 \times (a'_p + a''_p)$. Else, let $a_p = a'_p - 1/2$. Define

$$S_p = \{\underline{y} \in \mathbb{R}^m \ : \ a_p \leq \underline{y}^T \underline{w}^p \leq b_p\}.$$

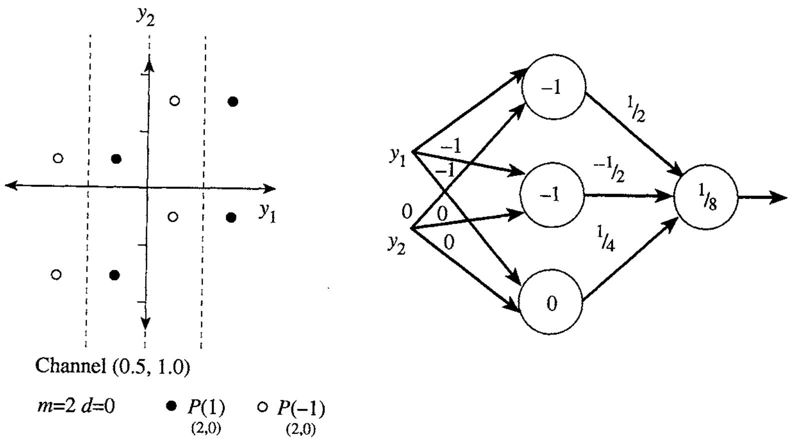


Figure 4. MLP constructed by the Slab Algorithm for channel 1

Sometimes, one of a pair of hidden layer nodes may be eliminated. If $V_{p-1} \subset S_p$, replace S_p by a single hyperplane, namely

$$\{\underline{y} \in \mathbb{R}^m \; : \; \underline{y}^T \underline{w}^p = b_p\},$$

corresponding to a single hidden layer node with weights $w_1^p, \ldots, w_m^p$ and threshold b_p. The corresponding weight for the output node in the second layer is $1/2^{(p)}$. Similarly, if $U_{p-1} \subset S_p$, replace S_p by the hyperplane

$$\{\underline{y} \in \mathbb{R}^m \; : \; \underline{y}^T \underline{w}^p = a_p\},$$

corresponding to a single hidden layer node with weights $-w_1^p, \ldots, -w_m^p$ and threshold $-a_p$. The corresponding weight for the output node in the second layer is $-1/2^{(p)}$.

In the case of channel equalization, the hyperplanes bounding each slab may be chosen equidistant from the origin and the final separating hyperplane may be chosen to contain the origin (see Figures 4 and 5). Note that $\underline{y} \in P_{(m,d)}(1)$ if and only if $-\underline{y} \in P_{(m,d)}(-1)$. Taking account of this symmetry reduces the number of parameters to be determined (for a general study see [9]). In a more general case, this symmetry will not be present. We also require $\underline{0} \notin P_{(m,d)}(\pm 1)$.

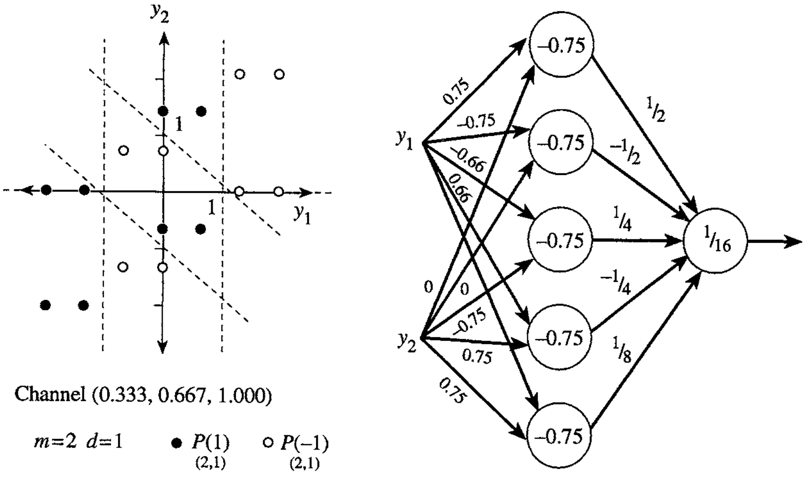


Figure 5. MLP constructed by the Slab Algorithm for channel 2

4 Comparisons

4.1 The Slab Algorithm versus the Upstart Algorithm

Another algorithm designed to separate two finite disjoint sets of points by constructing a MLP with one hidden layer and one output node is the Upstart Algorithm [2]. It is more specialized than our algorithm, designed only to separate two disjoint sets of hypercube vertices in Euclidean space. The hypercube in $\mathbb{R}^m$ is $\{\underline{y} \in \mathbb{R}^m \ : \ 0 \le y_i \le 1, 1 \le i \le m\}$. Its vertices are the elements of $\mathbb{B}^m$ where $\mathbb{B} = \{0, 1\}$.

Both algorithms yield the most efficient solution for the parity problem. In this problem, a hypercube vertex is labelled positive if the sum of its components is odd and is labelled negative otherwise. Given the parity problem in $\mathbb{R}^m$, both algorithms produce a layer of m hidden nodes and an output node. These hidden nodes can be visualized as parallel hyperplanes slicing diagonally through the hypercube.

In the random mapping problem, each hypercube vertex in $\mathbb{R}^m$ is labelled positive or negative at random, with equal probability. Frean [2] reports that the number of hidden nodes generated by the Upstart Algorithm is approximately $2^m/9$, $1 \le m \le 10$. The Slab Algorithm generates approximately $2^m/5$ slabs or $2^m/4$ hidden layer nodes (after pruning), $3 \le m \le 6$.

The Upstart Algorithm cannot be used alone as a channel equalizer as it is designed only to separate two disjoint sets of hypercube vertices.

4.2 The Slab Algorithm applied to channel equalization

4.2.1 Linear Transversal Equalizers

The most commonly used equalizer is the linear transversal equalizer (LTE) trained by the least means squares method (LMS) [10]. A LTE, though not difficult to train, will succeed in the absence of noise if and only if the two sets $P_{(m,d)}(1)$ and $P_{(m,d)}(-1)$ are linearly separable. This is not always the case [4]. Even if the two sets are linearly separable, the optimal decision boundary is generally non–linear [4].

The Wiener optimal filter of order m and delay d is the LTE obtained by minimising a statistical mean squared error cost function. It is estimated by minimising the data dependent least mean squares error cost function [7]. If the sets $P_{(m,d)}(1)$ and $P_{(m,d)}(-1)$ are linearly separable, the Slab Algorithm produces a LTE which appears to approximate the Wiener optimal filter well.

4.2.2 Bayesian optimal equalizers

Given values for the channel response function, the order, the delay and a specification of the additive noise at the output, we can determine the output of an equalizer which minimizes the probability of a wrong decision in the estimation of the transmitted signal x_{i-d}.

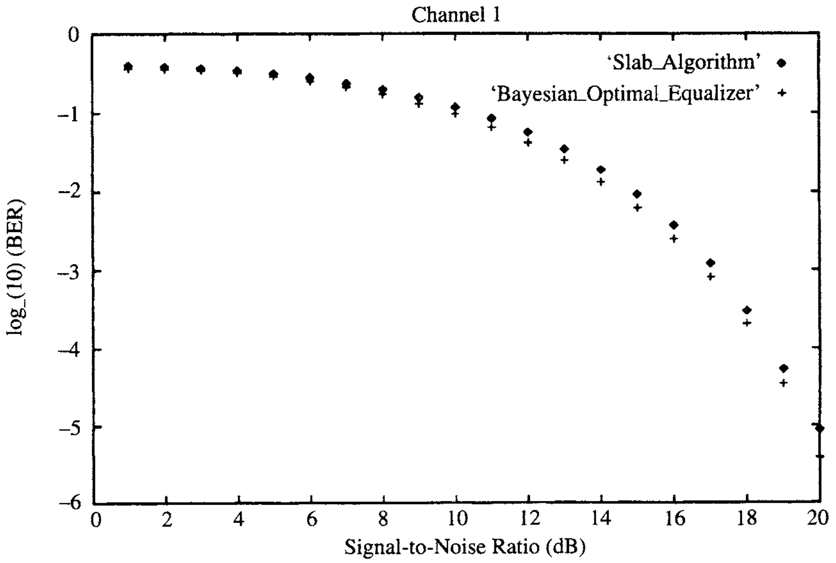


Figure 6. Slab Algorithm and optimal equalizer BERs for channel 1

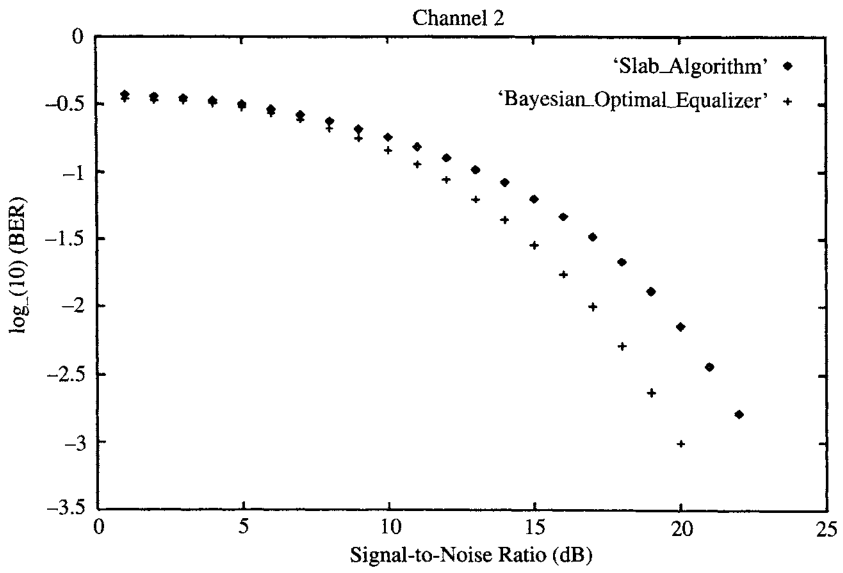


Figure 7. Slab Algorithm and optimal equalizer BERs for channel 2

Let $f_1 : \mathbb{R}^m \rightarrow \mathbb{R}$ be the probability density function of the observed channel output vectors $\{\hat{\underline{y}}_i \in \mathbb{R}^m : \underline{y}_i \in P_{(m,d)}(1)\}$ and let f_{-1} be similarly defined. Let

$$g : \mathbb{R}^m \rightarrow \mathbb{R}$$

be defined by

$$g(y) \mapsto \text{sgn}(f_1(\underline{y}) - f_{-1}(\underline{y}))$$

where $\text{sgn}(x) = 1$ if $x \geq 0$ and $\text{sgn}(x) = -1$ otherwise. An equalizer whose output is g minimizes the probability of a wrong decision in the estimation of the transmitted signal x_{i-d} [4]. We call it a Bayesian optimal equalizer, or simply an optimal equalizer. If the additive noise at the output is Gaussian, an optimal equalizer is constructed naturally using a radial basis function network [1] but may require a prohibitively large number of parameters for even modest values of k, the channel memory. Failure rates, called bit error rates (BERs), for the two channels defined by

1. $k = 1$, $m = 2$, $d = 0$, $a_0 = 0.5$, $a_1 = 1.0$ (see Figures 4 and 6) and
2. $k = 2$, $m = 2$, $d = 1$, $a_0 = 0.333$, $a_1 = 0.667$, $a_2 = 1.000$ (see Figures 5 and 7)

were simulated at signal-to-noise ratios between $1dB$ and $20dB$, in steps of $1dB$, using decision boundaries formed by

1. the Slab Algorithm and
2. an optimal equalizer.

It was assumed that the additive noise samples σ_i were chosen randomly and independently from a Gaussian distribution with mean 0 and variance σ^2.

In each case, the sets $P_{(m,d)}(\pm 1)$ are not linearly separable. These BERs suggest that the Slab Algorithm may be used to construct a channel equalizer whose performance approximates that of a Bayesian optimal equalizer well, especially at high signal–to–noise ratios.

4.2.3 Channel estimation

We have assumed knowledge of the channel characteristics. In practice, the delay and order are known and the channel response function vector $\underline{a} = (a_0, \ldots, a_k)$ is estimated [1]. In general, a MLP constructed by the Slab Algorithm using estimated channel response function coefficients generates a higher BER, at a given signal–to–noise ratio, than the MLP constructed using the true values of the coefficients. We assume that the additive noise samples σ_i are chosen randomly and independently from a Gaussian distribution with mean 0 and variance σ^2. Let $f : \mathbb{R}^{k+1} \rightarrow \mathbb{R}$ be the probability density function of the estimated channel response function vectors $\underline{a}' \in \mathbb{R}^{k+1}$. We assume that f is a Gaussian distribution with mean $\underline{a} = (a_0, \ldots, a_k)$ and covariance matrix equal to

$$\frac{\sigma^2 \mathrm{I}_{k+1}}{\sigma_x^2 M}$$

where σ^2 is the power of the additive noise samples at the output of the channel, I_{k+1} is the identity matrix, M is the number of observations taken in the process of estimating $\underline{a}$ and σ_x^2 is the power of the incoming signal. We assume $\sigma_x^2 = 1$ and $M = 4(k+1)$.

The effect of estimating the channel response function coefficients was studied for the two channels

1. $k = 1$, $m = 2$, $d = 0$, $a_0 = 0.5$, $a_1 = 1.0$ and
2. $k = 2$, $m = 2$, $d = 1$, $a_0 = 0.333$, $a_1 = 0.667$, $a_2 = 1.000$

at (output) signal–to–noise ratios of

1. $7dB$
2. $18dB$.

In each case, the bit error rate generated by the MLP constructed using the true channel response function vector, namely BER($\underline{a}$), is compared with an estimate of

$$\int_{\mathbb{R}^{k+1}} \mathrm{BER}(\underline{a}') f(\underline{a}') d\underline{a}'$$

Table 1. Slab Algorithm

	Channel 1 (7dB)	Channel 1 (18dB)	Channel 2 (7dB)	Channel 2 (18dB)
BER($\underline{a}$)	0.238	0.000299	0.264	0.0215
$\log_{10}$(BER($\underline{a}$))	-0.624	-3.525	-0.578	-1.668
estimate of				
$\int_{\mathrm{R}^{k+1}}$ BER($\underline{a}'$)$f(\underline{a}')d\underline{a}'$	0.259	0.000815	0.296	0.109
$\log_{10}$ (estimate)	-0.587	-3.089	-0.528	-0.964

Table 2. Bayesian optimal equalizer

	Channel 1 (7dB)	Channel 1 (18dB)	Channel 2 (7dB)	Channel 2 (18dB)
BER($\underline{a}$)	0.212	0.000207	0.243	0.00517
$\log_{10}$(BER($\underline{a}$))	-0.674	-3.684	-0.615	-2.287
estimate of				
$\int_{\mathrm{R}^{k+1}}$ BER($\underline{a}'$)$f(\underline{a}')d\underline{a}'$	0.240	0.000425	0.281	0.00751
$\log_{10}$ (estimate)	-0.621	-3.372	-0.552	-2.125

where BER($\underline{a}'$) is the bit error rate generated by a MLP constructed using the estimate $\underline{a}'$ in place of $\underline{a}$ (see Tables 1 and 2). The integrals were estimated by sampling at random from the probability density function f, calculating a BER at each sampled point $\underline{a}'$ and averaging.

5 Conclusions

The Slab Algorithm will always succeed in separating two finite disjoint sets of points in $\mathbb{R}^m$. It produces a MLP whose design is simple and predictable. If the sets to be separated are linearly separable, it constructs a single separating node. Otherwise, it constructs a MLP with one hidden layer and one output node. In the latter case, the decision boundary is non–linear, formed from hyperplanes.

It is fast and efficient, especially when separating sets U and V exhibiting the symmetry $\underline{y} \in U$ if and only if $-\underline{y} \in V$.

Our results suggest that the Slab Algorithm may be used to construct a channel equalizer whose performance approximates that of an optimal equalizer well, especially at high signal–to–noise ratios. Its speed may mean it is useful in the case of time varying channels.

Our simulations suggest that the equalizer constructed is robust with respect to error in channel estimation at low signal–to–noise ratios. It remains to be seen how the performance at high signal–to–noise ratios can be improved; for example, by increasing M, the number of observations taken in the estimation process.

References

1. Newson, P. and Mulgrew, B. (1994). Model based adaptive channel identification algorithms for digital mobile radio. *IEEE Int. Conf. Communications*, 1531–1535.

2. Frean, M. (1990). The Upstart Algorithm: a method for constructing and training feed–forward neural networks. *Neural Computation*, **2**, 198–209.

3. Gibson, G.J. (1993). A combinatorial approach to understanding perceptron capabilities. *IEEE Transactions on Neural Networks*, **4**, 989–992.

4. Gibson, G.J., Siu, S. and Cowan, C.F.N. (1991). The application of nonlinear structures to the reconstruction of binary signals. *IEEE Transactions on Signal Processing*, **39**, 1877–1884.

5. McCulloch, W.S. and Pitts, W. (1943). A logical calculus of the ideas imminent in nervous activity. *Bull. of Mathematical Biophysics*, **5**, 115–133.

6. McLaughlin, S., Mulgrew, B. and Cowan, C.F.N. (1989). A performance study of three adaptive equalisers in the mobile communications environment. *IEEE Int. Conf. Communications—ICC 89*, Boston, 193–197.

7. Mulgrew, B. and Cowan, C.F.N. (1988). *Adaptive filters and Equalisers*, Kluwer Academic Publishers, Massachusetts, USA.

8. Rumelhart, D.E., Hinton, G.D. and Williams, R.J. (1986). Learning internal representations by error propagation. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Editors: D.E. Rumelhart and J.L. McClelland, Cambridge, MA, MIT Press.

9. Shawe–Taylor, J. (1993). Symmetries and discriminability in feedforward network architectures. *IEEE Transactions on Neural Networks*, **4**, 816–826.

10. Widrow, B. and Hoff, M. (1960). Adaptive switching circuits. *IRE WESCON Conv. Record*, **4**, 96–104.

11. Widrow, B., Winter, R. and Baxter, R. (1988). Layered neural nets for pattern recognition. *IEEE Transactions on Acoustics, Speech and Sig. Processing*, **36**, 1109–1118.

The Weight Distribution of KM Codes

Stuart Hoggar and Keith Pickavance

Department of Mathematics, University of Glasgow, Scotland

1 Introduction

This paper is about linear error-correcting codes which *adapt* to the degree of interference encountered. The idea was mooted by Mandelbaum [1] in 1974, and in succeeding years various schemes were proposed. Essentially we require a generator matrix for an (n,k) code partitioned into blocks, $G = [G_1|G_2|\dots|G_L]$, so that the code C_i with generator matrix $[G_1|G_2|\dots|G_i]$ has error correction and/or detection ability greater than C_{i-1}, for $2 \le i \le L$.

Let Z_2 denote the field with elements 0,1 subject to 1+1 = 0. A *linear* (n,k,d) *code* C is here a subset of the n-tuples over Z_2 such that the sum of any two members of C is a third, and further, the *minimum distance* of C, the least number of coordinates in which distinct members of C differ, is at least d. Then when a codeword of C arrives from a noisy channel, up to $d-1$ errors can be detected, and e corrected if $2e+1 \le d$. A generator matrix for C is a $k \times n$ matrix G for which C consists of all possible sums of rows. For more information see [2] or [3].

$$[G_1\; G_2\; G_3\; G_4] =$$

$$\begin{bmatrix} 1\,0\,1 & | & 1\,0\,1 & | & 1\,0\,1 & | & 0 \\ 0\,1\,1 & | & 0\,1\,1 & | & 0\,1\,1 & | & 0 \\ 0\,0\,0 & | & 1\,0\,1 & | & 1\,1\,0 & | & 0 \\ 0\,0\,0 & | & 0\,1\,1 & | & 1\,0\,1 & | & 0 \\ 0\,0\,0 & | & 1\,0\,1 & | & 0\,1\,1 & | & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1\,0\,1 & | & 1\,0\,1 & | & 1\,0\,1 & | & 0 \\ 0\,1\,1 & | & 0\,1\,1 & | & 0\,1\,1 & | & 0 \\ 0\,0\,0 & | & 1\,0\,1 & | & 1\,1\,0 & | & 0 \\ 0\,0\,0 & | & 0\,1\,1 & | & 1\,0\,1 & | & 1 \end{bmatrix} \tag{1.1}$$

- Shunting ($\rightarrow$) to obtain a generator matrix for a KM (10,4,4) code from that of a KM (10,5,3) code The first three blocks come from polynomial multiplications modulo respectively u^2, u^2+1, u^2+u+1; the last column is devoted to wraparound (see Example 1). The "shunting" property is explained in Section 2.

Suppose k, d are given and C_L is to be an (n, k, d) code for some n, preferably as small as possible consistent with the existence of suitable C_1 to C_{L-1}. An ingenious solution is given by the KM codes of [4–7], following the work of Lempel and Winograd [8] which exposed the connection between codes and bilinear forms. Let $z = [z_0, \ldots, z_{k-1}]$ and $y = [y_0, \ldots, y_{d-1}]$ be sequences of indeterminates. The product of their respective generating polynomials $Z(u)$, $Y(u)$ (over Z_2) in a variable u is $F(u) = \sum_r f_r u^r$, where

$$f_0 = z_0 y_0, \quad f_1 = z_0 y_1 + z_1 y_0, \quad f_2 = z_0 y_2 + z_1 y_1 + z_2 y_0, \quad \ldots. \tag{1.2}$$

In the basic version $G_1 \ldots G_L$ are constructed as follows. We start with a polynomial $P(u)$ which is the product of L coprime polynomials,

$$P(u) = P_1(u) \ldots P_L(u) \quad (P_t(u) \text{ of degree } D_t). \tag{1.3}$$

Let $g_t(u)$ be the reduction mod $P_t(u)(1 \le t \le L)$ of any polynomial $g(u)$, and $\hat{g}(u)$ its reduction mod $P(u)$. By the Chinese Remainder Theorem, $\hat{F}(u)$ may be reconstructed from the reductions $F_t(u)$. We choose $P(u)$ of degree $N = k+d-1$ so that determining $\hat{F}(u)$ is equivalent to determining $F(u)$ itself. The length n of the code C_L is the number of multiplications $m_0 \ldots m_{n-1}$ we use to enable each coefficient f_r to be expressed as a sum of certain of these, with

$$m_i = (zg_i^T)(yh_i^T), \text{ where } g_i = [g_{0i} g_{1i} \cdots g_{k-1,i}],\ h_i = [h_{0i} h_{1i} \ldots h_{d-1,i}]. \tag{1.4}$$

Thus $m_i = (g_{0i}z_0 + g_{1i}z_1 + \ldots + g_{k-1,i}z_{k-1}) \times (h_{0i}y_0 + h_{1i}y_1 + \ldots + h_{d-1,i}y_{d-1})$, and $G = [g_{ri}]$ has ith column g_i containing the z-coefficients in m_i. For $1 \le t \le L$ the columns of G_t correspond to the multiplications required to calculate $F_t(u)$.

1.1 Wraparound

We may often save multiplications by going beyond the basic version and allowing *wraparound s*, meaning that we choose $P(u)$ of degree $D = N - s$, with $s = 0$ giving the basic version. For reference

$$D = \text{degree } P(u), \quad N = D + s, \quad k + d = N + 1. \tag{1.5}$$

We add a *wraparound block* to G from the multiplications that enable us to calculate $\overline{Z}(u)\overline{Y}(u)$ mod u^s; here $\overline{g}(u)$ denotes the *reciprocal* polynomial of $g(u)$, obtained by writing the coefficients in reverse order, or formally $\overline{g}(u) = u^{\text{degree } g} g(1/u)$. Note the relation $\overline{fg} = \overline{f}\overline{g}$. (Exercise: verify that $u^r g(u)$ has the same reciprocal as $g(u)$.)

Example 1.

1. *The result of calculating $(a_0 + a_1u)(b_0 + b_1u)$ mod $u^2 + 1$ is $a_0b_0 + a_1b_1 + (a_0b_1 + a_1b_0)u$. Arranged this way, it uses four multiplications, but this may be reduced to three (additions are not counted) by setting $m_0 = a_0b_0$, $m_1 = a_1b_1$, $m_2 = (a_0 + a_1)(b_0 + b_1)$, and re-expressing the product as $(m_0+m_1)+(m_0+m_1+m_2)u$. This is best possible for finding the product of two polynomials modulo a polynomial of degree two, and yields the first three blocks in Equation 1.1 (see Example 4).*

2. *The fourth block in Equation 1.1 corresponds to wraparound $s = 1$. Here $\overline{Z}(u)\overline{Y}(u) = (z_4 + z_3u + z_2u^2 + z_1u^3 + z_0u^4)(y_2 + y_1u + y_0u^2)$ reduces mod u^1 to just z_4y_2, giving a single column block $[0\ 0\ 0\ 0\ 1]^T$. More generally, see Theorem 11.*

We emphasise that to say the product $F(u) = Z(u)Y(u)$ *can be computed with n multiplications* means that there exist $m_0, \ldots, m_{n-1}$ of form (1.4) such that every coefficient f_i implied in Equation 1.2 is a Z_2-linear combination of $m_0, \ldots, m_{n-1}$, i.e. is a sum of certain m_i*'s*. The least such n is called the *multiplicative complexity* of this product, also denoted n_{min} in the present context. Here is the basic Theorem.

Theorem 2. *(***Krishna and Morgera** [6].*) If $Z(u)Y(u)$ can be computed in n multiplications then $G = [g_{ri}]$ is a generator matrix for a KM (n, k, d) code.*

The parameter d is called the *designed distance* of the code, since the actual minimum distance may fortuitously be greater. Now, given a block in G, we may be able to reduce the code's length by omitting from this block any multiplication which is a linear combination of multiplications from earlier blocks, as is done where possible in Section 3. If we do not do this, the block is *self-contained*, or *independent*, in the sense that $Z(u)Y(u)$ mod $P_i(u)$ (or $\overline{ZY}$ mod u^s in a wraparound block) may be computed from multiplications within that block alone.

Of course the error-correcting capability of a KM code depends not only on its minimum distance but on the distribution of other distances. Such codes being linear, the distance between codewords u, v equals the *weight* $wt(u - v)$, where $wt(w)$ is the number of 1's in a codeword w. In particular the minimum distance equals the *least weight* w_{min} of any codeword. Further, the distribution is determined by the *weight enumerator* $A(x) = \sum_{i=0}^{n} A_ix^i$, where A_i is the number of codewords of weight i.

Now, $A(x)$ can be determined by computer for any given code. Our aim is to obtain results about $A(x)$ for families of KM codes in terms of their parameters. After Section 2, where the idea of "shunting" is introduced, we investigate in Section 3, for low values of parameters k, d, how the least possible values of n and w_{min} can be obtained simultaneously by varying (1.3), s, and the choice of multiplications. In Section 4 we present results enabling an approach to $A(x)$ via the dual code and MacWilliams identity, leading to expressions for the A_i in terms of the parameter d for certain families of KM codes related by shunting.

2 Reduction, and a shunting theorem

Remark A. (Wraparound lemma.) Let $F(u) = Z(u)Y(u) = \sum f_r u^r (0 \le r \le D + s - 1)$. Here is how the multiplications giving the reduction of $\hat{F}(u)$ mod $P(u)$, and those from the wraparound block, *together* determine $F(u)$ itself.

The reciprocal of $F(u)$ is $\overline{F}(u) = \sum f_{D+s-1-i} u^i (0 \le i \le D + s - 1)$, with a reduction mod u^s of $\sum f_{D+s-1-i} u^i (0 \le i \le s - 1)$. Thus $f_D, \ldots, f_{D+s-1}$ are determined by the wraparound multiplications. Moreover by definition of $\hat{F}(u)$, there are coefficients b_i such that $F(u) = \hat{F}(u) + (b_0 + b_1 u + \ldots + b_{s-1} u^{s-1})P(u)$. Equating coefficients of the top s powers of u gives $[f_{D+s-1} \cdots f_{D+1} f_D] = [b_{s-1} \ldots b_0]M$, where the s by s matrix M has ith row the truncation to length s of $[0^i p_D \ldots p_0 0^s](0 \le i \le s-1)$, 0^i denoting a sequence of i zeros. But $p_D = 1$ by definition of $P(u)$, so M has a main diagonal of 1's, and is zero below. Thus M is invertible, the b_i as well as $\hat{F}(u)$ are determined by the given multiplications, and therefore so is $F(u)$.

Theorem 3. (Chinese Remainder Theorem, or CRT). *A polynomial $g(u)$ may be determined modulo $P(u)$ of (1.3) from its reductions $g_i(u)$ modulo $P_i(u)$, $1 \le i \le L$, by*

$$g(u) = \sum_{i=1}^{L} S_i(u) g_i(u) \text{ mod } P(u), \tag{2.1}$$

where the polynomials $S_i(u)$ are determined by: $S_i(u) = 1$ mod $P_j(u)$ if $j = i$, otherwise $S_i(u) = 0$ mod $P_j(u)$. Indeed $S_i(u) = R_i(u) \prod P_j(u)(j \neq i)$, where $R_i(u) \prod P_j(u)(j \neq i) = 1$ mod $P_i(u)$. (Further reduction may be required after Equation 2.1 is calculated, as in Equation 4.14.)

2.1 A viewpoint on reduction

First some notation. Bold type $\mathbf{a} = [a_i] = [a_0 a_1 \ldots]$ denotes the sequence/vector of coefficients of a polynomial $a(u) = \sum a_i u^i$. Exceptionally $X(u) = \sum x_i u^i$ has coefficient vector $x = [x_i]$ and similarly for $Y(u)$, $Z(u)$. For simplicity let $P(u)$, of degree D, stand for $P_t(u)$, with u^s a special case. Let $X(u)$, of degree $h - 1$, reduce mod $P(u)$ to $a(u)$, necessarily of degree $D - 1$. Then there are constants r_{ij} such that

$$u^i \text{ mod } P(u) = \sum_{j=0}^{D-1} \mathbf{r}_{ij} \mathbf{u}^j. \tag{2.2}$$

Hence $a(u) = \sum_i x_i \sum_j r_{ij} u^j$, giving $a_j = \sum_i x_i r_{ij}$. We may write this in terms of the h by D matrix $R = [r_{ij}]$, whose first D rows (if $h \geq 1$) form the identity matrix $I = I_D$

$$\mathbf{a} = xR, \quad R^T = [I \ldots], \tag{2.3}$$

$$\text{row } i+1 \text{ of } R = [0 r_{i,0} \ldots r_{i,D-2}] + r_{i,D-1}[p_0 p_1 \cdots p_{D-1}]. \tag{2.4}$$

Thus Equation 2.3 gives the first D rows of R, then applying Equation 2.4 to any row (note the shift) gives the next. On the other hand, an algorithm for the product mod $P(u)$ of polynomials $a(u)$, $b(u)$ of degree $D-1$ gives multiplications $(\mathbf{a}\lambda_i^T)(\mathbf{b}\rho_i^T)$, where λ_i, ρ_i are D-vectors over Z_2. The a-parts form a vector $\mathbf{a}S$, with $S = [\lambda_0^T \lambda_1^T \ldots]$, whose first D columns commonly form the identity I. If $Z(u)$ reduces to $a(u)$ then $\mathbf{a} = zR$, so $\mathbf{a}S = zRS$ and finally

$$G_t = RS, \text{ where } zR \text{ represents reduction mod } P_t(u), \tag{2.5}$$

and

$$G_t = [R|RS'] \text{ if } S = [I|S']. \tag{2.6}$$

Example 4. *The first few rows of R for certain $P(u)$, by Equation 2.3 and Equation 2.4, are shown below:*

P(u)	u^D	u^D+1	u^2+u+1	u^3+u+1	u^4+u^2+1
Matrix R	$\begin{bmatrix} I \\ 0 \\ 0 \end{bmatrix}$ (3D rows)	$\begin{bmatrix} I \\ I \\ I \end{bmatrix}$ (3D rows)	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$

For case $D = 2$, Example 1 gives $[a_0 a_1]S = [a_0 \; a_1 \; a_0 + a_1]$, hence Equation 2.6 applies with $S = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$. With Example 4, this S gives the first three blocks in Equation 1.1.

2.2 Shunting—new codes from old

As indicated in Equation 1.1, we may be able to trade dimension for minimum distance by dropping the last row of G, except in a wraparound block where, because the order of coefficients is reversed in taking reciprocals, we must rather drop the first row and move the rest up one place. We shall call this process *shunting*, and offer criteria and proof, seemingly hard to pin down and lacking in the literature, of when and why it works, i.e. why it produces a KM $(n, k-1, d+1)$ code from an (n, k, d).

Theorem 5. (**Shunting theorem.**) *An (n,k,d) KM code with independent blocks may be shunted to a KM $(n, k-1, d+1)$ code if $k \geq 2$ and $d \geq Max_t$ (degree $P_t(u)$, s).*

Proof. Consider first the case of no wraparound: $s = 0$. Shunting means setting $z_{k-1} = 0$ in all multiplications, and will by Theorem 2 yield a $(n, k-1, d+1)$ code if the y-parts of the multiplications can be modified so that the new ZY can be constructed. The key idea is first to prove the following, without restriction on $k+d$.

Lemma 6. *Let $k \geq 2$ and $d \geq D$ (= degree $P(u)$). Suppose that for certain constants k_{ij}, k-vectors λ_j and d-vectors ρ_j, the multiplications $m_j = (z\lambda_j^T)(y\rho_j^T)$ yield the identity mod $P(u)$*

$$Z(u)Y(u) = \sum_{i=0}^{D-1} \left(\sum_{j=0}^{n-1} k_{ij} m_j \right) u^i. \tag{2.7}$$

Then there are $(d+1)$-vectors σ_j such that the following identity holds mod $P(u)$

$$Z(u)(Y(u) + y_d u^d) = \sum_{i=0}^{D-1} \left(\sum_{j=0}^{n-1} k_{ij} m'_j \right) u^i, \text{ where } m'_j = (z\lambda_j^T)([y|y_d]\sigma_j^T). \tag{2.8}$$

To prove this, assume Equation 2.7 holds, let $\mathbf{a} = [a_0 \ldots a_{D-1}]$ be a vector of indeterminants, set y equal to $\mathbf{a}$ extended by 0's up to length d, and let μ_j be the truncation of ρ_j down to length D. Crucially, the latter two operations are well defined because $d \geq D$. We now have $y\rho_j^T = \mathbf{a}\mu_j^T$, and so

$$Z(u)(a_0 + a_1 + \ldots + a_{D-1}u^{D-1}) = \sum_{i,j} k_{ij}(z\lambda_j^T)(\mathbf{a}\mu_j^T)u^i. \tag{2.9}$$

Having dropped the y's, start again with $y_0 \ldots y_d$ and let $\sum_{i=0}^{D-1} a_i u^i$ be the reduction mod $P(u)$ of $y_0 + y_1 u + \ldots + y_d u^d$. Then $\mathbf{a} = [y_0 \ldots y_d]R$ for the $d+1$ by D matrix R given by Equation 2.3, and substituting for $\mathbf{a}$ in Equation 2.9 gives Equation 2.8 with $\sigma_j = \mu_j R^T$.

Now, with the hypotheses of the present theorem, $d \geq$ degree $P_t(u)$, $1 \leq t \leq L$, so by the lemma, $F(u) = (z_0 + \ldots + z_{k-1}u^{k-1})(y_0 + y_1 u + y_d u^d)$ is reconstructible mod $P_t(u)$ for $1 \leq t \leq L$ and hence mod the product $P(u)$ by the Chinese Remainder Theorem 3. Since $d \geq s$ is also given our multiplications also reconstruct the reciprocal of $F(u)$, mod u^s, by a slight modification of the lemma. Therefore we can reconstruct $F(u)$ itself provided its degree is less than $D + s$ (see Remark A). But this inequality is achieved subject to the required $k + d = D + s + 1$, provided we set $z_{k-1} = 0$, which is precisely what we do in shunting. An application of Theorem 2 with k replaced by $k - 1$ and d by $d + 1$ completes the proof.

3 Minimum length KM codes with $N \leq 4$

3.1 Preliminaries, and cases $N = 1, 2, 3$

We investigate for each given N and admissible k, d the KM codes of least possible length n_{min}. Thus we vary

1. k, d subject to $1 \leq k,\ d \leq N$ and $k + d - 1 = N$.
2. $P(u)$, its factors, and their number L.
3. The order of blocks corresponding to these factors.
4. The wraparound s.
5. The choice of multiplications of least number $n = n_{\text{min}}$.

Minimality of the length implies that a block column is deleted if it corresponds to a multiplication that is a linear combination of those already used. Note that columns are not deleted on the grounds of *column* dependency, but on the grounds of that of their multiplications. We are especially interested in the *weight enumerator* $A(x)$ of a KM (n, k, d) code. Now, without changing $A(x)$ we may convert a generator matrix G into a standard form in two stages: (a) reduce G to its unique Reduced Echelon form by elementary row operations, here switching distinct rows $i, j (R_i \leftrightarrow R_j)$ or adding row j to row $i (R_i \to R_i + R_j)$, and then (b) perform column interchanges $C_i \leftrightarrow C_j$ to obtain $G \approx [I_k|U]$ where the k by $n - k$ matrix U is determined up to the order of its columns; the symbol $\approx$ denotes equivalence under operations (a) and (b). Of course all generator matrices for a given code are equivalent under operations (a). There follows an observation on *extremal cases* $k = 1, N$ for a KM code of minimal length which holds independently of choices allowed by (2)–(5). Let 1^r denote a sequence of r1's.

Theorem 7. *A minimum length KM (n, k, d) code satisfies $n = N$ in extremal cases: (a) $k = 1$: $G = [1^N]$ and $A(x) = 1 + x^N$, and (b) $k = N$: $G \approx [I_N]$ and $A(x) = (1 + x)^N$. Up to equivalence of G we have the following possibilities for $N < 3$.*

$N = 1:$ $k = 1 = d$, $G = [1]$, $A(x) = 1$,

$N = 2:$ $k = 1$, $d = 2$, $G = [11]$, $A(x) = 1 + x^2$; $k = 2$, $d = 1$, $G = I_2$, $A(x) = (1 + x)^2$,

$N = 3:$ $k = 1$, $d = 3$, $G = [111]$, $A(x) = 1 + x^3$; $k = 2$, $d = 2$,
$$G = \begin{bmatrix} 1 & 0 & 1 \\ \\ 0 & 1 & 1 \end{bmatrix}, \ A(x) = 1 + 3x^2, k = 3, \ d = 1, \ G = I_3,$$
$A(x) = (1 + x)^3$.

Proof. (a) we have $d = N$ by Equation 1.5 and hence $Z(u)Y(u) = z_0(y_0 + y_1 u + \ldots + y_{N-1}u^{N-1})$, so the number n of multiplications is exactly N, which equals d. Thus $G = [11\ldots1]$, of length N. In case (b) $Z(u)Y(u) = (z_0 + z_1 u + \ldots + z_{N-1}u^{N-1})y_0$, implying $n = N = k$, $G \approx I_N$. The number of codewords of weight r then equals the coefficient of x^r in $(1 + x)^N$. The rest follows straight from (a), (b) except for case $N = 3$ with $k = d = 2$. These values imply $Z(u)Y(u) = (z_0 + z_1 u)(y_0 + y_1 u)$, so from Example 1 the least possible length is $n = 3$. Hence $G \approx [I_2|U]$ with $U = [ab]^T(a, b \in Z_2)$. But $d = 2$ forces $a = b = 1$.

3.2 Cases with $N = 4$

The extremal cases were described in Theorem 7. We note that they satisfy $n = 4$. There remain, in order of increasing interest, the cases $k = 2, 3$. Both have length $n_{\text{min}} = 5$ according to Theorem 8 below, and are related by shunting. The first has only one outcome for $A(x)$, over all admissable choices ((2)–(5) in Section 3.1). The second has three outcomes; however we shall show that the multiplications can always be selected, for each subcase allowed by varying $P(u)$ and s, so as to give $A(x)$ a preferred "optimal" outcome. Part (a) of the next theorem is proved by adapting the methods of [9].

Theorem 8. *(a) Let $N = 4$. In cases $k = 2$, $d = 3$ and $k = 3$, $d = 2$ we have $n_{min} = 5$. (b) If $k = 2$ then for all admissible s, $P(u)$, its factorisation Equation 1.3, and all valid choices of multiplications giving $n = 5$, the weight enumerator $A(x)$ equals* $1 + 2x^3 + x^4$.

Proof. (b) for any of the allowed choices, G is a 2 by 5 matrix of rank 2 and so may be transformed by operations which do not affect $A(x)$ (see above Theorem 7) to the form $[I_2|U]$, where the rows of U are 3-vectors u, v say. Since the minimum weight is at least three we have that $u \neq v$ and u, v have weights at least two. Thus u, v are distinct members of $\{101, 110, 011, 111\}$. As a result the three nonzero codewords have weights 3,3,4 in some order and $A(x)$ is as stated.

3.3 The $N = 4$ case $k = 3$, $d = 2$

3.3.1 The possibilities for $A(x)$

Given that a generator matrix G is 3 by 5 of rank 3 it may be transformed by operations not affecting $A(x)$ (see above Theorem 7) to the form $[I_3|U]$, where the rows of U are 2-vectors u, v, w which may be permuted amongst themselves without affecting $A(x)$; likewise the two columns of U may be interchanged. Since the least weight is at least two u, v, w are all nonzero. They need not be distinct. If identical they must all be 11, otherwise G has a zero column. Here are representative choices under $\approx$, and their outcomes for $A(x)$ ordered by increasing $A_{w\text{min}}$, the number of codewords of least weight.

$u, v, w = 10, 01, 11$ or $10, 11, 11$	$A_{w\text{min}} = 2$	$A(x) = 1 + 2x^2 + 4x^3 + x^4$
$u, v, w = 10, 10, 11$ or $11, 11, 11$	$A_{w\text{min}} = 3$	$A(x) = 1 + 3x^2 + 3x^3 + x^5$
$u, v, w = 10, 10, 01$	$A_{w\text{min}} = 4$	$A(x) = 1 + 4x^2 + 3x^4.$

For a given w_{min} it is desirable for error correction that the number of codewords $A_{w\text{min}}$ of this weight be as small as possible. We shall call $A(x)$ *optimal* when this minimum is attained for given N, k, d, which occurs in the present case when u, v, w are distinct or exactly two of u, v, w are 11 (see above). We find that, in general, the value of $A_{w\text{min}}$ depends upon the choice of multiplications used, but that in every option of Table 1 to follow *there exists a choice of multiplications for which $A(x)$ is optimal.*

3.3.2 Multiplications for optimal $A(x)$

We wish to achieve the product $Z(u)Y(u) = (z_0 + z_1u + z_2u^2)(y_0 + y_1u)$, and a suitable set of multiplications m_i can be sought directly from the product itself, or built up in steps from the calculations $Z_t(u)Y_t(u)$, and $\overline{Z}(u)\overline{Y}(u)$ mod u^s. Here is a set $m_0, \ldots, m_4$ taken from a look at the product itself, which give optimal $A(x)$, together with a set of alternatives m_i^* such that if exactly one m_i is replaced by m_i^* the optimality remains. The *s carry over to the abbreviating notation Equation 3.1 below, which is *restricted to Section 3.3.*

$$m_i + m_j + \ldots + m_k = m_{ij\ldots k}, \quad m_i = (z\cdot G_i)(y\cdot H_i), \quad G_{ij\ldots k} = G_iG_j\ldots G_k \quad (3.1)$$

$$\begin{array}{ll} m_0 = z_0y_0 & m_0^* = m_1^* = (z_0+z_1+z_2)(y_0+y_1) = m_{01234} \\ m_1 = z_1y_1 & m_3^* = z_2y_0 = m_{03} \\ m_2 = (z_0+z_1)(y_0+y_1) & m_4^* = z_2y_1 = m_{14} \\ m_3 = (z_0+z_2)y_0 & \text{Double star below is a second alternative} \\ m_4 = (z_1+z_2)y_1 & m_3^{**} = m_4^{**} = z_2(y_0+y_1) = m_{0134} \end{array}$$

$Z(u)Y(u)$ is : $m_0 + m_{012}u + m_{013}u^2 + m_4^*u^3$, and

mod $u+b$: m_0 if $b=0$, m_0^* if $b=1$, and

mod $u^2+a_1u+a_0$: $m_0' + (m_0' + m_1' + m_2')u + m_1'(a_1u+a_0)$, where

$m_0' = (z_0 + a_0z_2)y_0 = m_0$ if $a_0 = 0$, else m_3,

$m_1' = (z_1 + a_1z_2)y_1 = m_1$ if $a_1 = 0$, else m_4,

$m_2' = [z_0 + z_1 + (a_0+a_1)z_2](y_0+y_1) = m_2$ if $a_0 = a_1$, else m_0^*.

Note (a) $\overline{Z}(u)\overline{Y}(u) = m_4^* + m_{013}u + m_{012}u^2 + m_0u^3 = \overline{m}_0 + \overline{m}_{012}u + \overline{m}_{013}u^2 + \overline{m_4^*}u^3$, where a *reciprocal multiplication* $\overline{m}$ is obtained from m by $z_0 \leftrightarrow z_2$, $y_0 \leftrightarrow y_1$, (b) calculating ZY mod $u^2+a_1u+a_0$ requires three multiplications except in the special case $u^2+u = u(u+1)$, when two suffice as a consequence of the Chinese Remainder Theorem (Theorem 3), with m_0 for mod u and m_0^* for mod $u+1$.

Example 9. *$[G_{0-4}]$, $[G_0^*\ G_{123}\ G_4^*]$, $[G_{2-4}|G_0|G_0^*]$ are respectively*

$$\begin{bmatrix} 10110 \\ 01101 \\ 00011 \end{bmatrix}, \quad \begin{bmatrix} 10110 \\ 11100 \\ 10011 \end{bmatrix}, \quad \left[\begin{array}{c|c|c} 110 & 1 & 1 \\ 101 & 0 & 1 \\ 011 & 0 & 1 \end{array}\right]$$

Derivation of Table 1.

- Nr 7. Our standard ways to reduce ZY mod Q_2, ZY mod $u+b (b = 0, 1)$, and $\overline{ZY}$ mod u, give the first block matrix below, which is equivalent to the second by the operation $R_2 \to R_2 - a_0R_0 - a_1R_1$.

$$\left[\begin{array}{ccc|c|c} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & b & 0 \\ a_0 & a_1 & a_0+a_1 & b & 1 \end{array}\right] \approx \left[\begin{array}{ccc|c|c} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & b & 0 \\ 0 & 0 & 0 & b+a_0+a_1b & 1 \end{array}\right]$$

But $u + b$ and Q_2 are coprime, implying $Q_2(b) = 1$, i.e. that the sum in the second matrix is 1. Hence after interchange of columns two and four G is in a form which, by the list in Section 3.3.1, gives optimal $A(x)$ for both values of b.

- Nr 8. Both reductions ZY mod $Q_2(\neq u^2+1)$ and $\overline{ZY}$ mod u^2, require three multiplications, so one must be saved to achieve $n = 5$. With ZY using the m' of Section 3.3.2, which equal various m_i according to the identity of Q_2, Table 1 shows how to obtain optimal $A(x)$.

- Nr 9. We have to reduce ZY mod u, $u+1$, and $\overline{ZY}$ mod u^2. Since this can be done with respective multiplication sets m_0, m_0^* and m_1, m_4, m_3^* we are down to $n = 5$ and the blocks are independent. Also G is equivalent to a top line case in the list of Section 3.3.1, so $A(x)$ is optimal.

3.3.3 Non-optimal Weight enumerators

All three logically possible non-optimal $A(x)'s$ do occur. For example we find that $[G_{012}G_3^*G_4^*]$ for Option 1 of Table 1 gives $w_{\min} = 4$, whereas $[G_3^*G_{123}G_4^*]$ gives $w_{\min} = 3$.

3.3.4 Shunting

According to Theorem 5 shunting must work to produce KM (5,2,3) and (5,1,4) codes in each case of Table 1 for which the blocks are independent. This holds for exactly one case of Option 8, for example. Of course $k_{\min} = 3$ or 2 may hold fortuitously, as in the first line of Option 5. Notice that after shunting to $k = 2$, the resulting code must satisfy $A(x) = 1 + 2x^3 + x^4$, by Theorem 8.

Table 1. Generator matrices G for KM $(n,3,2)$ codes for all admissible choices of s and $P_1 \dots P_L$. Multiplications chosen to satisfy (a) the code has minimum length $n = 5$, (b) $A(x)$ is optimal. Q_i stands for an arbitrary polynomial $u^i + \dots$ of degree i, but $Q_3 \neq u^3$. Nr = option number. (I) denotes Independent blocks, $k_{\min}$ is the lowest k to which the code shunts. The wraparound block is last unless otherwise stated. Nrs 1,7,9 exemplify independent blocks

Nr	s L	Admissible Coprime $P_1 \dots P_L$: Block Generator Matrix	$k_{\min}$
1	0 1	Q_4: $[G_{0-3}G_0^*]$ (I)	1
2	0 2	uQ_3: $[G_0 \| G_{1-4}]$, $(u+1)Q_3$: $[G_0^* \| G_{1-4}]$,	2,2
		$u^3(u+1)$: $[G_{0-3} \| G_0^*]$ (I)	1
3	1 1	Q_3: $[G_4^* \| G_{0-3}]$ (wrap first), u^3: $[G_{0-3} \| G_4^*]$ (I)	1,1
4	3 1	u: $[G_0 \| G_{134}G_4^*]$ (I), $u+1$: $[G_0^* \| G_{134}G_4^*]$ (I)	1,1
5	0 2	$u^2(u^2+1)$: $[G_{012} \| G_0^*G_3]$,	
		$(u^2+1)(u^2+u+1)$: $[G_{13}G_0^* \| G_{24}]$	1,2
		$u^2(u^2+u+1)$: $[G_{012} \| G_{34}]$,	
		$(u^2+u)(u^2+u+1)$: $[G_0G_0^* \| G_{2-4}]$ (I)	2,1
6	0 3	$u(u+1)(u^2+u+1)$: $[G_0 \| G_0^* \| G_{234}]$ (I)	1
7	1 2	uQ_2: $[G_0 \| G'_{012} \| G_4^*]$ (I), $(u+1)Q_2$: $[G_0^* \| G'_{012} \| G_4^*]$ (I)	1,1
8	2 1	u^2: $[G_{012} \| G_{34}]$, u^2+1: $[G_{13}G_0^* \| G_{24}]$,	2,2
		u^2+u+1: $[G_{234} \| G_0G_0^*]$, u^2+u: $[G_0G_0^* \| G_{14}G_3^*]$ (I)	3,1
9	2 2	$u(u+1)$: $[G_0 \| G_0^* \| G_{14}G_3^*]$ (I)	1

4 Weight enumerators of shunted families

4.1 The method of duals

Keeping independent blocks, our general method is in three stages:

1. Determine $n-k$ independent relations on the codeword entries $c_0 \dots c_{n-1}$ and hence generators for the dual code.
2. Determine the weight enumerator $B(x)$ of the dual code.
3. Dualise $B(x)$ to obtain the weight enumerator $A(x)$ of the KM code, by using the MacWilliams Identity [3]: *if an (n,q) code has weight enumerator $W(x)$ then its dual has weight enumerator*

$$W^{\perp}(x) = 2^{-q}(1+x)^n W\left(\frac{1-x}{1+x}\right) \tag{4.1}$$

Theorem 10. *(a) The coefficients in $\hat{Z}(u)(= Z(u) \bmod P(u))$ are linear combinations of the columns of zG, and (b) the rank of G_t equals the degree D_t of $P_t(u)$.*

Proof. (a) By hypothesis the multiplications m_j giving block G_t, say $a \le j \le b$, satisfy

$$Z_t(u)Y_t(u) = \sum_{i=0}^{D_t-1}\left(\sum_{j=a}^{b} k_{ij}m_j\right)u^i \bmod P_t(u) \tag{4.2}$$

for certain constants k_{ij}, where $m_j = (zg_j^T)(yh_j^T)$, g_j^T is column j of G, and h_j is a d-vector. In particular, we may set $y_0 = 1$ and $y_i = 0$ otherwise, so that $Y_t(u) = 1$ and

$$Z_t(u) = \sum_{i=0}^{D_t-1}\left(\sum_{j=a}^{b} l_{ij}(zg_j^T)\right)u^i \tag{4.3}$$

where l_{ij} equals k_{ij} times the 0'th entry of h_j. Since such an equality holds for each t, the Chinese Remainder Theorem shows that the coefficients in $\hat{Z}(u)$ are linear combinations of the $zg_j^T(0 \le j \le n-1)$. (b) The D_t coefficients of powers of u in Equation 4.3, necessarily independent, are linear combinations of the columns of zG_t by Equation 4.3, hence $D_t \le$ column rank of G_t. But the latter cannot exceed D_t, since each zg_t^T is a linear combination of the D_t columns of matrix zR, with R as in Equation 2.3. This proves (b).

It is helpful to divide the relations into the three categories below and to view $z_0 \dots z_{k-1}$ as information bits defining a codeword by

$$[z_0 \dots z_{k-1}]G = [c_0 \dots c_{n-1}]. \tag{4.4}$$

4.2 Inner relations

A linear relation between entries c_i is of *inner* type if it corresponds via Equation 4.4 to a relation between columns of the same block (otherwise *outer*). Let block G_t have n_t columns. By Theorem 10 this block has exactly $n_t - D_t$ independent relations between its columns. These reduce the dimension from n to $n - \sum(n_t - D_t) = n - \sum n_t + \sum D_t = n - n + D = D$, leaving $D - k$ relations to find. If $s = 0$ we have $k + d = N + 1 = D + 1$, and so need $(D - k =)d - 1$ more relations. These are supplied by the next type.

4.3 Outer relations

By Theorem 10 and Equation 4.4, *the coefficients $\hat{z}_i$ in $\hat{Z}(u)$ are linear combinations of the entries c_i*. In particular, if $s = 0$ there are expressions for the coefficients z_k to z_{D-1}, which must all equal zero, giving $d - 1$ *outer relations.* These may be obtained by the CRT or perhaps more easily by the matrix method of Equation 4.15.

4.4 Wraparound relations

If $s > 0$ then the outer relations refer to $\hat{z}_i^{\prime}s$ rather than to $z_i^{\prime}s$, and Theorem 11 below is crucial for converting back to $z_i^{\prime}s$. We need a further $n_{L+1} - s$ relations, and this is the number of inner relations in the wraparound block. They are converted to relations among the c_i by equating columns in Equation 4.4.

Theorem 11. *Let $P(u)$ have reciprocal $Q(u) = q_0 + q_1 u + \ldots + q_E u^E$, of degree E. Write $Q_j(x) = q_0 + q_1 x + \ldots + q_j x^j (0 \leq j \leq E)$. Then (a), with coefficients d_i independent of k*

$$\hat{z}_i = z_i, \ \mathit{if}\, 0 \leq i < D - E \ (\mathit{case}\ D > E), \tag{4.5}$$

$$\hat{z}_{D-j} = z_{D-j} + \sum_{i \geq o} d_i z_{D+i} (D + i < k), \ \mathit{if}\, 1 \leq j \leq E, \tag{4.6}$$

where $d_n = q_1 d_{n-1} + q_2 d_{n-2} + \ldots + q_E d_{n-E} (n \geq 0)$ with initial values $d_{-s} = \delta_{sj} (1 \leq s \leq E)$. (b) Let $G_j(x) = \sum_{r \geq 0} d_{r-j} x^r = d_{-j} + d_{-j+1} x + \ldots$. Then $G_j(x) = Q_{j-1}(x)/Q(x)$.

Corollary. (a) $G_1(x) = 1/Q(x)$, $G_E(x) = 1 - x^E G_1(x)$. Hence G_1, G_E have the same period, equal to ord(Q), and d_i (Case $j = E$) $= d_{i-1}$ (Case $j = 1$), for $i \geq 0$. (b) The period L of $G_j(x)$ satisfies, (i) $L|$ ord(Q), (ii) $L >$Max(int$(E/2)$, $j - 1$, $E - j$), where *int* denotes "integer part of", (iii) if $Q(x)$ is irreducible then all $G_j(x)$ have the same period L and $L|(2^E - 1)$, and (iv) for any $1 \leq j \leq E$, if $q_j = 0$ then $G_{j+1} = G_j(x)$.

Proof of Theorem 11. By its definition, $\hat{Z}(u)$ satisfies $Z(u) = \hat{Z}(u)+a(u)P(u)$ for some polynomial $a(u) = \sum a_i u^i$ of degree $k-1-D$. Writing $l(u) = Z(u) - \hat{Z}(u) = \sum l_i u^i$, we cast the relation in matrix form as $[l_0 \dots l_{k-1}] = [a_0 \dots a_{k-D-1}]M$, where the $k-D$ by k matrix M has i'th row $[0^i|p_0 \dots p_D|0 \dots 0]$ (0^i is a sequence of i zeros). An example is the case $k=8$, $D=3$ shown below, where the separator lies between columns $D-1$ and D of M, and zero entries are left blank.

$$[l_0 l_1 \dots l_7] = [a_0 \dots a_4] \left[\begin{array}{ccc|ccccc} p_0 & p_1 & p_2 & p_3 & & & & \\ & p_0 & p_1 & p_2 & p_3 & & & \\ & & p_0 & p_1 & p_2 & p_3 & & \\ & & & p_0 & p_1 & p_2 & p_3 & \\ & & & & p_0 & p_1 & p_2 & p_3 \end{array}\right] \tag{4.7}$$

Now Equation 4.5 follows because the definition of $Q(x)$ implies that the first $D-E$ columns of M are zero. Relation Equation 4.6 will follow from a relation between columns of M. Let $\{d_n\}_{n \geq -E}$ be a sequence of constants. Then a relation of the form

$$x^{-j}(Q(x) - Q_{j-1}(x)) = \sum_{i \geq 0} d_i x^i Q(x) \tag{4.8}$$

must hold, provided the d_i have suitable initial values and recurrence, since the left hand side is a polynomial and $Q(x)$ has constant term $q_0 = 1$ (by definition). But we can take care of the initial value problem by rewriting Equation 4.8 as

$$Q_{j-1}(x) = \sum_{i \geq -E} d_i x^{i+j} Q(x) (d_{-E} \dots d_{-1} \text{ is zero except for } d_{-j} = 1). \tag{4.9}$$

Equating coefficients of $x^{n+j} (n \geq 0)$ we obtain $0 = q_0 d_n + q_1 d_{n-1} + \dots + q_E d_{n-E}$. Invoking again $q_0 = 1$ gives the E-term recurrence relation of the Theorem. Now identify a column $[r_0 r_1 \dots]^T$ of M with the series $\sum_{r \geq 0} r_i x^i$. Then, up to the power $k-D$ of x, column $D-j$ equals the left hand side of Equation 4.8 ($0 \leq j \leq E$) and column $D+i$ equals $x^i Q(x) (i \geq 0)$. This completes the proof of Equation 4.6. Part (b) of the Theorem follows from Equation 4.9 on dividing both sides by $Q(x)$.

4.5 A *KM* $(9, k, 7-k)$ family, and the matrix method

Let $P(u) = u^2(u^2+1)(u^2+u+1)$. Although $k+d = 7$ it is convenient, for finding a neat set of relations, to start with $k=6$, $d=2$ in the product $Z(u)Y(u)$ before writing down the usual matrix G with blocks corresponding to reductions mod $P_i(u)$. Regarding $z = z_0 \dots z_5$ as information bits defining codewords $c = c_0 \dots c_8$, we obtain Equation 4.10.

$$[z_0 \dots z_5] \left[\begin{array}{c|c|c} 1\,0\,1 & 1\,0\,1 & 1\,0\,1 \\ 0\,1\,1 & 0\,1\,1 & 0\,1\,1 \\ 0\,0\,0 & 1\,0\,1 & 1\,1\,0 \\ 0\,0\,0 & 0\,1\,1 & 1\,0\,1 \\ 0\,0\,0 & 1\,0\,1 & 0\,1\,1 \\ 0\,0\,0 & 0\,1\,1 & 1\,1\,0 \end{array}\right] = [c_0 \dots c_8]. \qquad (4.10)$$

- Matrix of blocks for reductions of $Z(u)Y(u)$ mod u^2, u^2+1, u^2+u+1, in case $k = 6$, $d = 2$. Deleting the last row (setting $z_5 = 0$) gives a shuntable KM (9,5,2) code.

The polynomial ZY, of degree six, can be reconstructed via the Chinese Remainder Theorem 3, from the represented multiplications, but only modulo $P(u)$, whose degree is also six. Thus G is not yet the generator matrix of a KM code. However, by adding a preliminary relation $z_5 = 0$ we reduce ZY to degree five and so have a KM (9,5,2) code by Theorem 2. Shunting can now commence, by Theorem 5. Noting that within each block in Equation 4.10 the sum of the columns is zero, we exhibit below a complete set of independent relations for each KM code of the present shunt related family with parameters $(9, k, 7-k)$, $2 \le k \le 5$.

Notation :	$c_i + c_j + _{\dots} + c_k = c_{ij\cdots k}$	(4.11)
Inner relations :	$c_{012} = c_{345} = c_{678} = 0$	(4.12)
Outer relations :	$z_k = \dots = z_5 = 0$	(4.13)

The outer relations in terms of c. In the notation of Example 4 ($D = 2$), $Z_1(u) = a_0 + a_1 u = zg_0^T + (zg_1^T)u$, which equals $c_0 + c_1 u$ from Equation 4.10. Similarly $Z_2(u) = c_3 + c_4 u$ and $Z_3(u) = c_6 + c_7 u$. Now, according to the Chinese Remainder Theorem, the polynomials $Z_i(u)$ determine $Z(u)$ modulo $P(u)$. However this gives $Z(u)$ (unlike ZY) exactly, since its degree is just five, one less than that of $P(u)$. We shall first reconstruct $Z(u)$ by the standard method of the CRT, then show how the approach of Equation 4.15 may be simpler in the present context. For the CRT we must first *compute the auxiliary polynomials* R_i, $1 \le i \le 3$ from $R_i \prod P_j(u)(j \ne i) = 1 \bmod P_i(u)$, then the S_i from $S_i(u) = R_i \prod P_j(u)(j \ne i)$. Some details are given for comparison's sake. We have $R_1 \cdot (u^2+1)(u^2+u+1) = 1 \bmod u^2$, $R_2 \cdot u^2(u^2+u+1) = 1 \bmod (u^2+1)$, and $R_3 \cdot u^2(u^2+1) = 1 \bmod (u^2+u+1)$, implying $R_1 = u+1$, $R_2 = u$, $R_3 = 1$. Then $S_1 = R_1P_2P_3 = (u+1)(u^2+1)(u^2+u+1) = (u^2+1)(u^3+1)$, $S_2 = u^2.u(u^2+u+1)$, $S_3 = u^2(u^2+1)$. In the notation of Equation 4.11, $Z(u) = \sum S_i(u)Z_i(u)$ is

$$= (u^5+u^3+u^2+1)(c_0+c_1u) + (u^5+u^4+u^3)(c_3+c_4u) + (u^4+u^2)(c_6+c_7u)$$

$$= c_0 + c_1u + c_{06}u^2 + c_{0137}u^3 + c_{1346}u^4 + c_{0347}u^5 + c_{14}u^6.$$

Reducing modulo $P(u)$ and equating coefficients,

$$z_0 = c_0, \quad z_1 = c_1 \quad z_2 = c_{0146}, \quad z_3 = c_{0347}, \quad z_4 = c_{1346}, z_5 = c_{0137}. \qquad (4.14)$$

Finally, writing $z_i = \zeta_i c^T (0 \le i \le 5)$ defines the dual code generator ζ_i corresponding to the relation $z_i = 0$. Notice the z's are determined by the $P_i(u)$ rather than the choice of multiplication algorithm (cf. (2.5) and (2.6)).

The matrix method. Let $s = 0$ and $z = [z_0 \dots z_{D-1}]$. By the CRT, Theorem 3, the equation $zG = c$ has a unique solution for z in terms of c. We shall reduce G to an invertible matrix. Now, the columns of $G_t = RS$ (Equations 2.5 and 2.6) are linear combinations of those of R. Since by Theorem 10 the rank of G_t is D_t we may, by rearranging if necessary, take the first D_t columns as independent; the remaining columns are then linear combinations of these. Indeed, for simplicity we shall assume that, as is usually the case, these first D_t columns constitute R itself, and write $\underline{G} = [R_1R_2 \dots R_L]$. Now the equation $z\underline{G} = \underline{c}$ is obtained from $zG = c$ by deleting certain columns of G and their counterparts in c, and so has a unique solution for z in terms of $\underline{c}$. But since $\underline{G}$ is square of order D, uniqueness implies that $\underline{G}$ is invertible, $\underline{z} = \underline{c}\underline{G}^{-1}$. Thus with the same entry deletions indicated by underlining,

$$[\underline{\zeta}_0^T \cdots \underline{\zeta}_{D-1}^T] = \underline{G}^{-1}, \text{ where } z_i = \zeta_i c^T \ (0 \le i \le D-1). \qquad (4.15)$$

Periodicity of R and G_t. The following analysis for $L = 1$, $P_t(u) = P(u)$ (not divisible by u), applies to each block in case $L > 1$. By Equation 2.2, row i of R is the vector $[r_{i0} \dots r_{i,D-1}]$, where $u^i \bmod P(u) = \sum r_{ij}u^j (0 \le j \le D-1)$. Hence row q = row $r \Leftrightarrow = u^q = u^r \bmod P(u) \Leftrightarrow e|(r-q)$, where $e = e(P(u))$ is both (a) the *order of u mod $P(u)$*, the least $m \ge 1$ such that $u^m = 1 \bmod P(u)$, and (b) the *order of $P(u)$*, namely the least m such that $P(u)|(u^m - 1)$. In particular, the rows of R repeat, so far as they are present, *with period $e \le 2^D - 1$*, and hence so do the rows of $G_t = RS$. This periodicity may be seen in Example 4 and in the last two blocks of Equation 4.10.

The case $P(u)$ is irreducible. Now e is the order of the element α representing polynomial u in the field $Z_2[u]/(P(u))$, of finite order 2^D, and in this field the i'th row of R gives the expression of α^i as a polynomial in α of degree $D-1$,

$$\alpha^i = r_{i0} + r_{i1}\alpha + \ldots + r_{i,D-1}\alpha^{D-1}. \tag{4.16}$$

Using the matrix method. The method is useful when inverting $\underline{G}$ is easier than the CRT calculation. A type of code fairly easily handled, which fits nicely with the conditions for shunting (Theorem 5), and includes the present case, is given by $D_t = D/L(1 \le t \le L) = m$, say. We may take as coprime factors of $P(u) : P_1(u) = u^m$, $P_2(u) = u^m + 1$, and any or all the irreducibles of degree m. Note $e(u^m + 1) = m$, $e(u^2 + u + 1) = 3$.

Example 12. *Write I for I_m, where m is any positive integer. For $L = 2$ we have uniquely $\underline{G} = \begin{bmatrix} I & I \\ 0 & I \end{bmatrix} = \underline{G}^{-1}$. In case $L = 3$ the first matrix below is $[\underline{G}|I_{3m}]$, a 3×6 block matrix of $m \times m$ sized blocks. Let R_i denote block row i. We perform "higher" row operations which convert $\underline{G}$ to I_{3m} and hence I_{3m} to $\underline{G}^{-1}$. Firstly $R_1 \to R_1 - R_2$, $R_3 \to R_3 - R_2$ gives the second matrix below (remember we are working mod 2). Since $\underline{G}^{-1}$ exists, so does $C = (A+B)^{-1}$. It remains to perform $R_3 \to CR_3$, $R_1 \to R_1 - (I+A)R_3$, $R_2 \to R_2 - AR_3$, and we have $\underline{G}^{-1}$ as shown. The matrix $\underline{G}^{-1}$ of Equation 4.18 is the result applied to $KM\ (9, k, 7-k)$, for which it is trivial to compute that $A = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$, $A+B = I$.*

$$\left[\begin{array}{ccc|ccc} I & I & I & I & 0 & 0 \\ 0 & I & A & 0 & I & 0 \\ 0 & I & B & 0 & 0 & I \end{array}\right] \to \left[\begin{array}{ccc|ccc} I & 0 & I+A & I & I & 0 \\ 0 & I & A & 0 & I & 0 \\ 0 & 0 & A+B & 0 & I & I \end{array}\right]$$

$$\underline{G}^{-1} = \begin{bmatrix} I & (I+B)C & (I+A)C \\ 0 & BC & AC \\ 0 & C & C \end{bmatrix}, C = (A+B)^{-1}. \tag{4.17}$$

$$For\ P(u) = u^2(u^2+1)(u^2+u+1):$$

$$\underline{G}^{-1} = \begin{bmatrix} I & A & B \\ 0 & B & A \\ 0 & I & I \end{bmatrix}, \quad \begin{array}{lcl} \zeta_2 & = & 110\ \ 010\ \ 100 \\ \zeta_3 & = & 100\ \ 110\ \ 010 \\ \zeta_4 & = & 010\ \ 110\ \ 100 \\ \zeta_5 & = & 110\ \ 100\ \ 010 \end{array} \tag{4.18}$$

Notation A. (Calculating $A(x)$). Let U be the space of n-vectors over Z_2, viewed as sequences of three triples $u = u_1u_2u_3$ followed by $n - 9$ single digits. We define the following, where V is any subset of U.

E	The span $Sp(E_1, E_2, E_3)$ where E_i is zero except for i'th triple 111.
$u + V$	The *translate* $\{u + v : v \in V\}$ of V by u.
S_k	$Sp(\zeta_k, \zeta_{k+1}, \ldots, \zeta_5)$ as subspace of $U (0 \le k \le 5)$, defined as $\{0\}$ if $k > 5$.
$t(u)$	The number of nonzero triples of $u \in S_k$. Thus $t(0) = 0$.
$R(V)$	The weight enumerator $\sum x^{wt(v)} (v \in V)$ of V.

Table 2. Weight enumerators $A(x)$ of the shunt related KM $(9, k, 7-k)$ codes, $2 \leq k \leq 5$, based on: $t(u) = 3$ for $u = \zeta_k$, $\zeta_2 + \zeta_3$, $\zeta_4 + \zeta_5$ otherwise $t(u) = 2$ for u in S_k

k	d	α_0's	α_2's	α_3's	$A(x)$	$b(x)$: $B(x) = b(x) + x^5 \bar{b}(x)$
5	2	1	0	1	$1 + 3x^2 + 15x^4 + 13x^6$	$1 + 4x^3 + 3x^4$
4	3	1	0	3	$1 + 9x^4 + 6x^6$	$1 + 6x^3 + 9x^4$
3	4	1	3	4	$1 + 3x^4 + 4x^6$	$1 + 3x^2 + 13x^3 + 15x^4$
2	5	1	9	6	$1 + 3x^6$	$1 + 9x^2 + 27x^3 + 27x^4$

From Equation 4.12 and Equation 4.13 the dual code is generated by E_1, E_2, E_3, $\zeta_k, \ldots, \zeta_5$, or

$$\text{Dual } KM\ (9, k, 7-k) \text{ code} = \cup(u + E)(u \in S_k). \tag{4.19}$$

Lemma 13. $R(u + E) = (x + x^2)^t (1 + x^3)^{3-t}$, *where* $t = t(u)$.

Proof. Let $V_i = \{u_i, u_i + E_i\}$. Then $u + E$ equals the Cartesian product $V_1 \times V_2 \times V_3$ and $R(u + E) = R(V_1)R(V_2)R(V_3)$. Now, by construction of the ζ's the third digit of each triple is 0, so u_i is one of 000,100,010,110. Hence $R(V_i) = 1 + x^3$ if $u_i = 000$, otherwise $x + x^2$, and $R(u + V)$ is as stated.

This Lemma gives the dual code's weight enumerator $B(x)$, which we dualise back to $A(x)$ by the MacWilliams Identity Equation 4.1, as shown below. The result is Table 2.

$$\alpha_t = (1-y)^t (1+3y)^{3-t}\ (y = x^2), \quad \beta_t = (x + x^2)^t (1 + x^3)^{3-t}, \tag{4.20}$$

$$A(x) = 2^{k-6} \sum_u \alpha_{t(u)}, \quad B(x) = \sum_u \beta_{t(u)}\ (u \in S_k). \tag{4.21}$$

4.6 A KM $(10, k, 8-k)$ family with wraparound $s = 1$

With $P(u)$ as in Equation 4.18 we now incorporate wraparound $s = 1$, so that $N = D + s = 7$ and $k + d = N + 1 = 8$. The shunted family have parameters $(10, k, 8-k)$, with $2 \leq k$, $d \leq 6$, and generator matrix represented in Equation 4.22 below by case $k = 4$ (see also Equation 1.1). The last column ends in a 1, whatever the value of k.

$$[z_0 \ldots z_{k-1}] \left[\begin{array}{c|c|c|c} 1\,0\,1 & 1\,0\,1 & 1\,0\,1 & 0 \\ 0\,1\,1 & 0\,1\,1 & 0\,1\,1 & 0 \\ 0\,0\,0 & 1\,0\,1 & 1\,1\,0 & 0 \\ 0\,0\,0 & 0\,1\,1 & 1\,0\,1 & 1 \end{array}\right] = [c_0 \ldots c_9] \tag{4.22}$$

- Generator matrix for $KM\ (10, k, 8-k)$ family represented by case $k = 4$. The last column, from wraparound $s = 1$, has a 1 in its last row for all k.

Some notation. Equating column 9 of the two sides in Equation 4.22 we have a new kind of equation, $z_{k-1} = c_9$. The corresponding dual generator is $\zeta'_{k-1} = \zeta_{k-1} + e_9$, where e_i is the vector with only position i nonzero, in the space U of Notation A. The ζ_i of the previous section are extended by one zero, leaving values of t unchanged. We can now give the necessary $10 - k$ independent generators for the dual code in the form Equation 4.23 and, with β_t as in Equation 4.20, the weight enumerators (Equation 4.24) of the translates of E which make up this code.

$$\text{Dual } KM\ (10, k, 8-k) \text{ code} = \bigcup_u (u + E)(u \in Sp(\zeta'_{k-1}, S_k)) \tag{4.23}$$

$$R(u + E) = \beta_{t(u)} \text{ if } u \in S_k, \text{ but } x\beta_{t(u)} \text{ if } u \in \zeta'_{k-1} + S_k. \tag{4.24}$$

As before, we apply the MacWilliams Identity Equation 4.1. Changing the code length to 10 adds an extra factor $1 + x$ to the α_t of Equation 4.20, and multiplying β_t by x replaces the $1 + x$ by $1 - x$, resulting in say α'_t, which equals $((1-x)/(1+x))\alpha_t$. As an aid to calculation we give the α's in the form Equation 4.25 below. The result is Table 3. We omit $B(x)$; it is easily recovered from Equation 4.26 by removing the factor 2^{k-7} and replacing the symbols α by β but α' by $x\beta$.

$$\begin{aligned} \alpha_t &= \textstyle\sum_{r=0}^{3} \left(b_{tr}x^{2r} + b_{tr}x^{2r+1}\right) \\ \alpha'_t &= \textstyle\sum_{r=0}^{3} \left(b_{tr}x^{2r} - b_{tr}x^{2r+1}\right) \end{aligned} \quad \text{where } [b_{tr}] = \begin{bmatrix} 1 & 9 & 27 & 27 \\ 1 & 5 & 3 & -9 \\ 1 & 1 & -5 & 3 \\ 1 & -3 & 3 & -1 \end{bmatrix} \tag{4.25}$$

$$A(x) = 2^{k-7}\left\{\sum_u \alpha_{t(u)}(u \in S_k) + \sum_v \alpha'_{t(v)}(v \in \zeta'_{k-1} + S_k)\right\} \tag{4.26}$$

Table 3. Weight enumerators of the KM $(10, k, 8-k)$ family with coefficients a_i, a'_i of α_i, α'_i determined by counting separately the vectors u, v of Equation 4.26 with each value of t. Note: $a_0 = 1$ always, and $a_1 = 0$ here, for $2 \le k \le 6$

k	d	$Sp(\zeta'_{k-1}, S_k)$	a'_1	a_2	a'_2	a_3	a'_3	$A(x)$
6	2	0, $\zeta_5{}'$	0	0	0	0	1	$1+3x^2+6x^3+15x^4+12x^5$ $+13x^6+14x^7$
5	3	S_5, $\zeta_4{}'+S_5$	0	0	0	1	2	$1+3x^3+9x^4+6x^5+6x^6+7x^7$
4	4	S_4, $\zeta_3{}'+S_4$	0	0	3	3	1	$1+3x^4+6x^5+4x^6+2x^7$
3	5	S_3, $\zeta_2{}'+S_3$	0	3	6	4	2	$1+3x^5+3x^6+x^7$
2	6	S_2, $\zeta_1{}'+S_2$	3	9	6	6	7	$1+x^6+2x^7$

Can we get explicit formulae for the A_i? We would like both to bypass the actual counting of the number of times $t = 2$ and $t = 3$ occur, and to find a systematic way of obtaining formulae $a_i(d)$, $a'_i(d)$, and especially $A_i(d)$, expressing everything in terms of d (or equally, k). As a first step we deduce a compact expression for the A_i from Equation 4.25, and Equation 4.26, replacing 2^{k-7} by 2^{1-d} in the latter

$$\begin{aligned} 2^{d-1}A_{2r} &= b_{0,r} + b_{2,r}(a_2 + a'_2) + b_{3,r}(a_3 + a'_3) \quad (d \ge 2), \\ 2^{d-1}A_{2r+1} &= b_{0,r} + b_{2,r}(a_2 - a'_2) + b_{3,r}(a_3 - a'_3) \quad (d \ge 2). \end{aligned} \tag{4.27}$$

Without knowing the actual distribution of t values, we may infer from the construction of the $Sp(\ldots)$ column of Table 3, and the size $|S_k| = 2^{d-2} (d \ge 2)$, that

$$a_2(d+1) = a_2(d) + a'_2(d), \quad a_3(d+1) = a_3(d) + a'_3(d) \quad (2 \le d \le 5), \tag{4.28}$$

$$a_2 + a_3 = 2^{d-2} - 1, \quad a'_2 + a'_3 = 2^{d-2} \quad (2 \le d \le 5). \tag{4.29}$$

The a_i and a'_i are trivial to determine for $d = 2$, so we may conveniently use them as initial values for Equation 4.28. The hypothesis $d \ge 2$ implies that $A_0 = 1$, $A_1 = 0$ which, by Equation 4.25, Equation 4.27 are together equivalent to Equation 4.29; furthermore for $d \ge 3$ we have $A_2 = 0$, which translates to the first equation of Equation 4.30 below; the second is obtained by adding the two

Table 4. Coefficients in the weight enumerators of the KM $(10, k, d)$ family, as functions of the parameter $d = 8 - k$. Ditto marks are shown thus "

k	d	A_2	A_3	A_4	A_5	A_6	A_7
6	2	3	6	15	12	13	14
5	3	0	3	$3(2^{5-d}-1)$	6	$2^{5-d}+2$	7
4	4	0	0	"	3.2^{5-d}	"	2^{5-d}
3	5	0	0	"	"	"	"
2	6	0	0	0	0	1	2

equations of Equation 4.29. Solving Equation 4.30 as simultaneous equations in two variables $a_2 + a_2'$ and $a_3 + a_3'$ yields Equation 4.31.

$$(a_2 + a_2') - 3(a_3 + a_3') = -9(d \geq 3), \quad (a_2 + a_2') + (a_3 + a_3') = 2^{d-1} - 1 \quad (d \geq 2) \tag{4.30}$$

$$a_2 + a_2' = 3(2^{d-3} - 1)(d \geq 3), \quad a_3 + a_3' = 2^{d-3} + 2 \quad (3 \leq d \leq 5). \tag{4.31}$$

Now we simply substitute Equation 4.31 into Equation 4.28 to get the first two explicit expressions in Equation 4.32 for a_2, a_3, then substitute these back into Equation 4.31 to obtain a_2', a_3'. Finally we may use these values with Equation 4.25 to compute Table 4. Notice that the general formula for A_{2r} holds for $d \geq 3$ rather than just $d \geq 4$ because it may be derived from Equation 4.25 using only Equation 4.31.

$$a_2 = 3(2^{d-4} - 1), \quad a_3 = 2^{d-4} + 2, \quad a_2' = 3.2^{d-4}, \quad a_3' = 2^{d-4}(d \geq 4) \tag{4.32}$$

Essentially we have formulae in the range between impractically small k or d (see Table 4).

Conjecture. For each N, k, d and $P(u)$, the shortest KM code and the least $A_{w\text{min}}$ can be achieved simultaneously by suitable choice of multiplications (depending on the factorisation of $P(u)$).

4.7 Objectives

Give formulae for the A_i in terms of parameters for general KM codes. Show how to obtain the "best" codes for a given purpose. Determine the "best" value of s to use. Determine the precise way in which a code's properties depend on the choice of $P(u)$, its factorisation, and the multiplications chosen.

On the other hand we would like to produce results which are independent of the multiplication algorithms chosen, following on for example from Equations 2.5 and 4.15.

References

1. Mandelbaum, D.M. (1974). Adaptive-feedback coding scheme using incremental redundancy. *IEEE Transactions on Information Theory*, **IT-20**, 388–389.
2. Hill, R. (1988). *A First Course in Coding Theory*, Oxford University Press.
3. MacWilliams, F.J. and Sloane, N.J.A. (1977). *The Theory of Error-Correcting Codes*, North-Holland, Amsterdam.
4. Krishna, H. (1987). *Computational Complexity of Bilinear Forms*, *Lecture Notes in Control and Information Sciences*, **94**, Springer-Verlag.
5. Krishna, H. (1993). On a signal processing algorithms based class of linear codes. *AAECC*, **4**, 41–57.
6. Krishna, H. and Morgera, S.D. (1987). A new error control scheme for hybrid ARQ systems. *IEEE Transactions Communications*, **COM-35**, 981–990.
7. McFarlane, I. (1991). Generalised type-II hybrid automatic repeat request schemes and *KM* codes. *MSc Thesis*, Glasgow University.
8. Lempel, A. and Winograd, S. (1977). A new approach to error-correcting codes. *IEEE Transactions Information Theory*, **IT-23**, 503–508.
9. Winograd, S. (1970). On the number of multiplications necessary to compute certain functions. *Comm. Pure and Applied Maths.*, **23**, 165–179.

Densities of Perfect Linear Complexity Profile Binary Sequences

A.E.D. Houston[1]

Royal Holloway, University of London

Abstract

Binary Perfect Linear Complexity Profile sequences (PLCPs) have the profile closest to the expected linear complexity of random binary sequences. In this paper we investigate the proportion of ones in finite length PLCPs, showing that they do not in general have the balance property of random sequences. The PLCP with uniformly least density is given, as well as evidence to support the conjecture that the limit of the maximum density, as the length tends to infinity, is 2/3.

1 Introduction

In a stream cipher, the data sequence in bits is added modulo 2 to a keystream before transmission. At the receiver, the same keystream is added to the received ciphertext to regain the message. The keystream must be deterministic (so it can be reproduced at both ends of the communication channel), but for security it is recommended that it have pseudo–random properties (see [1]) so that an eavesdropper, even if he knows some of the keystream sequence, cannot predict any of the rest. Although it is not easy to quantify randomness, the following have been suggested in [2] and [3] as pseudo–random properties;

- a balance of 0's and 1's in a period,
- ideal run frequencies in a period,
- low out-of-phase auto-correlation, and
- linear complexity close to half the length of the sequence.

Perfect Linear Complexity Profile sequences (PLCPs) satisfy the last condition exactly. In this paper we shall investigate to what degree PLCPs satisfy the first pseudo–random property, i.e. we look at the proportion of ones (the density) in finite length PLCPs. After giving some basic definitions and relevant results, we shall look in turn at some specific PLCPs and their densities, then at the

[1]This research is supported by the EPSRC Case award number: 92500137, in conjunction with Vodafone Limited.

maximum and minimum densities over all PLCPs for particular lengths, and the limits of these as the length tends to infinity. The PLCP with the uniformly least density for all lengths is given, as well as evidence to support the conjecture that the limit of the maximum density is 2/3.

1.1 Density

Definition 1. *The* density $D(n)$ *of a binary sequence* $s_1, s_2, \dots, s_n$ *of length* n *is the number of ones in the sequence, divided by* n*, i.e.*

$$D(n) = \frac{\sum_{i=1}^{n} s_i}{n}.$$

Clearly,

$$0 \le D(n) \le 1, \text{ for all } n.$$

The density measures the proportion of ones in a finite sequence, and hence indicates how balanced the sequence is. The expected density for a purely random sequence is 1/2, for any length, hence a balance of 0's and 1's is desirable in a pseudo–random keystream.

Example 2. *The sequence 1100101.... has* $D(1) = 1$*,* $D(2) = 1$*,* $D(3) = 2/3$*,* $D(4) = 1/2$*,* $D(5) = 3/5$ *etc.*

1.2 Linear complexity profiles

Definition 3. *Let* $s_1, s_2, \dots, s_n$ *be an arbitrary finite binary sequence. Then the* linear complexity $L(n)$ *of the sequence is the length of the shortest linear feedback shift register that can generate* $s_1, s_2, \dots, s_n$.

We write L as a function of n to note the length of the sequence involved. Explicitly, it is the least k such that there exist binary integers $c_0, \dots, c_{k-1}$ satisfying:

$$s_{i+k} = c_0 s_i \oplus c_1 s_{i+1} \oplus \dots \oplus c_{k-1} s_{i+k-1}$$

for $i = 1, \dots, n-k$, where $\oplus$ denotes addition modulo 2. The shift register is uniquely defined by the above recursion coefficients $c_0, \dots, c_{k-1}$, and the sequence is uniquely defined by the c_i and the initial values $s_1, \dots, s_k$. The integer k is the length of the shift register, and we shall use the convention that the zero sequence 000... is generated by a shift register of length 0.

Definition 4. *The* Linear Complexity Profile *of the (possibly infinite) sequence* $s_1, s_2, \dots$ *is the corresponding sequence* $L(1), L(2), \dots$*, i.e. the sequence of values the linear complexity takes as the binary sequence gets longer and longer.*

This is most clearly pictured as a monotone increasing step–graph of $L(n)$ against n. In [4], Massey shows that the iterative algorithm introduced by Berlekamp in [5] for decoding BCH codes provides a general solution to the problem of finding the linear complexity profile of a prescribed finite sequence. Informally, the algorithm shows that the linear complexity as another bit is added to the sequence either remains the same, or jumps up to an equal distance on the other side of the $y = x/2$ line on the graph, i.e. $L(n+1) = L(n)$, or $n + 1 - L(n)$.

1.3 Perfect Linear Complexity Profile Sequences (PLCPs)

A purely random sequence of n bits would be expected to have linear complexity of approximately $n/2$. Explicitly, Rueppel has shown in [3] that the expected value of $L(n)$, for large n, is $\frac{n}{2} + c_n$, with $0 \leq c_n \leq \frac{5}{18}$. The profile which follows as closely as possible the $y = x/2$ line, see Figure 1, is thus called the **perfect** profile, and is given explicitly by;

$$L(n) = \lfloor \frac{n+1}{2} \rfloor \text{ for } 1 \leq n < \infty.$$

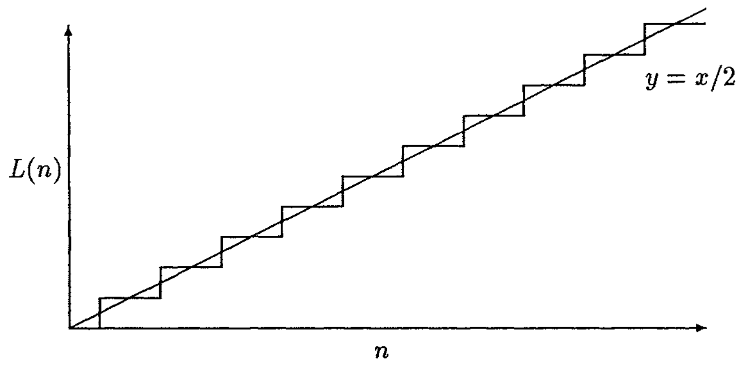


Figure 1. The Perfect Linear Complexity Profile

Rueppel suggested a candidate PLCP sequence in 1984, but it was not proven to have a perfect profile until 1986, by Dai in [6]. Massey and Wang were the first to characterize all PLCP sequences in terms of a simple linear recurrence, see [7]. Their result is stated in the following theorem:

Theorem 5. (Massey-Wang) *A binary sequence of length n has PLCP if and only if $s_1 = 1$ and for $i \geq 1$*

$$s_{2i+1} = s_{2i} \oplus s_i.$$

Thus the even positions in the sequence can be arbitrary, but the odd positions are dependent on the previous even position and the bit half way back along the sequence. This also shows that although the PLCP sequences have linear complexity profile close to that of random sequences, they are far from unpredictable. Since there are two choices for each even bit, we can count the number of PLCP sequences of length n to be $2^{\lfloor n/2 \rfloor}$.

1.4 Examples

1.4.1 Generalised Rueppel Sequences (GRS)

Defined in [8], as their name suggests, this family of PLCPs contains the Rueppel sequence. Let the binary sequence $c_1, c_2, \ldots$ determine the monotonic increasing sequence of integers $n_1, n_2 \ldots$ by:

$$n_1 = 1, \text{ and } n_{j+1} = 2n_j + c_j.$$

This sequence in turn determines the positions of the ones in the corresponding GRS, i.e.

$$s_i = 1 \Leftrightarrow i = n_j \text{ for some } j.$$

Two simple examples of Generalised Rueppel sequences are:

- $c_i = 0$ for all i—this gives the Rueppel sequence $11010001 0^7 1 0^{15} 1 \ldots$, i.e. $r_1, r_2, \ldots$, where
$$r_i = 1 \Leftrightarrow i = 2^j \text{ for } j \geq 0.$$
This description is also equivalent to the relation $r_{2i} = r_i$ with $r_{2i+1} = r_{2i} \oplus r_i$ from the Massey–Wang characterization.

- $c_i = 1$ for all i—this is the Morii–Kasahara (M-K) sequence, see [9], which is just the Rueppel sequence with the first bit removed,
i.e. $1010001 0^7 100 \ldots$ etc., or $m_1, m_2, \ldots$, where
$$m_i = 1 \Leftrightarrow i = 2^j - 1 \text{ for } j \geq 0.$$
It also satisfies $m_{2i} = 0$ for all i, along with the Massey–Wang relation.

Having seen sequences which satisfy $s_{2i} = 0$ and $s_{2i} = s_i$ for all i, it is natural to look at their "complements" $s_{2i} = 1$ and $s_{2i} = \overline{s_i}$, for all i.

Definition 6. *The* Type I *and* Type II *sequences are the PLCPs defined by $s_{2i} = 1$ and $s_{2i} = \overline{s_i}$, for all i, respectively.*

1.4.2 The Type I sequence

Since $s_{2i} = 1$, for all i, we have from Theorem 5 that

$$s_{2i+1} = 1 \oplus s_i.$$

This leads us to two cases:

- i is even or 1, so $s_i = 1$ which implies $s_{2i+1} = 1 \oplus 1 = 0$;
- i is odd, say $i = 2j+1$, which implies $s_{2i+1} = 1 \oplus s_{2j+1} = 1 \oplus 1 \oplus s_j = s_j$, which leads us to two subcases:
  - * j is even or 1, which implies $s_{2i+1} = 1$;
  - * j is odd, say $j = 2k+1$, which implies $s_{2i+1} = 1 \oplus s_k$, which leads us to two subcases:
    - ∘ k is even or 1, which implies $s_{2i+1} = 0$;
    - ∘ k is odd, say $k = 2l+1$, which implies $s_{2i+1} = s_l$, etc....

Continuing to break down $2i+1$ in this way, we obtain a sequence $n, i, j, k, l, \ldots$ of odd numbers which terminates in an even number or a 1.

Lemma 7. *The Type I sequence has zeros at the positions s_n if and only if*

$$\begin{array}{lcll} n & = & 2^{2K} - 1, & or \\ n & \equiv & 2^{2K-1} - 1 & mod\ 2^{2K}\ for\ n > 2^{2K} \quad for\ some\ K \geq 1 \end{array}$$

Proof. From the above discussion, $s_{2i+1} = 0$ if there is an odd number of odd numbers in the sequence $n, i, j, k, l, \ldots$ before the even or 1. If there is an even number of odds, then $s_{2i+1} = 1$. To find the general form of n such that $s_n = 0$, we have two cases:

1. $n = \overbrace{2(2(2(\cdots(2}^{\text{odd no.}}(2X)+1)\cdots)+1)+1)+1$, for some $X \geq 1$, or
2. $n = \overbrace{2(2(2(\cdots(2}^{\text{odd no.}}(1)+1)\cdots)+1)+1)+1$,

where, in both cases there are an odd number of 2s before the central bracket. Multiplying the brackets out, letting the odd number of 2s be $2K-1$, we get

1. $n = 2^{2K}X + 2^{2K-2} + \cdots + 2 + 1 = 2^{2K}X + 2^{2K-1} - 1$, or
2. $n = 2^{2K-1} + 2^{2K-2} + \cdots + 2 + 1 = 2^{2K} - 1$,

which give us the formulae stated in the Lemma 7 for the positions of zeros.

1.4.3 The Type II sequence

Lemma 8. *The Type II sequence has zeros at the positions s_n if and only if*

$$n \equiv 2^{2K-1} \mod 2^{2K} \text{ for some } K \geq 1$$

Proof. This is a slightly simpler version of the above proof for Type I sequences, so is omitted.

2 Densities of PLCPs

2.1 Results for the Rueppel, M-K, Type I and II sequences

The density, $D_R(n)$, of the Rueppel sequence is easy since we know exactly where the ones are:

$$D_R(n) = \frac{\lfloor \log_2 n \rfloor + 1}{n}.$$

Similarly for the M-K sequence:

$$D_M(n) = \frac{\lfloor \log_2 (n+1) \rfloor}{n}.$$

Since the Rueppel sequence has ones most often, and the M-K sequence has ones least often out of all Generalised Rueppel sequences, we have that

$$D_M(n) \leq D_{GRS}(n) \leq D_R(n), \text{ for all } n,$$

but clearly the exact value depends on the sequence $\{c_j\}$.

For the Type I and II sequences it is a little more complicated, but if we consider splitting the sequences into blocks of size 2^{2K}, for each fixed K, we see that each congruence in Lemmas 7 and 8 gives us exactly one zero per block. It is easy to check that the congruences do not overlap, i.e. some n cannot be a solution for two different values of K. Thus, for example, we get one zero in every block of 4 from $K = 1$, another zero in every block of 16 from $K = 2$ etc. If we cut the sequence off at $n = 2^{2K}$, it follows that the density of *zeros* is

$$\begin{aligned} D^0_{I,II}(2^{2K}) &= \frac{1}{4} + \frac{1}{16} + \frac{1}{64} + \ldots + \frac{1}{2^{2K}} \\ &= \frac{1}{3}\left(1 - \left(\frac{1}{4}\right)^K\right), \end{aligned}$$

so the density of *ones* is given by:

Lemma 9. *The density for a Type I or Type II sequence of length 2^{2K} is*

$$D_{I,II}(2^{2K}) = \frac{2}{3} + \frac{1}{3}\left(\frac{1}{4}\right)^K.$$

Proof. From $D(n) = 1 - D^0(n)$.

Corollary. The number of zeros in a Type I or II sequence of length 2^{2K}, $N^0(2^{2K})$, is given by:

$$N^0(2^{2K}) = \frac{1}{3}\left(2^{2K} - 1\right).$$

Proof. Multiply the density in Lemma 9 by 2^{2K}.

We can use this result to work out the densities for any length n, by expressing n in its base 4 expansion, i.e. $n = \sum_{i=0}^{l} a_i 4^i$, for $a_i \in \{0, 1, 2, 3\}$, and counting the number of zeros in the a_i complete blocks of length 4^i, for each i.

For the Type II sequence, a block of length 2^{2i} contains 4 full blocks of length 2^{2i-2}, with all their zeros, as well as its contribution of one zero half-way along at 2^{2i-1}. Hence;

$$N^0(2.2^{2i-2}) = 1 + 2.N^0(2^{2i-2}) = 1 + \frac{2}{3}(2^{2i-2} - 1) = \frac{1}{3}(1 + 2^{2i-1}),$$

and

$$N^0(3.2^{2i-2}) = 1 + 3.N^0(2^{2i-2}) = 1 + (2^{2i-2} - 1) = 2^{2i-2}.$$

Define the function $S_{a_i}(i)$, for $i \geq 0$, by:

$$S_{a_i}(i) = \begin{cases} 0 & \text{if} \quad a_i = 0, \\ \frac{1}{3}(4^i - 1) & \text{if} \quad a_i = 1, \\ \frac{1}{3}(\frac{4^{i+1}}{2} + 1) & \text{if} \quad a_i = 2, \\ 4^i & \text{if} \quad a_i = 3. \end{cases}$$

Then $S_{a_i}(i)$ counts the number of zeros contributed by the a_i complete blocks of length 4^i. For example, when $i = 1$, we are looking at complete blocks of length 4. If $a_1 = 1$, we know there is only one zero in the middle of that block. If $a_1 = 2$, we know there are 3 zeros—2 from the centres of the 4–blocks and one contributed by the mod 16 congruence. Finally, if $a_1 = 3$, there are 4 zeros. Checking these values with $S_{a_1}(1)$:

$$S_{a_1}(1) = \begin{cases} 0 & \text{if} \quad a_1 = 0, \\ \frac{1}{3}(4^1 - 1) = 1 & \text{if} \quad a_1 = 1, \\ \frac{1}{3}(\frac{4^{1+1}}{2} + 1) = 3 & \text{if} \quad a_1 = 2, \\ 4^1 = 4 & \text{if} \quad a_1 = 3. \end{cases}$$

Similarly with $i = 0$, in a block of 4 we are counting the units. The only zero is at the second unit, so the cumulative number of zeros is 0,0,1,1 when a_0 is 0,1,2,and 3 respectively. This also agrees with $S_{a_0}(0)$.

Theorem 10. *Let $n = \sum_{i=0}^{l} a_i 4^i$, for $a_i \in \{0,1,2,3\}$, then the density of a Type II sequence of length n is given by:*

$$D_{II}(n) = \frac{2}{3} - \frac{1}{3n}\left(\#\{i : a_i = 2\} - \#\{i : a_i = 1\}\right)$$

where $\#\{i : a_i = k\}$ denotes the number of the coefficients equal to k.

Proof. From the above discussion, we now have the complete number of zeros:

$$N^0(n) = \sum_{i=0}^{l} S_{a_i}(i).$$

So

$$D_{II}(n) = \frac{\sum_{i=0}^{l} a_i 4^i - \sum_{i=0}^{l} S_{a_i}(i)}{n},$$

since we require the number of ones divided by the length. Note that all the expressions for $S_{a_i}(i)$ contain a term $a_i 4^i$, in fact

$$S_{a_i}(i) = \frac{1}{3}(a_i 4^i + T_{a_i}),$$

where

$$T_{a_i} = \begin{cases} 0 & \text{if} \quad a_i = 0, \\ -1 & \text{if} \quad a_i = 1, \\ 1 & \text{if} \quad a_i = 2, \\ 0 & \text{if} \quad a_i = 3. \end{cases}$$

Thus

$$\begin{aligned} D_{II}(n) &= \frac{1}{n}\left(n - \frac{1}{3}n - \frac{1}{3}\sum_{i=0}^{l} T_{a_i}\right) \\ &= \frac{2}{3} - \frac{1}{3n}\sum_{i=0}^{l} T_{a_i}, \end{aligned}$$

which gives the result stated in the theorem, since all T does is add up the number of a_i's equal to 2, and subtract the number if a_i's equal to 1.

Example 11. *$n = 29 = 16 + 3.4 + 1$, so $a_2 = 1$, $a_1 = 3$, and $a_0 = 1$. This gives us the result that the total number of zeros is $\frac{1}{3}(16 - 1) + 4 + 0 = 9$. We can check this from the definition of the positions of zeros—we have zeros at bits 2, 6, 10, 14, 18, 22, 26, 30... etc from the modulo 4 condition; bits 8, 24, 40... etc. from the modulo 16 condition; bits 32, 96... etc. from mod 64; etc. This shows the 9 zeros in length 29.*

For the Type I sequence, let $n = \sum_{i=0}^{l} a_i 4^i$, where $a_l \neq 0$, so that $4^l \leq n < 4^{l+1}$. Thus we have a_l complete blocks of length 4^l, each one contributing $\frac{1}{3}(4^l - 1)$ zeros. The zero from the 4^{l+1} congruence comes at the $(4^{l+1}-1)$st position (since we are in the first block of this length: recall Lemma 7) and so we only count it if $n = 4^{l+1} - 1$, i.e. if all the a_i, $0 \leq i \leq l$, are equal to 3. If $a_{l-1} \geq 2$, or $a_{l-1} = 1$ and all subsequent a_i, $i = l-2, \ldots, 0$, are equal to 3, (so that $n = a_l 4^l + 2.4^{l-1} - 1$), then we must count the zero from the next incomplete block of length 4^l. Otherwise we just have the zeros in a complete block of length 4^{l-1}, and we move on to a_{l-2}, a_{l-3}... etc., until a_1, having to add additional zeros to the complete blocks only when $a_i \geq 2$ or $a_i = 1$ and all subsequent coefficients are 3. Care must be taken within a block of length 4, since the zero occurs for positions congruent to 1 mod 4, and so if $a_0 > 0$ we must count this zero. Putting all of this together, we get that the total number of zeros in a Type I sequence of length n is

$$\begin{aligned} N^0(n) &= \sum_{i=1}^{l} \frac{a_i}{3}(4^i - 1) + \#\{i : a_i \geq 2, i \neq l, i \neq 0\} \\ &\quad + \begin{cases} 1 & \text{if } a_0 > 0 \\ 0 & \text{if } a_0 = 0 \end{cases} \\ &\quad + \begin{cases} 1 & \text{if last} \geq 2 \text{ consecutive } a_i\text{s are } 13...3 \\ 0 & \text{otherwise} \end{cases} \\ &\quad + \begin{cases} 1 & \text{if all } a_i = 3,\ i = 0, \ldots, l \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

This gives us the following theorem:

Theorem 12. *The density of a Type I sequence of length $n = \sum_{i=0}^{l} a_i 4^i$, where $a_l \neq 0$, is given by*

$$D_I(n) = \frac{2}{3} + \frac{1}{n}\left(\sum_{i=0}^{l} \frac{a_i}{3} - \#\{i : a_i \geq 2, i \neq l, i \neq 0\} - K\right),$$

where K, $0 \leq K \leq 2$, counts the occurences of the events $\{a_0 > 0\}$, $\{$all $a_i = 3$ for $i = 0, \ldots l\}$ and $\{$the last ≥ 2 consecutive a_i are 13...3$\}$.

Example 13. *The Type I sequence has zeros at positions 3, 5, 9, 13, 17, 21, 25, 29, 33, 37, 41, 45, 49, 53, 57, 61, 65... etc., from the mod 4 congruence; 15, 23, 39, 55, 71... etc., from the mod 16; 63, 95... etc., from the mod 64. Thus for $n = 63$, we can count directly the 21 zeros. Using the formula in the above theorem:*

$$n = 3.16 + 3.4 + 3 \;\Rightarrow\; a_2 = a_1 = a_0 = 3,$$

and

$$N^0(63) = (16-1) + (4-1) + 1 + 1 + 1 = 21,$$

since $a_0 > 0$, $a_1 \geq 2$, and all the a_i's are 3.

2.2 Limits, maximum and minimum densities

We can see that, as $n \to \infty$, $D_R(n) \to 0$ and $D_M(n) \to 0$. Since we know that the density of a Generalised Rueppel sequence is at least that of the M-K sequence and at most that of the Rueppel sequence, by squeezing, the limit of the density of a GRS must also be 0. For the Type I and II sequences as $n \to \infty$, $D(n) \to \frac{2}{3}$. Since $D_M(n) \leq D_{GRS}(n)$ for all n, and its limit is zero, is the M-K sequence the PLCP with uniformly least density? That is, for all PLCP sequences $S(n)$, of length n, is it true that $D_M(n) \leq D_S(n)$ for all n? This is an open problem stated in [9].

Consider firstly the following lemma:

Lemma 14. *If there* does *exist a uniformly least dense PLCP, then it must be the Morii–Kasahara sequence.*

Proof. Suppose an infinite PLCP, L, with uniformly least density exists, and that it has a 1 in one of its even positions. Consider the sequence which agrees with L in all but that even bit, i.e. it has a 0 instead of a 1 in that position, then this sequence has strictly less density than L at that point, which contradicts the uniform minimality of L. Hence L must have zeros in all even positions, and so must be the M-K sequence.

Suppose there exists a PLCP, S, with lower density than the M-K sequence at some point. Then there exists a least index i such that $D_S(i) < D_M(i)$. For all $j < i$ we must therefore have $D_S(j) \geq D_M(j)$, in particular $D_S(i-1) \geq D_M(i-1)$. For the density of S to go from at least the density of M-K at length $i-1$ to strictly less than it at length i, we must have that $s_i = 0$ and $m_i = 1$. Hence $i = 2^K - 1$ for some K.

Lemma 15. *Let $i = 2^K - 1 > 1$ be the smallest index such that $D_S(i) < D_M(i)$, then*

$$D_S(j) = D_M(j), \quad \textit{for all } j < i.$$

Proof. Suppose there exists a greatest index $k < i$ such that $D_S(k) > D_M(k)$. Note that $k+1 < i$ also, as $D_S(k) > D_M(k)$ implies $D_S(k+1) = D_M(k+1)$, since we can at best only add one other one to the M-K sequence at the $k+1$st bit, and leave the number of ones in S unchanged. Thus we must have $s_{k+1} = 0$ and $m_{k+1} = 1$, which implies that $k+1 = 2^h - 1$ for some $h < K$. We must also have that $D_S(j) = D_M(j)$ for all $k < j < i$, and thus $s_j = m_j$ for $k+1 < j < i$.

Table 1.

Position:	1	2	$\cdots$	$2^h - 1$	2^h	$\cdots$	$2^{h+1} - 2$	$2^{h+1} - 1$
M-K	1	0	$\cdots$	1	0	$\cdots$	0	1
S	1	?	$\cdots$	0	0	$\cdots$	0	0

Now suppose $h + 1 \neq K$. Thus $s_{2^{h+1}-1} = m_{2^{h+1}-1} = 1$ since the bits must agree up to $2^K - 2$. But from the PLCP recurrence:

$$s_{2^{h+1}-1} = s_{2^{h+1}-2} \oplus s_{2^h-1} = s_{2^{h+1}-2} = 1,$$

since $s_{2^h-1} = 0$. Since $m_{2^{h+1}-2} = 0$, this contradicts the fact that the bits must agree, so we conclude that $h + 1 = K$ (as shown in Table 1).

Now let $H \leq k$ satisfying $2^l - 1 < H < 2^{l+1} - 1$, for some $1 \leq l \leq h$, be the greatest index such that $s_H = 1$ and $m_H = 0$. H must exist for S to have greater density than M-K at length k. Since $m_{2H} = 0$ by definition, we must have $s_{2H} = 0$, otherwise we would contradict the maximality of H. Thus

$$s_{2H+1} = s_{2H} \oplus s_H = 0 \oplus 1 = 1.$$

This implies $m_{2H+1} = 1$, otherwise we would again contradict the maximality of H, and so $2H + 1 = 2^L - 1$ for some L. But $2^l - 1 < H < 2^{l+1} - 1$ implies that

$$2^{l+1} - 1 < 2H + 1 < 2^{l+2} - 1,$$

which is impossible if $2H + 1 = 2^L - 1$. Hence we have reached a contradiction, and so there cannot exist a $k < i$ with $D_S(k) > D_M(k)$, and so we have proved that $D_S(j) = D_M(j)$ for all $j < i$.

Theorem 16. *The M-K sequence is the uniformly least dense PLCP.*

Proof. By the above lemma, if there does exist a PLCP, S, with lower density at some length, then S must agree with the M-K sequence up to length $i - 1$, using the above notation, and then $s_i = 0$ and $m_i = 1$. Recall $i = 2^K - 1$, so

$$\begin{aligned} s_i = 0 &= s_{2^K-2} \oplus s_{2^{K-1}-1} \\ &= m_{2^K-2} \oplus m_{2^{K-1}-1} \\ &= 0 \oplus 1 = 1 \end{aligned}$$

which is impossible. Thus S cannot exist and we have proved the theorem.

Now we know the uniformly least dense PLCP, is there a corresponding uniformly most dense one?

Lemma 17. *If a uniformly most dense PLCP exists, then it must be the Type I sequence.*

Proof. If it does exist, then it must have all the even positions equal to 1, since otherwise we could just replace a zero even bit with a 1 to get a sequence with strictly greater density for that length. The sequence with all its even bits equal to 1 is the Type I sequence.

Table 2 shows experimental results of PLCP densities. It gives for each sequence length up to 56 the maximum number of ones, found by computer search, over all the PLCPs, and compares this with the number of ones in the Type I and Type II sequences.

The values of n marked with a * are those for which the number of ones in the Type I sequence is *strictly* less than the maximum. This shows that the Type I sequence cannot have a uniformly maximum density for all n, even though it does attain the maximum number at some values. Hence we have proved:

Theorem 18. *There is no uniformly most dense PLCP.*

Now we know that there cannot be a uniformly most dense PLCP, what can we say about sequences which are close to it? From the experimental results, we see that both the Type I and Type II sequences either attain the maximum, or only have a couple less ones than the maximum at that length. Since the limits of both densities as $n \to \infty$ is $\frac{2}{3}$, is it true that this is the limit of the maximum density? The following theorem goes a little way to showing this, by proving it only for specific values of n.

Theorem 19. *Considering only values of n of the form $n = 2^K - 1$, for $K \geq 1$, we have:*

$$D_S(n) \leq D_{II}(n)$$

for all PLCP sequences $S(n)$, so that

$$\lim_{K\to\infty} \max_S D_S(2^K - 1) = \frac{2}{3}.$$

Proof. Recall that the density is given by

$$D(n) = \frac{1}{n}\sum_{i=1}^{n} s_i,$$

where the addition is over the integers. For a binary PLCP sequence we also have that $s_{2i+1} = s_{2i} \oplus s_i$, which in terms of normal integer addition is the same as

$$s_{2i+1} = s_{2i} + s_i - 2s_{2i}s_i.$$

Table 2. PLCP densities

n	Maximum	Type I	Type II	n	Maximum	Type I	Type II
1	1	1	1	29*	20	19	20
2	2	2	1	30*	21	20	20
3	2	2	2	31	21	21	21
4	3	3	3	32	22	22	21
5*	4	3	4	33*	23	22	22
6*	5	4	4	34*	24	23	22
7	5	5	5	35	24	24	23
8	6	6	5	36	25	25	24
9*	7	6	6	37*	26	25	25
10*	8	7	6	38*	27	26	25
11	8	8	7	39*	27	26	26
12	9	9	8	40*	28	27	26
13	9	9	9	41*	29	27	27
14	10	10	9	42*	30	28	27
15	10	10	10	43*	30	29	28
16	11	11	11	44*	31	30	29
17*	12	11	12	45*	31	30	30
18*	13	12	12	46*	32	31	30
19	13	13	13	47	32	32	31
20	14	14	14	48	33	33	32
21*	15	14	15	49*	34	33	33
22*	16	15	15	50*	35	34	33
23*	16	15	16	51	35	35	34
24*	17	16	17	52	36	36	35
25*	18	16	17	53	36	36	36
26*	19	17	17	54	37	37	36
27*	19	18	18	55	37	37	37
28*	20	19	19	56	38	38	37

So for $n = 2k+1$;

$$\begin{aligned}\sum_{i=1}^{2k+1} s_i &= s_1 + \sum_{i=1}^{k}(s_{2i} + s_{2i+1}) \\ &= 1 + \sum_{i=1}^{k}(2s_{2i} + s_i - 2s_{2i}s_i) \\ &= 1 + 2\sum_{i=1}^{k} s_{2i}(1-s_i) + \sum_{i=1}^{k} s_i\end{aligned}$$

Now suppose $k = 2j+1$, so we can break down $\sum_{i=1}^{k} s_i$ into the same sums, but from 1 to j, and continue in this way until we come down to s_1. Noting that $2^{K+1} - 1 = 2(2^K - 1) + 1 = 2(2(2^{K-1} - 1) + 1) + 1$ and so on, we have:

$$\sum_{i=1}^{2^{K+1}-1} s_i = K + 2\left(\sum_{h=1}^{K}\left[\sum_{i=1}^{2^h-1} s_{2i}(1-s_i)\right]\right) + s_1$$

Hence

$$D_S(2^{K+1}-1) = \frac{K + 1 + 2\left(\sum_{h=1}^{K}\left[\sum_{i=1}^{2^h-1} s_{2i}(1-s_i)\right]\right)}{2^{K+1}-1}$$

Since all the s_i are either 0 or 1, this expression for the density is maximized when $s_{2i} = 1 - s_i$ for all i, since we get as many 1×1's as possible in the $s_{2i}(1-s_i)$ terms. This is the Type II sequence.

If K is even, $K = 2l$, say, then

$$n = 2^{2l} - 1 = 3.2^{2l-2} + 3.2^{2l-4} + \cdots + 3.4 + 3,$$

so all the a_i from $i = 0$ to $l-1$ are equal to 3.

If K is odd, $K = 2l-1$, say, then

$$n = 2^{2l-1} - 1 = 1.2^{2l-2} + 3.2^{2l-4} + \cdots + 3.4 + 3,$$

so $a_{l-1} = 1$ and the rest are 3.

Putting these into the formula in Theorem 10 for the density of the Type II sequence, when K is even we have no coefficients equal to either 1 or 2, and when K is odd we only have one coefficient equal to 1.

Therefore we have proved that

$$D_{II}(2^K - 1) = \begin{cases} \frac{2}{3} & \text{if } K \text{ is even,} \\ \frac{2}{3} + \frac{1}{3(2^K-1)} & \text{if } K \text{ is odd,} \end{cases}$$

which gives the limit stated in the theorem.

3 Conclusion

Although these PLCPs have a profile like random sequences, they are not unpredictable and do not in general have the balance property of random sequences. (There are however examples of some which do; the PLCP defined by $s_{2i} = \overline{s_{2i-1}}$ does satisfy $D(n) \to \frac{1}{2}$ as $n \to \infty$, since each odd bit has its complement following it.) The Morii–Kasahara PLCP has the uniformly lowest proportion of ones, and this tends to 0 as the length tends to infinity, showing that zeros can dominate nearly all the bits of very long PLCPs. This is clearly undesirable for a keystream sequence. It was shown that no PLCP always has the maximum density for each length, but some examples come close to it, and for these examples the density tends to 2/3. We prove that the limit, as $K \to \infty$, of the maximum density for lengths of the form $2^K - 1$ is 2/3, and we conjecture[2] that this is the limit of the maximum density for all lengths.

References

1. Beker, H. and Piper, F. (1982). *Cipher systems—the protection of communications.* Van Nostrand, London.

2. Golomb, S.W. (1967). *Shift Register Sequences.* Holden–Day, San Francisco.

3. Rueppel, R.A. (1985). Linear complexity and random sequences. *Proceedings of Eurocrypt—LNCS*, **219**, 167–188.

4. Massey, J.L. (1969). Shift register synthesis and BCH decoding. *IEEE Transactions on Information Theory*, **IT-15**, 122–127.

5. Berlekamp, E.R. (1968). *Algebraic Coding Theory*, McGraw-Hill.

6. Dai, Z.-D. (1986). Proof of Rueppel's linear complexity conjecture. *IEEE Transactions on Info. Theory*, **IT-32**, 440–443.

7. Massey, J.L. and Wang, M.-Z. (1986). The characterisation of binary sequences with a perfect linear complexity profile. *Presented at Eurocrypt '86, Linköping, Sweden.*

8. Niederreiter, H. (1986). Continued fractions for formal power series, pseudorandom numbers and linear complexity of sequences. *Contributions to General Algebra*, **5**, 221–233.

9. Morii, M. and Kasahara, M. (1992). Perfect staircase profiles of linear complexity for finite sequences. *Info. Processing Letters*, **44**, 85–89.

[2]Since writing this paper, this conjecture has now been proved to be correct.

When Constraints are Easy to Satisfy

P. Jeavons*, D. Cohen* and M. Cooper**

**Department of Computer Science, Royal Holloway, University of London, and*
***IRIT, University of Toulouse III, France*

Abstract

Many practical combinatorial search problems may be expressed as constraint satisfaction problems, in which values must be assigned to a set of variables subject to a given set of constraints. Finding solutions to a constraint satisfaction problem is known to be an NP-complete problem in general, but may be tractable in cases where the form of allowed constraints is restricted. In this paper we describe two sets of constraints which give rise to tractable problems and give polynomial time algorithms for solving such problems. We also prove that the class of problems generated by *any* superset of these sets of constraints is NP-complete.

1 Introduction

A number of important problems in artificial intelligence, graph theory and operational research may be formulated very naturally as *constraint satisfaction problems* [14,16]: they involve putting together pieces of information (constraints) to obtain a global solution which simultaneously satisfies all of them. Recent examples include object recognition problems for robots [9], generalized graph coloring [13] and stock cutting problems [6].

Finding solutions to a constraint satisfaction problem is known to be an NP-complete problem in general [14] even when the constraints are restricted to binary constraints. However, many of the problems which arise in practice have special properties which allow them to be solved efficiently. The question of identifying restrictions to the general problem which are sufficient to ensure tractability has been discussed by a number of authors [3,7,10,17].

In this paper we examine what restrictions must be imposed on the type of constraints allowed in order to ensure that a constraint satisfaction problem may be solved efficiently. We identify two distinct families of tractable constraints, and we give polynomial time algorithms for solving any problem involving constraints from either one of those families.

Finally, we show that any collection of constraints which includes the whole of these families and *any* other constraint will generate a class of problems which is NP-complete. This means that these tractable families are both defined in the most general way possible (assuming P is not equal to NP).

2 Definitions

Definition 1. *A* constraint satisfaction problem, *P, consists of*

- *A finite set of variables, N, identified by the natural numbers $1, 2, \ldots, n$.*
- *A finite domain of values, D.*
- *A set of constraints $\{C(S_1), C(S_2), \ldots, C(S_c)\}$.*

Each S_i is an ordered subset of the variables, and each constraint $C(S_i)$ is a set of tuples indicating the mutually consistent values for the variables in S_i.

A *solution* to a constraint satisfaction problem is an assignment of values to the variables which is consistent with all of the constraints.

The length of the tuples in a given constraint will be called the 'arity' of that constraint. In particular, unary constraints specify the allowed values for a single variable, and binary constraints specify the allowed combinations of values for a pair of variables.

It is convenient to make use of the following operations from relational algebra [2].

Definition 2. *Let S be any ordered set of r variables and let $C(S)$ be a constraint on S.*

For any ordered subset $S' \subseteq S$, let $(i_1, i_2, \ldots, i_k)$ be the indices of the elements of S' in S. Define the projection of $C(S)$ onto S', denoted $\pi_{S'}(C(S))$, as follows

$$\pi_{S'}(C(S)) = \{(x_{i_1}, x_{i_2}, \ldots, x_{i_k}) \mid \exists (x_1, x_2, \ldots, x_r) \in C(S)\}.$$

Definition 3. *For any constraints $C(S_1)$ and $C(S_2)$, the* join *of $C(S_1)$ and $C(S_2)$, denoted $C(S_1) \bowtie C(S_2)$ is the constraint on $S_1 \cup S_2$ containing all tuples t such that $\pi_{S_1}(\{t\}) \subseteq C(S_1)$ and $\pi_{S_2}(\{t\}) \subseteq C(S_2)$.*

We shall assume, for simplicity, that each variable is subject to at least one constraint. Hence, the set of all solutions to a constraint satisfaction problem P, denoted Sol(P), is simply the join of all the constraints [10]

$$\text{Sol}(P) = C(S_1) \bowtie C(S_2) \bowtie \cdots \bowtie C(S_c).$$

The decision problem for a constraint satisfaction problem is to determine whether or not this join is non-empty. For the purposes of this paper, a CSP will be considered solved once a single solution has been found, or alternatively the fact that there are no solutions has been established.

The class of problems in which the constraints all belong to some set $\mathcal{C}$ will be denoted CSP ($\mathcal{C}$).

3 0/1/all constraints

We now define the first family of constraints, which contains only binary constraints.

Definition 4. *A binary constraint, $C(i,j)$, is a* **directed 0/1/all constraint** *if*

$$\forall x(\exists y, z((x,y) \in C(i,j) \wedge (x,z) \in C(i,j) \wedge y \neq z)$$

$$\Rightarrow \forall w \in \pi_{\{j\}}(C(i,j)) \quad ((x,w) \in C(i,j))). \tag{3.1}$$

In words, each value $x \in D$ is consistent with zero, one or all of the values in $\pi_{\{j\}}(C(i,j))$.

If both $C(i,j)$ and the corresponding constraint $C(j,i) = \{(y,x) : (x,y) \in C(i,j)\}$ are directed 0/1/all constraints, then $C(i,j)$ will be simply referred to as a "0/1/all constraint".

Example 5. *If $|D| = 2$, then every possible binary constraint is a 0/1/all constraint since each value for variable i is consistent with zero, one or two (i.e. all) values for variable j, and vice versa.*

Example 6. *To obtain a binary constraint which is* not *a 0/1/all constraint we must have a value (say 'a'), which occurs in the set of ordered pairs at some position with at least two, but not all, of the values occurring in the other position. (say 'a' and 'b' but not 'c'). Hence, a smallest possible example is*

$$\{(a,a),(a,b),(b,c)\}.$$

In order to characterise 0/1/all constraints, we define the following three types of constraint.

Definition 7. *A* **complete constraint** *is any constraint $C(i,j)$ which is equal to $A \times B$ for some $A \subseteq D$ and some $B \subseteq D$.*

A **permutation constraint** *is a constraint $C(i,j)$ which is equal to $\{(x,\sigma(x)) \mid x \in A\}$ for some $A \subseteq D$ and some bijection $\sigma : A \rightarrow B$, where $B \subseteq D$.*

A **two-fan constraint** *is a constraint $C(i,j)$ where there exists $x \in A \subseteq D$ and $y \in B \subseteq D$ with $C(i,j) = (x \times B) \cup (A \times y)$. Such a constraint will be denoted $< x, y >$.*

These three types of constraint are illustrated in Figure 1 which shows a typical complete constraint, permutation constraint and two-fan constraint. In this Figure the constraints are shown diagramatically by drawing a single point for each possible value for node i on the left, and a single point for each possible

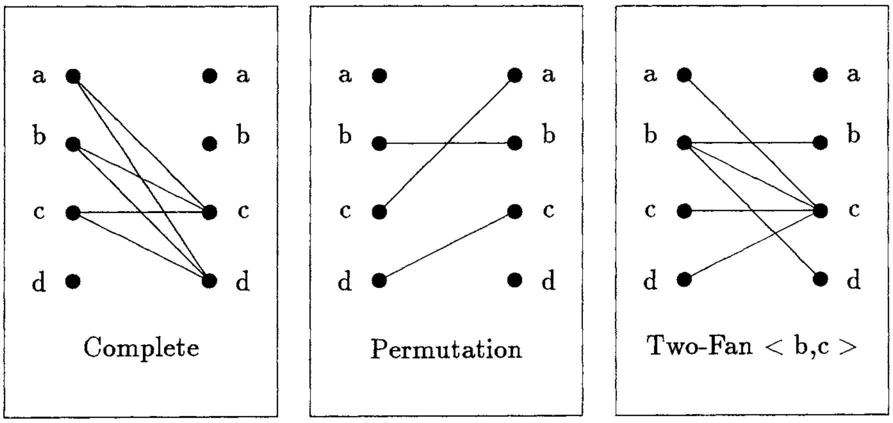


Figure 1. Three types of constraint

value for node j on the right, and then connecting two points if the corresponding combination of values is allowed by the constraint.

It can easily be verified that any complete constraint is a 0/1/all constraint, since the right-hand side of the implication in Definition 4 is always satisfied. Any permutation constraint also trivially satisfies the definition of a 0/1/all constraint, since the left-hand side of the implication is never satisfied. Finally, any two-fan constraint $C(i,j) = (x \times B) \cup (A \times y)$ is also a 0/1/all constraint since for any $p \in \pi_{\{i\}}(C(i,j))$, either $p = x$, and $C(i,j)$ allows p to be combined with every member of $\pi_{\{j\}}(C(i,j))$, or else $p \neq x$, in which case $C(i,j)$ only allows p to be combined with y (similar remarks apply to $C(j,i)$).

We will now prove that there are no other types of 0/1/all constraint. First we define a fan-out.

Definition 8. *A value $x \in D$ such that*

$$\forall y \in \pi_{\{j\}}(C(i,j)) \quad ((x,y) \in C(i,j))$$

will be called a **fan-out** *from i to j.*

We now prove a technical lemma.

Lemma 9. *If $C(i,j)$ is a 0/1/all constraint which is not a complete constraint, then there is at most one fan-out from i to j.*

Proof. Let $C(i,j)$ be a 0/1/all constraint, and assume that there is more than one fan-out from i to j. In other words, there exists w, $x \in D$, $w \neq x$ such that

$$\forall y \in \pi_{\{j\}}(C(i,j)) \quad ((w,y) \in C(i,j) \wedge (x,y) \in C(i,j)).$$

Then, directly from Definition 4, we can deduce that

$$\forall y \in \pi_{\{j\}}(C(i,j)) \quad (\forall z \in \pi_{\{i\}}(C(i,j)) \quad ((z,y) \in C(i,j)))$$

which is equivalent to the definition of a complete constraint. The lemma follows immediately.

We may now show that all 0/1/all constraints must be of a type described in Definition 7.

Lemma 10. (Categorisation) *Any 0/1/all constraint is either a complete constraint, a permutation constraint, or a two-fan constraint.*

Proof. Let $C(i,j)$ be a 0/1/all constraint which is neither a permutation constraint nor a complete constraint. We shall demonstrate that $C(i,j)$ must be a two-fan constraint.

Without loss of generality, suppose that $C(i,j)$ is not a permutation constraint because $(x,u),(x,v) \in C(i,j)$ where $u \neq v$. By Definition 4, x must be a fan-out, that is

$$\forall z \in \pi_{\{j\}}(C(i,j)) \quad ((x,z) \in C(i,j)).$$

If $\pi_{\{i\}}(C(i,j)) = \{x\}$, then $C(i,j)$ would be complete, so we can choose w $\neq x$ such that $(w,y) \in C(i,j)$ for some $y \in \pi_{\{j\}}(C(i,j))$. But then, by Definition 4, since $C(j,i)$ is a 0/1/all constraint and $(w,y),(x,y) \in C(j,i)$, y must be a fan-out, that is

$$\forall z \in \pi_{\{i\}}(C(i,j)) \quad ((z,y) \in C(i,j)).$$

By Lemma 9, we know that $C(i,j)$ contains at most one fan-out from i to j, and at most one fan-out from j to i. This means that $C(i,j)$ is a two-fan constraint with fan-outs at x and y.
The family of all 0/1/all constraints over some fixed domain of values D will be denoted $\mathcal{Z}_D$. To complete the description of $\mathcal{Z}_D$, we show that it is preserved under some standard operations on constraints.

Lemma 11. *The family of 0/1/all constraints, $\mathcal{Z}_D$, is closed under the following operations*

1. *Intersection,* $\cap$.
2. *Composition,* $\circ$, *where* $C(i,j) \circ C(j,k)$ *is the constraint* $C(i,k)$ *between* i *and* k, *given by*

$$C(i,k) = \{(p,q) : \exists r \in D, (p,r) \in C(i,j) \wedge (r,q) \in C(j,k)\}.$$

3. *Restriction, $\|$, where $C(i,j) \| (P \times Q) = \{(p,q) \in C(i,j) : p \in P, q \in Q\}$.*
4. *Permutation, for permutations σ and σ' of D, $C^{\sigma \times \sigma'}(i,j) = \{(\sigma(p), \sigma'(q)) : (p,q) \in C(i,j)\}$.*

Proof. The proof for intersection, restriction, and permutation follows immediately from Definition 4. The proof for composition follows from Lemma 10 by considering compositions of each of the three possible types of 0/1/all constraints individually.

3.1 A polynomial time algorithm for CSP ($\mathcal{Z}_D$)

Definition 12. *A CSP is* **path consistent** *(3-consistent) if, for any three nodes i, j and k, $C(i,k) \subseteq C(i,j) \circ C(j,k)$.*

The next two lemmas establish special properties of the class CSP($\mathcal{Z}_D$) relating to path consistency.

Lemma 13. *After application of a path consistency algorithm, any problem $P \in$ CSP($\mathcal{Z}_D$) is still in CSP ($\mathcal{Z}_D$).*

Proof. Path consistency can be established by repeatedly applying the operation

$$C(i,k) := C(i,k) \cap (C(i,j) \circ C(j,k))$$

for all triples of nodes i, j, k until there are no more changes in the constraints. (Any pair of nodes, i, j, for which no constraint is specified originally is assumed to be constrained by the complete constraint $D \times D$, which is a 0/1/all constraint.) Since the set of 0/1/all constraints is closed under intersection and composition (Lemma 11), we can deduce that it is closed under application of this path consistency algorithm.

Lemma 14. *In any path consistent $P \in$ CSP($\mathcal{Z}_D$), for any value $p \neq q$, if $C(i,j) =< s,p >$ and $C(j,k) =< q,r >$, then $C(i,k) =< s,r >$.*

Proof. Suppose that $C(i,j) =< s,p >$ and $C(j,k) =< q,r >$, where $p \neq q$. (Two constraints of this form are illustrated in Figure 2.) Let $x \in \pi_{\{i\}}(C(i,j))$. Now $(x,p) \in C(i,j)$, but $(p,y) \in C(j,k)$ only for $y = r$ (since $p \neq q$). Therefore, we must have $(x,r) \in C(i,k)$, otherwise (x,p) would have to be deleted from $C(i,j)$ by path consistency.

Similarly $(s,y) \in C(i,k)$ for all $y \in \pi_{\{k\}}C(i,k)$. Therefore $C(i,k) \supseteq< s,r >$. However, we know, by path consistency, that $C(i,k) \subseteq C(i,j) \circ C(j,k)$, the composition of $C(i,j)$ and $C(j,k)$. Any path from i to k either starts at s or finishes at r (see Figure 2), and hence $C(i,j) \circ C(j,k) \subseteq< s,r >$. Therefore $C(i,k) =< s,r >$.

We now prove the main result establishing the tractability of problems in $CSP(\mathcal{Z}_D)$.

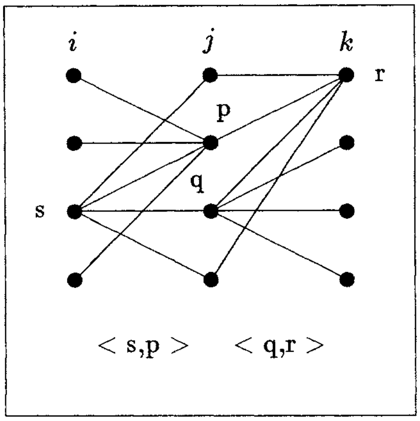


Figure 2. The composition of 2 two-fans

Theorem 15. *Any constraint in which the constraints are all 0/1/all constraints may be solved in polynomial time.*

Proof. Let P be a constraint in CSP($\mathcal{C}_D$), and compute an equivalent path-consistent problem P' also in CSP($\mathcal{C}_D$), by Lemma 13. It is well known that path consistency in any binary constraint satisfaction problem can be established in $O(n^3|D|^3)$ time [11].

We will show that for any i in the range 3 to n, any assignment of values $x_1, \ldots, x_{i-1}$ which satisfies all the constraints in P' on variables $1, 2, \ldots, i-1$ can be extended to an assignment $x_1, \ldots, x_{i-1}, x_i$ which satisfies all constraints on variables $1, 2, \ldots, i$.

There are two distinct cases

Case (a): There is a $j \in \{1, \ldots, i-1\}$ such that $C(j,i)$ is a permutation constraint associated with the bijection σ. By path consistency, x_j must be consistent with some value for variable i, so $\sigma(x_j)$ must be defined. In this case, we can simply choose $x_i = \sigma(x_j)$. By path consistency, x_i must be consistent with each x_k, $k < i$, since x_j is consistent with each x_k, $k < i$.

Case (b): Each constraint $C(j,i)$, for $j < i$, is either a complete constraint or a two-fan $< p, q >$.

Call a variable $j \in \{1, \ldots, i-1\}$ *restrictive* if $C(j,i) =< p_j, q_j >$ with $p_j \neq x_j$. If there is a restrictive variable r with $C(r,i) =< p_r, q_r >$, and a second restrictive variable s with $C(s,i) =< p_s, q_s >$, then we will show that $q_s = q_r$. Assume for contradiction that $q_s \neq q_r$. Then, by Lemma 14, $C(r,s) =< p_r, p_s >$. However, r and s are restrictive, so $x_r \neq p_r$ and $x_s \neq p_s$, which implies that

$(x_r, x_s) \notin C(r, s)$, so the pair of values x_r and x_s do not satisfy the constraint $C(r, s)$, which contradicts the choice of x_r and x_s.

If there is a restrictive variable r with $C(r, i) =< p_r, q_r >$, then we assign the value q_r to variable i with q_r, otherwise we choose an arbitrary value for variable i from the set $\pi_{\{i\}}(C(i, 1))$. We have shown that this choice of value satisfies $C(i, j)$ for any restrictive variable $j \in \{1, \ldots, i-1\}$, and it is also clearly satisfies $C(i, j)$ for any unrestrictive variable $j \in \{1, \ldots, i-1\}$.

This completes the proof for the second case. Hence, any pair $(x_1, x_2) \in C(1, 2)$ of P', may be extended to a complete solution of P' and hence of P.

Finally, this complete solution can be found in $O(n^2|D|)$ time, since for each new variable we simply check each possible value against all the constraints between that variable and the preceding variables.

3.2 NP-completeness of larger constraint sets

In this section we shall demonstrate that any superset of the set of 0/1/all constraints can generate intractable problems. Hence this set of constraints is a maximal set of tractable constraints.

We first prove the following lemmas.

Lemma 16. *Any constraint $C(i, j)$ which is not a 0/1/all constraint is equivalent to one of the three constraints illustrated in Figure 3 after suitable permutations and restrictions.*

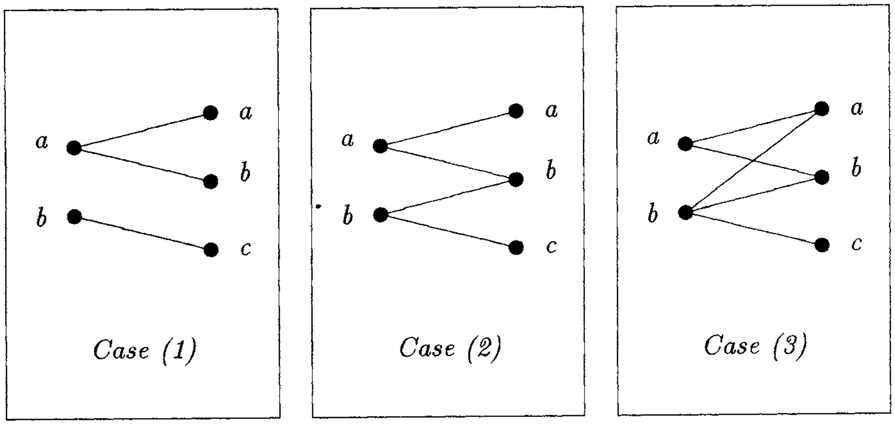


Figure 3. The three generic constraints which are not 0/1/all constraints

Proof. Without loss of generality, suppose that $C(i,j)$ does not satisfy Definition 4 of a directed 0/1/all constraint. Then there must be a value, say a, in D which is consistent with at least two, but not all, values in $\pi_{\{j\}}(C(i,j))$. Suppose that $(a,a),(a,b) \in C(i,j)$ and $(a,c) \notin C(i,j)$, where $c \in \pi_{\{j\}}(C(i,j))$. But $c \in \pi_{\{j\}}(C(i,j))$ implies that there is a value in $\pi_{\{i\}}(C(i,j))$, say $b \neq a$, such that $(b,c) \in C(i,j)$. We identify four different cases

1. $(b,a),(b,b) \notin C(i,j)$.
2. $(b,a) \notin C(i,j)$, $(b,b) \in C(i,j)$.
3. $(b,a) \in C(i,j)$, $(b,b) \notin C(i,j)$.
4. $(b,a),(b,b) \in C(i,j)$.

Cases 1, 2 and 4 are illustrated in Figure 3. Case 3 is equivalent to case 2 after permutation of values a and b. Hence after restricting the possible values of i to $\{a,b\}$, and the possible values for j to $\{a,b,c\}$, $C(i,j)$ is equal to one of the constraints given in Figure 3.

Lemma 17. *Any constraint between two variables with three possible values can be constructed by composition and intersection from any one of the constraints of Figure 3 and permutations of it.*

Proof. Figure 4 shows that, by inserting extra variables k_1, k_2, k_3 between variables i and j in Case 1, k_1, k_2 in Case 2 and k_1 in Case 4, we can arrange that the resulting constraint allows every pair of values except for one, say (a,a).

For example, in Case 1, $(a,b) \in C(i,j)$ since there exist a consistent sequence of values of the form $(a,\ldots,b)$ for variables (i,k_1,k_2,k_3,j), namely (a,b,c,b,b). However, $(a,a) \notin C(i,j)$ since there is no consistent sequence of values of the form $(a,\ldots,a)$ for (i,k_1,k_2,k_3,j). The four constraints $C_{ik_1}, C_{k_1k_2}, C_{k_2k_3}, C_{k_3j}$ can all be obtained from Case 1 in Figure 3 by permutations of the value sets.

Hence, by permuting the value sets as necessary, we can construct the constraint which allows every possible pair of values except (x,y), for any choice of values x and y. By connecting these constraints in parallel we can form the constraint which disallows *any* desired combinations of values, and the result follows.

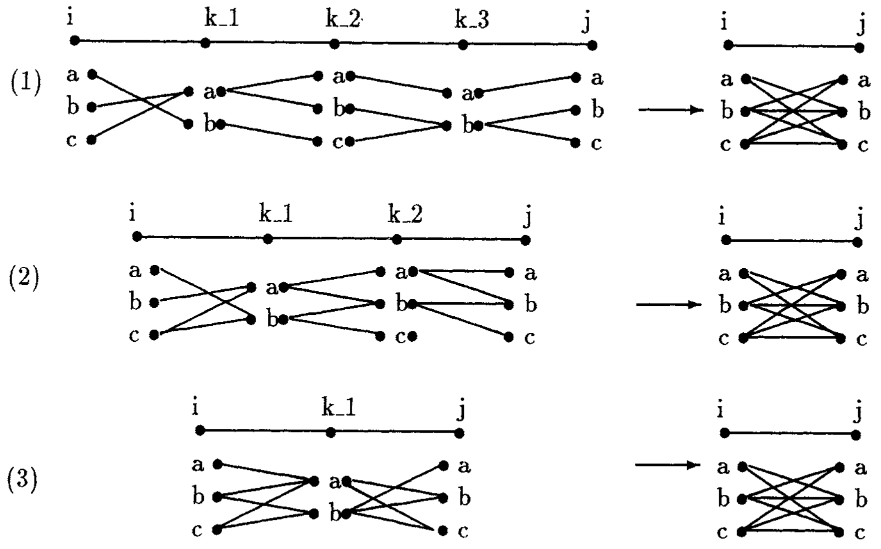


Figure 4. Construction used in Lemma 17

Lemma 18. *Let T be any set of constraints with at most three values for each variable. Let S be any set of constraints closed under permutations and restrictions, which does not consist entirely of 0/1/all constraints.*

There is a polynomial-time reduction from CSP(T) to CSP(S).

Proof. By Lemma 16 S must contain one of the constraints of Figure 3. By the proof of Lemma 17 any member of T may be constructed from this constraint and its permutations by using a network containing at most 27 additional variables and 36 edges. (Since at most 3 additional variables and 4 edges are required for the series construction shown in Figure 4.) Hence, given any problem P in CSP(T) we may replace each edge in the constraint graph with a network generating the same constraint in polynomial time, to obtain a problem P' in CSP(S). The solutions to P are projections of the solutions to P' onto the original variables.

Note that any permutation or restriction may be obtained by composition with some element of $\mathcal{Z}_D$. Hence we may apply Lemma 18 to the set $\mathcal{Z}_D \cup \{C\}$, giving the following theorem.

Theorem 19. *If C is not a 0/1/all constraint, then CSP($\mathcal{Z}_D \cup \{C\}$) is NP-complete.*

Proof. Any constraint satisfaction problem clearly belongs to NP [8].

Furthermore, GRAPH 3-COLORABILITY is an NP-complete problem [8] and Lemma 18 gives a polynomial-time reduction from GRAPH 3-COLORABILITY to CSP($\mathcal{C}_D \cup \{C\}$).

4 Max-closed constraints

In this section we shall break the symmetry of the domain of values D and assume that D is a totally-ordered set. This assumption is not unreasonable, since in many applications the domain may be considered to be a subset of the natural numbers, or the real numbers.

As a consequence of assuming that the domain is ordered, we may define the following operation on the elements of any constraint.

Definition 20. *Let C be a constraint and let $t = (x_1, x_2, \ldots, x_r)$ and $t' = (x'_1, x'_2, \ldots, x'_r)$ be elements of C.*

The maximum of t and t', denoted $t \sqcup t'$ is defined as follows

$$t \sqcup t' = (\max(x_1, x'_1), \max(x_2, x'_2), \ldots, \max(x_r, x'_r)).$$

The minimum of t and t', denoted $t \sqcap t'$ is defined as follows

$$t \sqcap t' = (\min(x_1, x'_1), \min(x_2, x'_2), \ldots, \min(x_r, x'_r)).$$

Using these operations on tuples, we may now define the following property of constraints.

Definition 21. *A constraint C is said to be max-closed if, for all $t, t' \in C$,*

$$t \sqcup t' \in C.$$

Similarly, C is said to be min-closed if, for all $t, t' \in C$,

$$t \sqcap t' \in C.$$

Lemma 22. *All unary constraints are max-closed.*

Proof. Since the domain D is assumed to be totally ordered, we know that for any unary constraint C, and any $(x), (x') \in C$, $(x) \sqcup (x') = (\max(x, x')) = (x)$ or (x'), so $(x) \sqcup (x') \in C$. Hence, C is max-closed.

Example 23. *Figure 5 shows some examples of binary max-closed constraints.*

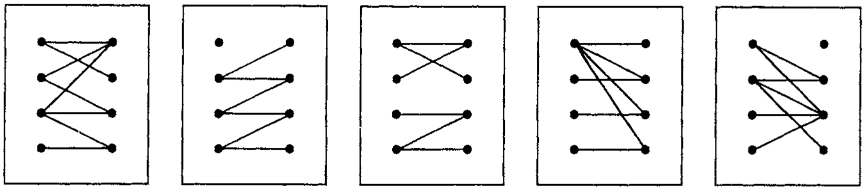


Figure 5. Examples of binary max-closed constaints

Example 24. *Any binary constraint which restricts the differences between the values of two variables to a fixed interval of the real numbers [4], is max-closed.*

Example 25. *The constraint programming language CHIP incorporates a number of constraint solving techniques for arithmetic and other constraints. In particular, it provides a constraint solver for a restricted class of constraints over natural numbers, referred to as basic constraints [17]. These basic constraints are of two kinds.*

Domain constraints:

- $X \in \{a_1, \ldots, a_n\}$

Arithmetic constraints:

- $aX \neq b$
- $aX = bY + c$
- $aX \leq bY + c$
- $aX \geq bY + c$

where variables are represented by upper-case letters, and constants by lower case letters, all constants are positive and a is non-zero.

It may be shown, using the definition above that all of these constraints are max-closed (and also min-closed). The results given below therefore confirm that any system of constraints of this restricted type may be solved efficiently [17].

Other arithmetic constraints which are also max-closed, and could therefore be added to this set without losing this property, include

- $a_1X + a_2Y + \cdots + rZ \geq c.$
- $aXY \geq c.$
- $(a_1X \geq b_1) \vee (a_2Y \geq b_2) \vee (a_3Z \leq b_3).$

The class of all max-closed constraints over some fixed domain D will be denoted $\mathcal{M}_D$. Note that $\mathcal{M}_D$ is not restricted to binary constraints, but includes constraints of all possible arities.

The following properties of $\mathcal{M}_D$ follow directly from the above definitions.

Proposition 26.

- *$\mathcal{M}_D$ is closed under the join operation, in other words, for any pair of constraints $C(S_1), C(S_2) \in \mathcal{M}_D$,*

$$C(S_1) \bowtie C(S_2) \in \mathcal{M}_D.$$

- *$\mathcal{M}_D$ is closed under projection, in other words, for any constraint $C(S) \in \mathcal{M}_D$ and any subset $S' \subseteq S$,*

$$\pi_{S'}(C(S)) \in \mathcal{M}_D.$$

4.1 A polynomial time algorithm for CSP($\mathcal{M}_D$)

A constraint satisfaction problem is said to be 'interconsistent' [12] if for any pair of constraints $C(S_1), C(S_2), \pi_{S_1 \cap S_2}(C(S_1)) = \pi_{S_1 \cap S_2}(C(S_2))$. When all constraints are binary, interconsistency is equivalent to arc-consistency [11] (provided that in establishing arc-consistency the constraints are updated as well as the domains).

Proposition 27. *For any $P \in CSP(\mathcal{M}_D)$ with c constraints of arity at most r, an equivalent interconsistent problem $P' \in CSP(\mathcal{M}_D)$ may be calculated in $O(c^{2-\frac{1}{r}}|D|^{2r})$ time.*

Proof. Interconsistency may be established by a succession of join and projection operations, hence by Proposition 26, P' is still in CSP($\mathcal{M}_D$).

Since interconsistency is equivalent to arc-consistency in the dual problem [5], it may be established in $O(ea^2)$ time by using an efficient algorithm for arc-consistency [1,15], where e is the number of pairs of overlapping constraints $(C(S_i), C(S_j)$ such that $S_i \cap S_j \neq \emptyset)$ and a is the maximum cardinality of a constraint. It can be shown that e is $O(c^{2-\frac{1}{r}})$ with the upper bound being achieved when the constraints form a complete hypergraph of degree r on $O(c^{\frac{1}{r}})$ vertices. The value of a is clearly $O(|D|^r)$, so the result follows.

Using Proposition 27 we are able to obtain the main result establishing the tractability of CSP($\mathcal{M}_D$).

Theorem 28. *Any constraint satisfaction problem which the constraints are all max-closed may be solved in polynomial time.*

Proof. Let P be a constraint satisfaction problem in CSP($\mathcal{M}_D$), and compute an equivalent interconsistent problem P' also in CSP($\mathcal{M}_D$), by Proposition 27.

If any of the constraints of P' are empty, then P' has no solutions, hence P has no solutions. Otherwise, for each variable i, let x_i be the maximum value allowed by all of the constraints on that variable.

$$x_i = \max\left(\bigcap_{\{C(S)\in P' \mid i\in S\}} \pi_{\{i\}}(C(S))\right)$$

We claim that $(x_1, x_2, \ldots, x_n)$ is a solution to P', and hence a solution to P.

To establish this claim, consider any constraint $C(S)$ of P', where $S = (i_1, i_2, \ldots, i_r)$, and any $i_j \in S$. By the choice of the x_i we must have some tuple $t_{i_j} \in C(S)$ whose jth coordinate is x_{i_j}.

Now consider the tuple $t = t_{i_1} \sqcup t_{i_2} \sqcup \cdots \sqcup t_{i_r}$. Since $C(S)$ is max-closed, we have $t \in C(S)$, and, by the choice of x_i, we have $t = (x_{i_1}, x_{i_2}, \ldots, x_{i_r})$. Hence $(x_1, x_2, \ldots, x_n)$ satisfies $C(S)$, and the claim follows.

Finally, it is clear that the solution $(x_1, x_2, \ldots, x_n)$ may be computed in $O(c|D|^r)$ time from P', where the values of c and r are as defined in Proposition 27.

Equivalent results clearly also hold for min-closed constraints.

4.2 NP-completeness of larger constraint sets

In this section we shall demonstrate that any superset of the set of max-closed (or min-closed) constraints can generate intractable problems. Hence each of these sets of constraints is a maximal set of tractable constraints.

We begin by characterizing max-closed constraints using the following property.

Definition 29. *A constraint $C(S)$ is said to be "crossover-closed" if, for all $i, j \in S$,*

$$[(x_i, x_j) \in \pi_{(i,j)}(C(S))] \wedge [(y_i, y_j) \in \pi_{(i,j)}(C(S))] \wedge (x_i > y_i) \wedge (x_j < y_j)$$

$$\Longrightarrow [(x_i, y_j) \in \pi_{(i,j)}(C(S))].$$

For binary constraints, this is equivalent to being max-closed.

Lemma 30. *A binary constraint is max-closed if and only if it is crossover-closed.*

However, for constraints of higher arity, it is possible to be crossover-closed without being max-closed, as the following example illustrates.

Example 31. *Consider the following constraint, C, consisting of 3 tuples*

$$C = \{(T,T,F),(T,F,T),(F,T,T)\}.$$

If the domain D is ordered such that $F < T$, then C is crossover-closed, because the projection of C onto any pair of variables is $\{(T,T),(T,F),(F,T)\}$.

However, C is not max-closed because the maximum of any pair of tuples is (T,T,T), which is not an element of C.

We now establish the precise relationship between these properties for constraints of any arity.

Lemma 32. *A constraint $C(S)$ is max-closed if and only if every intersection of $C(S)$ with max-closed constraints is crossover-closed.*

Proof. ($\Rightarrow$) If $C(S)$ is max-closed then, for all $i,j \in S$,

$$\left[(x_i,x_j) \in \pi_{(i,j)}(C(S))\right] \wedge \left[(y_i,y_j) \in \pi_{(i,j)}(C(S))\right]$$

$$\Longrightarrow \left[(\max(x_i,y_i),\max(x_j,y_j)) \in \pi_{(i,j)}(C(S))\right].$$

Hence, $C(S)$ is crossover-closed.

Furthermore, the join of $C(S)$ with any max-closed constraint remains max-closed, by Proposition 26. Hence, $C(S)$ remains crossover-closed no matter what further restrictions are imposed by max-closed constraints.

($\Leftarrow$) If $C(S)$ is not max-closed, then there exist $t_1, t_2 \in C(S)$, such that $t = t_1 \sqcup t_2 \notin C(S)$. Let $t_1 = (x_1,x_2,\ldots,x_r)$ and $t_2 = (y_1,y_2,\ldots,y_r)$ and let $t = (z_1,z_2,\ldots,z_r)$ where $z_i = \max(x_i,y_i)$.

Now impose a further constraint on S which restricts each variable $i \in S$ to values less than or equal to z_i. This additional constraint is clearly max-closed, and the intersection of $C(S)$ with this constraint results in a new constraint, which will be denoted $C'(S)$.

Choose a minimal subset $M = \{i_1,\ldots,i_m\} \subseteq S$ such that $\pi_M\{t\} \notin \pi_M(C'(S))$. By the choice of t, we have $2 \leq |M| \leq r$. Since M is minimal, for any $i_j \in M$ we have $\pi_{M\setminus\{i_j\}}\{t\} \in \pi_{M\setminus\{i_j\}}(C'(S))$. In other words, for any $i_j \in M$, $C'(S)$ must contain a tuple $(z_{i_1},\ldots,z_{i_{j-1}},z'_{i_j},z_{i_{j+1}},\ldots,z_{i_m})$ where $z'_{i_j} \neq z_{i_j}$. Since variable i_j is constrained by $C'(S)$ to take values less than or equal to z_{i_j}, it follows that $z'_{i_j} < z_{i_j}$.

Now choose any two distinct variables $i_j, i_k \in M$ and impose a further constraint on the variables in $M \setminus \{i_j,i_k\}$ (if any) which requires each variable i_p to take the value z_{i_p}. The intersection of $C'(S)$ with this additional (max-closed) constraint results in a new constraint $C''(S)$ such that

$$[(z_{i_j}, z'_{i_k}) \in \pi_{(i_j,i_k)}(C''(S))] \wedge \left[(z'_{i_j}, z_{i_k}) \in \pi_{(i_j,i_k)}(C''(S))\right]$$

$$\wedge (z_{i_j} > z'_{i_j}) \wedge (z'_{i_k} < z_{i_k}) \wedge [(z_{i_j}, z_{i_k}) \notin \pi_{(i_j,i_k)}(C''(S))] .$$

Hence $C''(S)$ is not crossover-closed.

Using this lemma, we are able to prove the main result of this section.

Theorem 33. *For any domain D, with $|D| \geq 3$, and any constraint C not in $\mathcal{M}_D$, CSP($\mathcal{M}_D \cup \{C\}$) is NP-complete.*

Furthermore, it remains NP-complete even when $\mathcal{M}_D$ is restricted to binary max-closed constraints.

Proof. Any CSP clearly belongs to NP since a solution may be checked against all of the constraints of the problem in polynomial time.

To demonstrate that CSP($\mathcal{M}_D \cup \{C\}$) is NP-complete we shall provide a polynomial time reduction from the NP-complete problem GRAPH 3-COLORABILITY [8].

To carry out this reduction, we first note that, by Lemma 32, since C is not max-closed, we may compose C with max-closed constraints to form a constraint C' which is not crossover-closed. In other words, on some pair of coordinate positions, there exist values x_1, x_2, y_1, y_2, with $y_1 < x_1$ and $y_2 > x_2$, such that C' allows the combinations (x_1, x_2) and (y_1, y_2) but does not allow the combination (x_1, y_2). Without loss of generality, we may assume that this holds in the first and last coordinate positions.

Now let r be the length of the tuples in C and consider the constraint satisfaction problem P with variables $\{1, 2, 3, \ldots, 2r, 2r+1, 2r+2\}$, domain $D = \{a, b, c, \ldots\}$ where $(a > b > c)$, and the following constraints

- $C(1, 2, \ldots, r) = C(r+1, r+2, \ldots, 2r) = C'$.
- $C(2r+1, 1) = \{(a, x_1), (b, x_1), (c, x_1), (b, y_1), (c, y_1)\}$.
- $C(2r+2, r) = \{(a, y_2), (b, y_2), (c, y_2), (b, x_2), (c, x_2)\}$.
- $C(2r+1, r+1) = \{(a, x_1), (b, x_1), (c, x_1), (a, y_1), (c, y_1)\}$.
- $C(2r+2, 2r) = \{(a, y_2), (b, y_2), (c, y_2), (a, x_2), (c, x_2)\}$.
- $C(2r+1, 2r+2) = \{(a, a), (a, b), (b, a), (b, b), (c, a), (a, c), (c, b), (b, c)\}$.

The problem P is illustrated in Figure 6. Note that the additional constraints used in P are all max-closed, hence $P \in$ CSP($\mathcal{M}_D \cup \{C\}$).

By explicitly constructing all possible solutions to P, we may show that all possible combinations of a, b and c are allowed for the variables $2r+1$ and $2r+2$ except for the pairs $(a, a), (b, b)$ and (c, c).

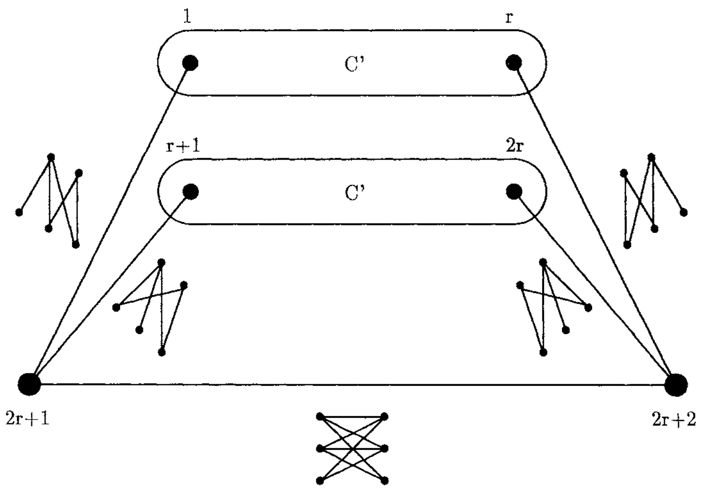


Figure 6. The CSP P used to construct a $\neq$ constraint

But this means that we may reduce any instance of GRAPH 3-COLORABILITY to a problem in CSP($\mathcal{M}_D \cup \{C\}$) in polynomial time, by replacing each edge in the graph with P and identifying the vertices of the edge with the variables corresponding to $2r+1$ and $2r+2$.

Since the above construction uses only binary constraints, this result remains true even when $\mathcal{M}_D$ is restricted to binary constraints.

Corresponding results may clearly also be obtained for min-closed constraints.

5 Conclusion

This paper has described three families of tractable constraints, which we have called "0/1/all constraints", "max-closed constraints" and "min-closed constraints".

We have shown that any constraint satisfaction problem where the constraints all lie within one of these families may be solved efficiently.

We have also shown that any class of problems allowing a larger set of constraints is NP-complete, so it is unlikely that efficient general solution techniques exist.

Acknowledgements

The authors wish to acknowledge the support of the British Council for this research.

References

1. Bessiere, C. (1994). Arc-consistency and arc-consistency again. *Artificial Intelligence*, **65**, 179–190.

2. Codd, E.F. (1970). A relational model of data for large shared databanks. *Communications of the ACM*, **13**, 377–387.

3. Dechter, R. (1992). From local to global consistency. *Artificial Intelligence*, **55**, 87–107.

4. Dechter, R., Meiri, I. and Pearl, J. (1991). Temporal constraint networks. *Artificial Intelligence*, **49**, 61–95.

5. Dechter, R. and Pearl J. (1988). Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, **34**, 1–38.

6. Dincbas, M., Simonis, H. and Van Hentenryck, P. (1988). Solving a stock cutting problem in constraint logic programming. *Logic Programming: Proceeding of the Fifth International Conference and Symposium*, MIT Press, 42–58.

7. Freuder, E.C. (1985). A sufficient condition for backtrack-bounded search. *Journal of the ACM*, **32**, 755–761.

8. Garey, M.R. and Johnson, D.S. (1979). *Computers and intractability: a guide to NP-completeness*, Freeman, San Francisco, California.

9. Grimson, W.E.L. (1990). The combinatorics of object recognition in cluttered environments using constrained search. *Artificial Intelligence*, **44**, 121–165.

10. Gyssens, M., Jeavons, P. and Cohen, D. (1994). Decomposing constraint satisfaction problems using database techniques. *Artificial Intelligence*, **66**, 57–89.

11. Han, C.C. and Lee, C.-H. (1988). Comments on Mohr and Henderson's path consistency algorithm. *Artificial Intelligence*, **36**, 125–130.

12. Janssen, P., Jegou, P., Nouguier, B. and Vilarem, M.C. (1989). A filtering process for general constraint satisfaction problems: achieving pair-wise consistency using an associated binary representation. *Proceedings of the IEEE Workshop on "Tools for Artificial Intelligence"*, 420–427.

13. Jeavons, P.G. (1993). Counting representable sets on simple graphs. *Discrete Applied Mathematics*, **47**, 33–46.

14. Mackworth, A.K. (1977). Consistency in networks of relations. *Artificial Intelligence*, **8**, 99–118.

15. Mohr, R. and Henderson, T.C. (1986). Arc and path consistency revisited. *Artificial Intelligence*, **28**, 225–233.

16. Montanari, U. (1974). Networks of constraints: fundamental properties and applications to picture processing. *Information Sciences*, **7**, 95–132.

17. Van Hentenryck, P., Deville, Y. and Teng, C-M. (1992). A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, **57**, 291–321.

A Unified Approach to Problems in Radio Channel Assignment

R.A. Leese

Mathematical Institute, Oxford, and Smith Institute, Surrey Research Park, Guildford

1 Introduction

Many problems of resource allocation can be thought of in terms of constrained combinatorial optimization. The purpose of this paper is to apply such techniques to radio channel assignment, where constraints arise from the need to avoid excessive interference levels between different signals. A key issue in the overall strategy is the way in which channels are reused in sufficiently separated areas. Channel reuse is a general feature of radio systems, and particularly important when the area of coverage is large, as for example with entertainment broadcasting and mobile telephony. The results of this work will be of interest both to the designers of large radio systems and to those responsible for spectrum management.

Cellular layouts provide the natural setting for studying channel assignment, with the most commonly studied geometry being a mesh of regular hexagons. In the simplest scenario, each cell is taken to represent the coverage area of a single transmitter, located at its centre; all transmitters are identical. Even when the true coverage areas are distorted by topographic features, regular geometries are very important in the design and planning stages, and shed light on the general principles of good spectral management.

The next section formulates the general problem, and in doing so defines the terminology that will be used throughout. Section 3 describes the use of regular tilings in regular cellular systems. Section 4 then discusses several algorithms for generating explicit assignments. A selection of results is presented in Section 5, together with topics for further investigation.

2 The channel assignment problem

The mathematical description of the assignment problem proceeds along the following lines. Suppose that the available channels are uniformly spaced in the spectrum. Each one is, in practice, a narrow band of frequencies, but for the assignment problem can be identified with the frequency at its centre. The frequency difference between adjacent channels will be called the *spectral unit.*

Channels are conveniently labelled by the positive integers, in ascending order of frequency. A channel assignment is then a map from the set $\mathcal{T}$ of all transmitters to the positive integers. Given an assignment $\mathcal{A}$, one can identify the number of different channels, $m(\mathcal{A})$, the highest channel used, $n(\mathcal{A})$, and the *characteristic distances* $\{d_i(\mathcal{A}) : i = 0, 1, 2, \ldots\}$, defined by

$$d_i(\mathcal{A}) = \min \left\{ \delta(t_1, t_2) : t_1, t_2 \in \mathcal{T} \text{ and } |\mathcal{A}(t_1) - \mathcal{A}(t_2)| = i \right\}, \qquad (2.1)$$

where (t_1, t_2) runs over all pairs of transmitters assigned channels that are i spectral units apart, and $\delta(t_1, t_2)$ is the geographical distance between them. If, for some i, there are no such pairs of transmitters, then $d_i(\mathcal{A}) = \infty$. In particular, $d_i(\mathcal{A}) = \infty$ for all $i \geq n(\mathcal{A})$.

Clearly $m(\mathcal{A}) \leq n(\mathcal{A})$, but there is no requirement that all channels $1, 2, \ldots, n(\mathcal{A})$ are used, i.e. it is not necessarily true that $m(\mathcal{A}) = n(\mathcal{A})$. Here one can make an analogy with formulations of the assignment problem in terms of graph colouring [1], where each transmitter is represented by a vertex, with pairs of potential interferers joined by edges; in particular, $m(\mathcal{A})$ will be called the *order* of the assignment, and $n(\mathcal{A}) - 1$ its *span* (assuming that channel 1 is used, which one can do without loss of generality).

Interference constraints are most commonly expressed by prescribing lower bounds D_i on the characteristic distances $d_i(\mathcal{A})$. In practice, system designers and spectrum managers need only take into account a few spectral separations (typically 4, for example, in broadcast systems); in other words $D_i = 0$ for all sufficiently large i. An assignment $\mathcal{A}$ is *feasible* if $d_i(\mathcal{A}) \geq D_i$ for all i; it is *optimal* if it has minimal span, i.e. if there is no feasible $\mathcal{A}'$ with $n(\mathcal{A}') < n(\mathcal{A})$. (Note that the optimal assignment for a given set of constraints is not necessarily unique.)

A further useful notion is to say that $\mathcal{A}$ is *saturated* if there is no $\mathcal{A}'$ having $n(\mathcal{A}') \leq n(\mathcal{A})$ and $d_i(\mathcal{A}') \geq d_i(\mathcal{A})$ for all i, with, in the second condition, strict inequality for some i. In other words, one cannot increase any characteristic distance of a saturated assignment, while at the same time maintaining all the others, without increasing the span. Note that saturated assignments are defined without reference to a set of constraints. Their importance lies in the fact that for any set of constraints D_i, there is an optimal $\mathcal{A}$ that is saturated; conversely, every saturated assignment is optimal for some set of constraints. There is therefore much to be gained from building up an understanding of saturated assignments. For example, one could construct a look-up table of saturated assignments that contains optimal solutions for whole sets of problems.

3 Regular problems

3.1 Assignment by regular tiling

The general problem of finding optimal assignments, given a set of constraints, is NP-complete. It is therefore attractive, in the initial stages at least, to aid

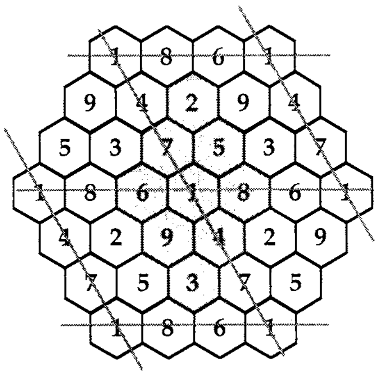


Figure 1. A saturated assignment in $\mathcal{F}_0$ of order 9, with $d_0 = 3$ and $d_1 = 1$

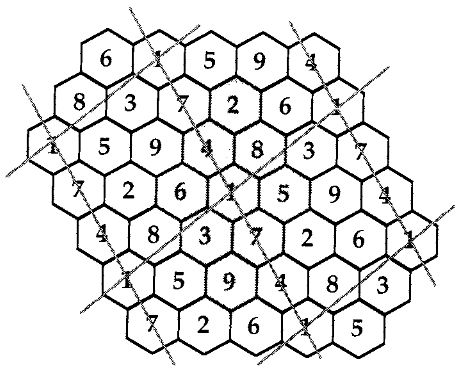


Figure 2. A saturated assignment in $\mathcal{F}_0$ of order 9, with $d_0^2 = 7$ and $d_1^2 = 3$

progress by imposing some added structure. The remainder of this paper concentrates on the regular hexagonal geometry mentioned above, which provides an excellent testbed for channel assignment techniques, and has the added advantage of already being widely used in industry. When considering a regular geometry, it is natural to construct assignments by copying a basic tile, which in the hexagonal case would be a polyhex (a rigid arrangement of hexagons, joined by their edges). Each channel in the image $\mathcal{A}(\mathcal{T})$ appears in the tile exactly once. Laying down sufficiently many copies allows the assignment to be extended to arbitrarily large spatial regions.

The tilings considered here are *regular*, meaning that all tiles have the same orientation, with the set of displacements between pairs of tiles taking the form $\{j\boldsymbol{v}_1 + k\boldsymbol{v}_2 : (j,k) \in \boldsymbol{Z}^2\}$ for some fixed vectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ (see, for example, [2] for a discussion of different classes of tiling). They may be thought of as making up a special set of assignments, $\mathcal{F}_0$. The definitions of optimal and saturated assignments are then modified by making a restriction to $\mathcal{F}_0$: for example, an assignment $\mathcal{A} \in \mathcal{F}_0$ is said to be *optimal in* $\mathcal{F}_0$ if there is no feasible $\mathcal{A}' \in \mathcal{F}_0$ with $n(\mathcal{A}') < n(\mathcal{A})$. The computational difficulty of the general problem stems to a large extent from the prohibitively large search spaces, and so a main theme of this work is to identify families of assignments contained in $\mathcal{F}_0$, which are described by small numbers of parameters, but which include optimal assignments for large classes of real problems.

Figures 1 and 2 show two assignments, each of order 9, that are saturated in $\mathcal{F}_0$. The shaded portion makes up the basic tile. In Figure 1, $d_0 = 3$ and $d_1 = 1$, while in Figure 2, $d_0 = \sqrt{7}$ and $d_1 = \sqrt{3}$. These examples will be used in later sections as illustrations for general techniques.

3.2 The co-channel lattice

The characterizing property of $\mathcal{F}_0$ is that channels are reused at the vertices of a two-dimensional lattice, called the co-channel lattice. In Figure 1, the lattice for channel 1 has been superimposed on the two assignments. Construction of assignments may be considered in two parts: defining the co-channel lattice and associated polyhex, and then laying down channels within each tile. The first stage, described in this section, is completely solved; the second, discussed in the next section, is very much still open, although results so far are very encouraging.

Regular geometries generally put restrictions on the values that can be taken by the characteristic distances. Suppose that, in the hexagonal cellular system, the centres of adjoining cells are unit distance apart; then the square of any characteristic distance is a rhombic number, i.e. $d_i^2 = p^2 + pq + q^2$ for some integer p and q. In calculations, it is useful to have the following characterization of the rhombic numbers:

Lemma 1. *A (positive) integer r is rhombic if and only if, after removing all square factors, its prime decomposition contains no prime other than 3 and primes of the form $6k + 1$.*

When characteristic distances are restricted to some set of values in this way, one can, without loss of generality, impose the same restriction on the constraints D_i.

3.3 Quadratic forms and the master equation

There is a correspondence between co-channel lattices and reduced binary quadratic forms of discriminant $-3m(\mathcal{A})^2$. The following remarks are intended to illustrate, rather than derive, this connection. The detailed derivation appears in [3]. Explicitly, the forms of interest are

$$f(x,y) = d_0^2x^2 + \beta xy + \gamma y^2 , \tag{3.1}$$

where, to ensure the correct discriminant,

$$\beta^2 - 4\gamma d_0^2 + 3m^2 = 0 . \tag{3.2}$$

Generally, reduced forms are given by $f(x,y) = \alpha x^2 + \beta xy + \gamma y^2$, where either $-\alpha < \beta \leq \alpha < \gamma$ or $0 \leq \beta \leq \alpha = \gamma$. The additional requirement here is that α is rhombic, since it must be equal to d_0^2. Moreover, one can also fix the sign of β to be positive, say. (In general, reduced forms come in pairs, with equal and opposite values of β; flipping the sign corresponds to an overall reflection, which is not important in the assignment problem.)

Equation 3.2 acts as a master equation for the generation of co-channel lattices. It forces β to have the same parity (odd or even) as m; also, γ is necessarily rhombic. Using the defining property of reduced forms, it is straightforward to show that $\beta \in [0, d_0^2]$ and $d_0^2 \leq [m]_R$, where $[x]_R$ denotes the largest rhombic number no greater than x. The upshot is that for a fixed order of assignment, m, the possible co-channel lattices are identified by looking for solutions (β, γ, d_0^2) of Equation 3.2 satisfying

$$0 \leq \beta \leq d_0^2 \leq [m]_R \text{ and } \gamma \geq d_0^2. \tag{3.3}$$

For example, Figure 1 has $(\beta, \gamma, d_0^2) = (9, 9, 9)$ and Figure 2 has $(\beta, \gamma, d_0^2) = (3, 9, 7)$. To carry through an explicit construction, these triplets must be translated into a pair of basis vectors (v_1, v_2) for the co-channel lattice. This is a straightforward task, described in [3].

3.4 Tile construction

Each possible co-channel lattice will admit various shapes of polyhexagonal tile from which the assignments can be built. The precise choice of tile is unimportant: it is the underlying co-channel lattice that determines the characteristic distance d_0, and the tile simply provides a framework in which to explore specific channel patterns. Nevertheless, a good choice of tile can help illuminate the relative merits of different algorithms.

A sensible way forward is to construct a polyhex $\mathcal{P}$ around each vertex $\mathcal{V}$ of the co-channel lattice, as follows: $\mathcal{P}$ includes every cell strictly closer to $\mathcal{V}$ than to any other vertex, and, conversely, includes no cell strictly closer to another vertex than to $\mathcal{V}$. (The distance to a cell is always understood to be the distance to its centre.) There is some freedom when there are cells equidistant from $\mathcal{V}$ and another vertex; these choices are resolved by including cells to make up a total of m (the required order), in such a way that $\mathcal{P}$ contains no pair of cells separated by a co-channel lattice vector. The shaded regions in Figures 1 and 2 are examples of this construction.

4 Regular problems: filling the tile

The previous section reviewed the way in which candidate co-channel lattices are constructed; the result is an understanding of the possible values of the first characteristic distance d_0 for each order m, and the associated shapes of tile. To complete the assignment, the arrangement of channels within each tile must now be specified, and the precise details will determine the values of all the higher characteristic distances d_i $(i > 0)$. As yet, there is no general theory for this phase of the procedure, but one can identify several subsets of the set $\mathcal{F}_0$ of all regular tilings, in an attempt to build up an overall picture. Three of these families are discussed below.

4.1 Successive Maximal Repetition

An assignment $\mathcal{A}$ is said to be generated by *Successive Maximal Repetition* (SMR) if its channels are laid out according to the algorithm shown schematically in Figure 3. The basic ingredients are a set of L nested loops, where L is called the *level* of the assignment. Each loop has a spatial offset α_i $(i = 1, \ldots, L)$, and a positive spectral offset c_i, repeated m_i times, so that the order m satisfies

$$\prod_{i=1}^{L} m_i = m \tag{4.1}$$

and the span is

$$n - 1 = m \sum_{i=1}^{L} \frac{c_i(m_i - 1)}{\prod_{j=1}^{i} m_i} . \tag{4.2}$$

The family of all SMR assignments will be denoted $\mathcal{F}_1$. Clearly $\mathcal{F}_1 \subset \mathcal{F}_0$. A much fuller description of SMR than is possible here appears in [3].

A feature of the SMR algorithm is that it assigns the channels in $\mathcal{A}(\mathcal{T})$ consecutively, in ascending order. At each step, the addition of the offset α_i to the cell position is taken modulo co-channel lattice vectors, so as to define a unique cell within the tile. Moreover, each α_i determines the corresponding m_i, namely m_i is defined to be the number of times that α_i can be applied until a cell is encountered that already has a channel assigned to it. In this sense, a

```
cell_position = origin
channel_number = 1
for $k_L$ = 1 to $m_L$
  for $k_{L-1}$ = 1 to $m_{L-1}$
    ....
      for $k_2$ = 1 to $m_2$
        for $k_1$ = 1 to $m_1$
          assign channel_number to cell_position
          if $k_1<m_1$ {cell_offset = $\alpha_1$; channel_offset = $c_1$}
          else if $k_2<m_2$ {cell_offset = $\alpha_2$; channel_offset = $c_2$}
          ....
          else if $k_{L-1}<m_{L-1}$ {cell_offset = $\alpha_{L-1}$; channel_offset = $c_{L-1}$}
          else {cell_offset = $\alpha_L$; channel_offset=$c_L$}
          cell_position = cell_position + cell_offset
          channel_number = channel_number + channel_offset
        next $k_1$
      next $k_2$
    ....
  next $k_{L-1}$
next $k_L$
```

Figure 3. Channel assignment by successive maximal repetition

new level is introduced into the assignment only when required. An important consequence is that the number of levels does not increase as n becomes large, i.e. the number of parameters needed to describe SMR assignments remains small. Explicit studies, such as that outlined in Section 5 below, have so far not required assignments with more than four levels.

If an assignment has only one level, then it is called *balanced.* This usage is consistent with previous work (see for example [4] and [5]), in which balanced assignments are characterised by fixed channel progressions (modulo m) along each direction in the cellular structure. Roughly speaking, in a balanced assignment, all channels in $\mathcal{A}(\mathcal{T})$ have the same relationship with their neighbours. More precisely, in terms of the general formulation of Section 2, consider a fixed (but arbitrary) spectral separation of i units between pairs of transmitters (t_1, t_2); if $\mathcal{A}$ is balanced, then the values of the geographical separation $\delta(t_1, t_2)$ that appear in Equation 2.1 are independent of the channel $\mathcal{A}(t_1)$ assigned to t_1. Figure 2 is an example of a balanced assignment, but Figure 1 is not (it has two levels).

If more than one level is used then a degree of inhomogeneity is introduced, in that generally the $\delta(t_1, t_2)$ appearing in Equation 2.1 are dependent on $\mathcal{A}(t_1)$. For example, consider the sites t_1 in Figure 1 with $\mathcal{A}(t_1) = 3$; there exists t_2 with $|\mathcal{A}(t_1) - \mathcal{A}(t_2)| = 1$ and $\delta(t_1, t_2) = 1$. On the other hand, if $\mathcal{A}(t_1) = 2$, say, then there is no such t_2.

The innermost loop in Figure 3 has a precise geometric interpretation: it traces out a group orbit in the set of cells making up the basic tile, under the action of the group G generated by the displacement α_1. Hence every assignment is really formed by tracing out in turn the orbits of G. The higher offsets $\alpha_2, \alpha_3, \ldots, \alpha_L$ serve only to specify the order in which the orbits are traced out and the starting point in each one.

Note that the SMR algorithm, as presented here, is a generalization of the version originally introduced in [3], which in turn is here called "local no-hole SMR" and described in Section 4.3.

4.2 No-hole SMR

As remarked in Section 2, the general assignment problem does not require that all channels $1, \ldots, n(\mathcal{A})$ be used. However, if they are all used then the assignment is said to be *no-hole*, in analogy with the terminology that has been used in graph-colouring problems [6]. The no-hole SMR assignments are generated by the algorithm of Figure 3, but with all the c_i set equal to 1, so that no channels are omitted. They form a subset of $\mathcal{F}_1$, which will be denoted by $\mathcal{F}_2$.

4.3 Local no-hole SMR

There is a final special case that is of interest here, producing a further subfamily of the SMR assignments, which will be denoted by $\mathcal{F}_3$; it is important because it is relatively easy to analyse in detail, while at the same time containing assignments that reproduce (and often improve upon) the results of earlier work. The motivation and definition for $\mathcal{F}_3$ are as follows. This is the version of the SMR algorithm that was described in detail in [3].

Recall that for a general SMR assignment, the offsets α_i are free, but each one determines the number of repetitions m_i. The family $\mathcal{F}_3$ comes with an additional prescription for the α_i also, which is very natural if, for given co-channel lattice, one's main aim is to minimize interference between transmitters operating on adjacent channels. To be explicit, each time a new offset is required, it is chosen to be as long as possible, i.e. it is chosen to be the displacement from the starting cell (called the origin in Figure 3) to the furthest cell in the basic polyhex without a channel currently assigned. If there are several equidistant possibilities then one with maximal m_i is chosen. Proceeding along these lines tends to maximise the characteristic distance d_1, since successive channels are always placed as far apart as possible. Moreover, any additional freedom is used to maximize m_i, therefore tending to control the number of levels.

The adjective "local" is used to describe this process because each offset is generated only when needed, and without any reference to how the choice of future offsets might be affected. The advantage of such a rule is that it is efficient to implement and amenable to theoretical analysis (see Section 5 and also [3]); but it may be that a more global construction, in which all offsets are determined collectively, might be better. More precisely, one can ask whether the saturated assignments in $\mathcal{F}_3$ are also saturated in $\mathcal{F}_2$.

4.4 Summary

To summarize, we have defined four families of assignment:

- $\mathcal{F}_0$: all assignments by regular tiling;
- $\mathcal{F}_1$: all regular tilings with successive maximal repetition;
- $\mathcal{F}_2$: all regular tilings with no-hole SMR;
- $\mathcal{F}_3$: all regular tilings with local no-hole SMR.

Clearly $\mathcal{F}_3 \subset \mathcal{F}_2 \subset \mathcal{F}_1 \subset \mathcal{F}_0$. An important question when one has a nested sequence of families like this is to ask which saturated assignments in the smaller ones are also saturated in the larger ones.

The family $\mathcal{F}_3$ attempts to maximize d_1 for given d_0 and is therefore more restricted in scope than the others. For instance, it would not be expected, in general, to include optimal assignments when there are non-zero constraints D_i with $i > 1$. However, even problems with only D_0 and D_1 non-zero have not previously been fully understood, and so $\mathcal{F}_3$ is certainly still of interest.

As a final remark, there is, in addition to the simple bound $d_0^2 \leq [m]_R$, a bound on d_1 in terms of the co-channel parameters (β, γ, d_0^2). It is calculated by considering Dirichlet regions in the co-channel lattice (see [3]) and (subject to the mild restriction $d_0 \geq \frac{1}{2}n$) given by

$$d_1^2 \leq \left[d_0^2 \left(\frac{1}{4} + \frac{1}{12} \left(\frac{2\gamma - \beta}{m} \right)^2 \right) \right]_R \tag{4.3}$$

5 Some results and areas for further work

5.1 Detailed study of the family $\mathcal{F}_3$

The local no-hole SMR assignments that make up $\mathcal{F}_3$ have the attraction of being efficient to compute and therefore amenable to detailed analysis. Reference [3] describes such a study, the results of which are summarized here.

First it is sensible to impose a lower bound on d_0^2/m, since otherwise the basic tile is allowed to become long and thin, which tends to preclude saturated assignments. In [3], $d_0^2 \geq \frac{1}{2}m$; there are 1077 such assignments in $\mathcal{F}_3$ with $m \leq 125$, and it takes of the order of 100 seconds on a workstation to compute them all. Figure 4 picks out the particular assignments with maximal d_0 for each order m. The values of d_0^2 and d_1^2 are plotted, together with the bound (Equation 4.3). There is the linear trend one would expect, together with fluctuations caused by the discrete nature of the underlying geometry.

The main purpose of Figure 4 is to compare with the results obtained by Gamst [7], who performed similar calculations (see especially Figure 9 in [7]). Gamst's methods look for no-hole regular tilings (a family containing $\mathcal{F}_2$), but are restricted to rhombic values of m and co-channel lattices with $d_0^2 = m$,

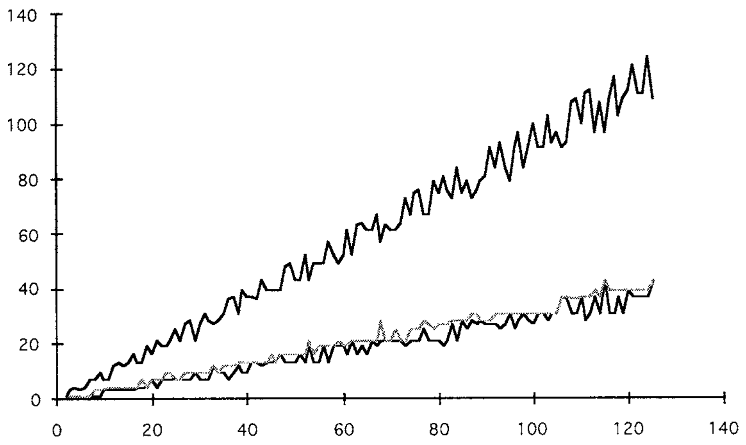


Figure 4. Summary of the assignments in $\mathcal{F}_3$ with maximal d_0 for each order; the upper and lower curves show the values of d_0^2 and d_1^2, respectively, against m. The upper bound (Equation 4.3) on d_1^2 is also shown

corresponding to the solution $(\beta, \gamma, d_0^2) = (m, m, m)$ in Equation 3.2. Local no-hole SMR is much more flexible, in that it allows non-rhombic m and all possible co-channel lattices. However, Gamst guaranteed the highest possible d_1; in other words he produced assignments that are saturated in the set of no-hole regular tilings. It turns out that $\mathcal{F}_3$ reproduces all of Gamst's figures exactly. This shows that $\mathcal{F}_3$ deserves further study, and in particular that many assignments saturated in $\mathcal{F}_3$ are also saturated in $\mathcal{F}_2$. Indeed, returning to the question posed at the end of Section 4.3, it seems reasonable to conjecture that *all* assignments saturated in $\mathcal{F}_3$ are also saturated in $\mathcal{F}_2$.

5.2 A no-hole assignment that is not optimal

As yet, there are no significant general results concerning assignments in which $n(\mathcal{A}) \neq m(\mathcal{A})$. The following example is not the simplest possible, but it illustrates several of the issues involved. Roughly speaking, when trying to reduce interference levels, there is a trade-off between geographical separation and spectral separation. Assignments that do not use all available channels have smaller order than the maximum consistent with their span, and hence tend to reduce geographical separations; but at the same time spectral separations are increased, and the combined effect might be beneficial.

To be specific, look for assignments that are optimal for the set of constraints

$$D_0 = 3,\ D_1 = \sqrt{3},\ D_i = 0\ (i > 1)\,. \tag{5.1}$$

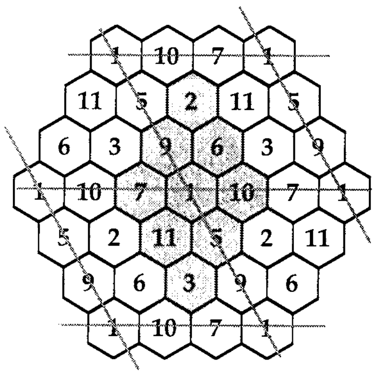


Figure 5. An optimal assignment in $\mathcal{F}_1$ for the constraints (Equation 5.1)

Restricting to the family $\mathcal{F}_3$, the optimal assignment, $\mathcal{A}_3$, has order 12 and span 11 and is well known (see both [7] and [8]). However, $\mathcal{F}_1$ contains an assignment $\mathcal{A}_1$, shown in Figure 5, which satisfies the constraints (Equation 5.1) and has order 9 and span 10. Although $\mathcal{A}_3$ is saturated in both $\mathcal{F}_3$ and $\mathcal{F}_1$, it is only in $\mathcal{F}_3$ that it is optimal for the constraints (Equation 5.1).

In terms of SMR parameters, $\mathcal{A}_1$ has two levels, with $m_1 = m_2 = 3$, $c_1 = 1$ and $c_2 = 2$. It has the same co-channel structure as Figure 1, but with a skipped channel at the end of each group orbit, i.e. at the end of each execution of the innermost loop. A similar idea was suggested by Prosch in [9], but not developed.

5.3 An example with more constraints

Previous results in this section have concentrated on the characteristic distances d_0 and d_1, effectively assuming that $D_i = 0$ for $i > 1$. As a final example, consider the more complicated set of constraints

$$D_0 = \sqrt{13},\ D_1 = \sqrt{3},\ D_2 = \sqrt{3},\ D_i = 0\ (i > 2)\,. \tag{5.2}$$

The family $\mathcal{F}_3$ is not appropriate here, because of the non-zero value of D_2 (although $\mathcal{F}_3$ does contain an assignment that would be optimal in $\mathcal{F}_0$ if D_2 were zero). It is not yet known how to incorporate the constraints D_2 and above into a general theory, but case-by-case analysis is still possible. Here, since $D_0 = \sqrt{13}$, any feasible regular tiling must have order at least 13. Within the no-hole SMR family $\mathcal{F}_2$, an exhaustive search reveals that 15 is the minimal order for Equation 5.2 to be satisfied. The corresponding optimal assignment is shown in Figure 6.

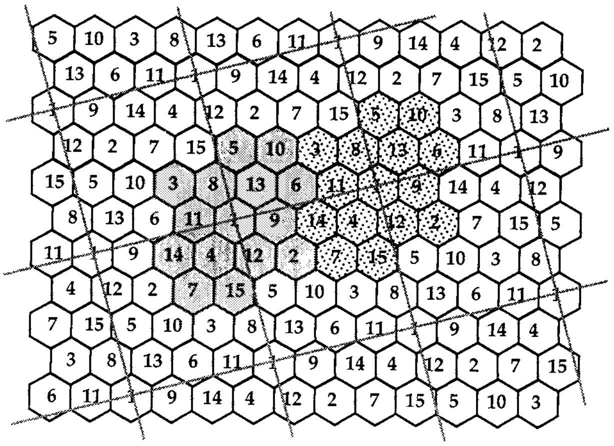


Figure 6. An optimal assignment in $\mathcal{F}_2$ for the constraints (Equation 5.2)

6 Summary and closing remarks

It is clear that combinatorial methods can address many problems in radio channel assignment within a unified framework. The regular tilings contained in $\mathcal{F}_0$ are very natural in design and planning work, and considerable progress is possible by restricting attention to various well-motivated subfamilies, described by small numbers of parameters. In particular, the SMR algorithm is very promising, and for local no-hole SMR detailed analysis already exists. However, the last two examples in Section 5 highlight areas in which further research is needed.

Radio channel assignment does not necessarily have to be addressed using the methods discussed here, and there is much to be gained from looking at alternatives. Formulations using graph colouring [1] can potentially provide useful checks and comparisons, as can approaches using large-scale optimization techniques, such as simulated annealing [10] and tabu search [11].

Finally, there are several directions in which the problem as stated in Section 2 can be extended to include further aspects of real systems. For example, in many problems each cell must be assigned several channels, rather than a single one as assumed here; this introduces extra, *frequency-only* constraints [12] to control effects such as intermodulation products. Another major issue concerns inhomogeneities, which tend to destroy the structure of a regular system. Cellular assignments can be distorted by localized regions of high demand in population centres, or by variations in terrain. It is worth trying to describe such effects by adding extra parameters to regular theories.

References

1. Roberts, F.S. (1991). T-colorings of graphs: recent results and open problems. *Discrete Mathematics*, **93**, 229–245.

2. Beauquier, D. and Nivat, M. (1991). On translating one polyomino to tile the plane. *Discrete and Computational Geometry*, **6**, 575–592.

3. Leese, R.A. (1994). A unified approach to the assignment of radio channels on a regular hexagonal grid. Submitted to *IEEE Transactions on Vehicular Technology*.

4. Arnaud, J.-F. (1980). Frequency planning for broadcast services in Europe. *Proc. IEEE*, **68**, 1515–1522.

5. Wang, S.-W. and Rappaport, S.S. (1989). Signal-to-interference calculations for balanced channel assignment patterns in cellular communications systems. *IEEE Transactions on Vehicular Technology*, **37**, 1077–1087.

6. Roberts, F.S. (1993). No-hole 2-distant colorings. *Mathematical and Computer Modelling*, **17**, 139–144; Sakai, D. and Wang, C. (1993). No-hole $(r+1)$-distant colorings. *Discrete Mathematics*, 175–189.

7. Gamst, A. (1982) Homogeneous distribution of frequencies in a regular hexagonal cell system. *IEEE Transactions on Vehicular Technology*, **31**, 132–144.

8. MacDonald, V.H. (1979). The cellular concept. *Bell System Technical J.*, **58**, 15–41.

9. Prosch, T.A. (1991). A possible frequency planning method and related model calculations for the sharing of VHF band II (87.5–108 MHz) between FM and DAB (digital audio broadcast) systems. *IEEE Transactions on Broadcasting*, **37**, 55–63.

10. Duque-Antón, M., Kunz, D. and Rüber, B. (1993). Channel assignment for cellular radio using simulated annealing. *IEEE Transactions on Vehicular Technology*, **42**, 14–21.

11. Castelino, D.J., Hurley, S. and Stephens, N.M. (1994). A tabu search algorithm for frequency assignment. To appear in *Annals of Operations Research*.

12. Hale, W.K. (1980). Frequency assignment: theory and applications. *Proc. IEEE*, **68**, 1497–1514.

Interconnection Networks Based on Two-dimensional de Bruijn Graphs

Kenneth G. Paterson[1]

Signal and Information Processing Laboratory, Swiss Federal Institute of Technology, Zurich, Switzerland

Abstract

A family of directed graphs generalising the de Bruijn graphs and retaining many of their attractive features are proposed as interconnection networks. For $k \geq 2$ and $u, v \geq 1$, the graph $B(k, u, v)$ has k^{uv} vertices and out-degree and in-degree equal to $k^u + k^v$. $B(k, u, v)$ has diameter equal to $\min(u, v)$ and connectivity at least $c = k^u + k^v - kuv - 2$. In the presence of up to $c - 1$ faulty vertices, the diameter of the graph increases to at most $\max(u, v) + 1$. A simple fault-tolerant routing algorithm is given for $B(k, u, v)$, and the graphs are shown to have a range of computationally useful sub-graphs, including complete trees, rings and meshes, all of large size.

1 Introduction

An interconnection network is a means of connecting together a large number of Processing Elements (PEs) by communication links. The careful design of such networks is of fundamental importance in building efficient, general-purpose parallel computers. The performance and characteristics of an interconnection network are traditionally studied in terms of properties of the underlying graph $G = (V, E)$: the vertices V of G represent the PEs and the edges E of G represent communication links between PEs. G is directed or undirected according to whether the links are uni- or bi-directional. Some standard graph theoretical parameters are of immediate practical interest: referring to [5] for definitions, the diameter of G is related to the worst-case communication delay between two PEs, while the degree (or out-degree in the directed case) is just the number of links emanating from the corresponding PE. The connectivity of the graph G is a measure of the network's fault-tolerance: if the connectivity of the graph is c, then a communication path can be maintained between any two non-faulty PEs in the presence of up to $c - 1$ PE failures.

[1]This work was supported by The Royal Society through its European Science Exchange Programme and the Swiss National Science Foundation.

The mathematical problem of constructing regular directed and undirected graphs with a large number of vertices for a given degree and diameter has received much attention (see [1] for a survey and [2] for recent results on this, the (d, k)-graph problem). This problem corresponds to designing networks with a large number of PEs for a fixed number of communication links per PE and a given worst case delay. At the same time, of practical importance is the problem of finding networks that have simple message routing algorithms, both in the fault-free and faulty cases. Equally, parallel algorithms are often tailored to a particular topology. For example, distributed searching and arithmetic algorithms may be suited to a tree structure, [27], while it is known that the FFT is well-adapted to implementation on a shuffle-exchange network [31] or a hypercube [6]. Other desirable network topologies are rings, meshes and Tree Machines [27]. To build computers capable of supporting many types of parallel algorithm, we therefore need to find graphs G in which such computationally useful topologies can be found as subgraphs or on which they can be emulated efficiently (see [25] for a discussion of the concept of emulation).

The family of de Bruijn graphs have very good performance with respect to the requirements outlined above. The de Bruijn graph $B(k, u)$ is a directed graph with vertices the set of k-ary u-tuples. There is an edge from vertex $(x_1, x_2, \ldots, x_u)$ to every vertex of the form $(x_2, \ldots, x_u, x_{u+1})$ so that $B(k, u)$ has k^u vertices, in-degree and out-degree equal to k and diameter u. This is the minimum possible diameter for a directed graph with out-degree k and k^u vertices [15]. It is also known that the connectivity of $G(k, u)$ is equal to $k - 1$ [16,29], the highest possible value given that the graph has loop-edges. The diameter increases by only one in the presence of up to $k - 2$ faulty vertices [29]. Optimal fault-tolerant routing algorithms based on string-matching have been developed for a large family of graphs which includes the de Bruijn graphs in [30]. The fault-tolerance of the de Bruijn graph was also the subject of [13]. In [27], it was shown that the undirected version of the de Bruijn graph (obtained by removing the orientation of edges of the de Bruijn graphs, see [8,23]) contains rings, complete binary trees and Tree Machines as subgraphs and can emulate shuffle-exchange networks. The de Bruijn graphs and their undirected relatives therefore give suitable topologies for implementing a large range of parallel algorithms. Because of all of these desirable properties, the de Bruijn graphs are held to be strong competitors with the hypercube graphs as interconnection networks [3,27].

A number of authors have already considered generalisations of the de Bruijn graphs as interconnection networks. In [14,15,24] families of graphs with arbitrary degree and number of vertices, minimum or close to minimum diameter, high connectivity [16] and containing the standard de Bruijn graphs as a subfamily were introduced. In [7] and [28] these graphs were shown to contain trees and cycles as subgraphs. As yet, however, it appears that no fault-tolerant routing algorithms have been developed for these graphs, and they are not known to have the important property of containing meshes as subgraphs. In a different direction, the properties of the undirected graph obtained from the direct product of two graphs $B(2, m)$ and $B(2, n)$ were examined in [25]. The so-called order-(m,n) Product Shuffle (PS) network has degree 8, diameter $m+n$ and 2^{m+n} vertices. In [25], the PS networks were shown to be capable of efficiently emulating other important networks. They were also shown to contain as subgraphs large binary trees, meshes and meshes of trees.

Here we study a somewhat different generalisation of the de Bruijn graphs — two-dimensional de Bruijn graphs. Our graphs appear to have been first considered by Fan et al. [11] in the binary case, in the context of constructing Perfect Maps (see Section 4 below). We describe our graphs and derive their basic properties (including a lower bound on their connectivity) in Section 2. We obtain simple fault-tolerant routing algorithms in Section 3. In Section 4 we show how to construct cycles, trees and meshes of large sizes as subgraphs of our graphs. We conclude with a discussion of the strengths and weaknesses of our graphs relative to other popular interconnection networks and by listing some topics for future exploration.

2 Two-dimensional de Bruijn graphs

Let $u,v \geq 1$ and $k \geq 2$. The vertex set V of the graph $B(k,u,v)$ is the set of $u \times v$ matrices with entries from $\{0,1,\dots,k-1\}$ (so $B(k,u,v)$ has k^{uv} vertices). $B(k,u,v)$ has two edge sets, labelled E_1 and E_2 and defined as follows. Let

$$A = [\,C_1, C_2, \dots, C_v\,] = \begin{bmatrix} R_1 \\ R_2 \\ \vdots \\ R_u \end{bmatrix}$$

be a vertex of $B(k,u,v)$ with columns $C_1, C_2, \dots C_v$ and rows $R_1, R_2, \dots, R_u$. There is a directed edge in E_1 from A to every vertex of the form

$$[\,C_2, \dots, C_v, C_{v+1}\,]$$

and a directed edge in E_2 from A to every vertex of the form

$$\begin{bmatrix} R_2 \\ \vdots \\ R_u \\ R_{u+1} \end{bmatrix}.$$

Here C_{v+1} is an arbitrary column (i.e. k-ary v-tuple) and R_{u+1} an arbitrary row (i.e. k-ary u-tuple).

Notice that the graph (V, E_1) is isomorphic to the de Bruijn graph $B(k^u, v)$: we merely identify each column of the vertex A with an integer in the range $0, 1, \dots, k^u - 1$. Similarly, (V, E_2) is isomorphic to $B(k^v, u)$. Thus the graph $B(k,u,v)$ has as subgraphs two de Bruijn graphs. Our simple analysis of the properties of the graphs $B(k,u,v)$ hinges on this observation. We can also think of the graph $B(k,u,v)$ as being obtained from $B(k^u,v)$ ($B(k^v,u)$) by adding k^u (respectively, k^v) out-edges at each vertex.

Every vertex of $B(k,u,v)$ is the initial vertex of k^u edges in E_1 and k^v edges in E_2, so every vertex has out-degree $k^u + k^v$. Similarly, every vertex has in-degree $k^u + k^v$. Moreover, since $B(k,u,v)$ contains as sub-graphs $B(k^u,v)$ and $B(k^v,u)$, we see immediately that $B(k,u,v)$ has diameter at most $\min(u,v)$. That this is in fact the diameter is easily seen by considering any path from the all-zero matrix to, say, the all-one matrix.

2.1 Connectivity

We make use of the basic results of [29] on the connectivity of $B(k,u)$. Let $A = (a_1, a_2, \ldots, a_u)$ and $B = (b_1, b_2, \ldots, b_u)$ be distinct vertices in $B(k,u)$. For $0 \le i < k$, let P_i denote the the path

$$\begin{aligned} P_i : (a_2, a_3, \ldots, a_u, i) \quad &\to (a_2, \ldots, a_u, i, b_1) \\ &\vdots \\ &\to (i, b_1, b_2, \ldots, b_{u-1}) \end{aligned}$$

from successor i of A to predecessor i of B. That the connectivity of $B(k,u)$ is $k-1$ is a simple consequence of the following result:

Result A. [29] There is at most one vertex of $B(k,u)$ that simultaneously lies on two distinct paths P_i, P_j.

This result applies to the subgraphs (V, E_1) and (V, E_2) of $B(k,u,v)$ to show that $B(k,u,v)$ has connectivity at least $\max(k^u - 1, k^v - 1)$. This argument does not make full use of the fact that paths with edges from E_1 and E_2 are simultaneously available and the bound can be improved as follows. Let A, B be vertices of $B(k,u,v)$ with the rows and columns of A labelled as before and with

$$B = [\, C_{v+1}, C_{v+2}, \ldots, C_{2v} \,] = \begin{bmatrix} R_{u+1} \\ R_{u+2} \\ \vdots \\ R_{2u} \end{bmatrix}$$

Let $C^0, C^1, \ldots, C^{k^u-1}$ be an ordering of the k-ary u-tuples. For $0 \le i < k^u$, we denote by P_i the path

$$\begin{aligned} P_i : [\, C_2, C_3, \ldots, C_v, C^i \,] \quad &\to [\, C_3, \ldots, C_v, C^i, C_{v+1} \,] \\ &\vdots \\ &\to [\, C^i, C_{v+1}, C_{v+2}, \ldots, C_{2v-1} \,] \end{aligned}$$

from successor i of A to predecessor i of B in the subgraph isomorphic to $B(k^u, v)$. Similarly, for an ordering $R^0, R^1, \ldots, R^{k^v-1}$ of the k-ary v-tuples, we denote by Q_j the path

$$Q_j : \begin{bmatrix} R_2 \\ R_3 \\ \vdots \\ R_u \\ R^j \end{bmatrix} \rightarrow \begin{bmatrix} R_3 \\ \vdots \\ R_u \\ R^j \\ R_{u+1} \end{bmatrix} \rightarrow \ldots \rightarrow \begin{bmatrix} R^j \\ R_{u+1} \\ R_{u+2} \\ \vdots \\ R_{2u-1} \end{bmatrix}$$

from the successor j of A to predecessor j of B in the subgraph isomorphic to $B(k^v, u)$.

Lemma 1. *There are at most kuv pairs of paths P_i, Q_j sharing a common vertex of $B(k, u, v)$.*

Proof. Suppose $1 \leq m < v$, $1 \leq n < u$ and a vertex F of $G(k, u, v)$ occurs both as the m^{th} vertex on path P_i and as the n^{th} vertex on path Q_j for some i, j. Then

$$F = [C_{m+1}, \ldots, C_v, C^i, C_{v+1}, \ldots, C_{v+m-1}] = \begin{bmatrix} R_{n+1} \\ \vdots \\ R_u \\ R^j \\ R_{u+1} \\ \vdots \\ R_{u+n-1} \end{bmatrix}$$

From this we see that every entry of C^i excepting entry $u - n + 1$ is determined by the rows of A and B while every entry of R^j excepting entry $v - m + 1$ is determined by the columns of A and B. Moreover these two undetermined entries must be equal. Since there are k possibilities for this common entry, we see that for each of the uv different choices of m, n there are at most k choices for the vertex F and so at most kuv pairs of paths P_i, Q_j share a vertex.

Lemma 2. *The graph $B(k, u, v)$ has connectivity at least $k^u + k^v - kuv - 2$.*

Proof. Let A and B be, as before, arbitrary vertices of $B(k, u, v)$ and consider the set of $k^u + k^v$ paths P_i, Q_j. We show that a subset of $k^u + k^v - kuv - 2$ of these paths are vertex disjoint. The bound on connectivity quickly follows from this.

Any vertex F on a path P_i or Q_j may lie on just one of the $k^u + k^v$ paths, or it could lie on two paths P_a, P_b according to Result A, on two paths Q_c, Q_d, on some pairs of paths (P_i, Q_j) according to Lemma 1, or some combination of these possibilities could occur. We consider the case where some vertex F actually does lie on two paths P_a, P_b and two paths Q_c, Q_d. By Result A, F is the only vertex with this property and P_a, P_b, Q_c, Q_d are distinct paths. Notice that the four pairs (P_a, Q_c), (P_a, Q_d), (P_b, Q_c), (P_b, Q_d) are of the type considered in Lemma 1. Thus there remain at most $kuv - 4$ pairs of paths P_i, Q_j sharing common vertices. The vertices shared in the pairs are all distinct from one another and from F (otherwise we would have more than one vertex lying on two paths P_a, P_b, contradicting Result A). This leaves at least $k^u + k^v - 4 - 2(kuv - 4) = k^u + k^v - 2kuv + 4$ vertex disjoint paths of the types P_i, Q_j. Taking all of these paths, one path from the four paths P_a, P_b, Q_c, Q_d and one path from each of the at most $kuv - 4$ pairs, we obtain at least $k^u + k^v - kuv + 1$ vertex disjoint paths of the types P_i, Q_j.

A careful analysis of the other cases, carried out using similar arguments, shows that at least $k^u + k^v - kuv - 2$ of the paths P_i, Q_j are vertex disjoint in every case.

Notice that we have only considered a special set of paths in proving Lemma 2. These paths will be put to good use in our routing algorithms in the next section, but notice that considering a more complex set of paths could lead to a better value for the connectivity of $B(k, u, v)$. Indeed Blackburn [4] has produced an asymptotically superior lower bound of $k^u + k^v - 12k - 20$ in this way. Of course the connectivity of $B(k, u, v)$ is at most $k^u + k^v - k - 1$, since this is the number of distinct successors of the all-zero vertex. We leave as an important outstanding problem the determination of the exact value of the connectivity of $B(k, u, v)$.

3 Routing in the graphs $B(k, u, v)$

Routing of messages is trivial in the absence of faults. Given two vertices A and B with rows and columns as before, the PE corresponding to A can easily compute *the E_1 and E_2 canonical paths from A to B*

$$\begin{aligned} A &\rightarrow [C_2, C_3, \ldots, C_v, C_{v+1}] \\ &\vdots \\ &\rightarrow [C^v, C_{v+1}, C_{v+2}, \ldots, C_{2v-1}] \\ &\rightarrow B \end{aligned}$$

and

$$A \rightarrow \begin{bmatrix} R_2 \\ R_3 \\ \vdots \\ R_u \\ R_{u+1} \end{bmatrix} \rightarrow \ldots \rightarrow \begin{bmatrix} R_u \\ R_{u+1} \\ R_{u+2} \\ \vdots \\ R_{2u-1} \end{bmatrix} \rightarrow B.$$

These paths require v and u hops, respectively. Shorter paths may be available if vertices A and B have an "overlap", i.e. share common rows or columns. The communication load on links and PEs can be balanced by using each canonical path half of the time. The average number of hops needed to route a message is then $\frac{1}{2}(u+v)$.

Suppose now that up to $k^u + k^v - kuv - 3$ PEs are faulty. Let A and B be non-faulty vertices and consider the $k^u + k^v$ paths from A to B obtained by traversing an edge from A to one of its successors then the appropriate path P_i or Q_j, then finally an edge from a predecessor of B to B itself. The proof of Lemma 2 guarantees that at least one of these paths is fault-free and so the diameter of the graph increases to at most $\max(u, v) + 1$ in the presence of up to $k^u + k^v - kuv - 3$ faulty vertices. If PE A has available the list of faulty vertices, then A can find a fault-free route to B by simply testing the vertices of each of the paths in turn until a fault-free path is found. An optimally efficient method for testing paths against a fault-set can be obtained by an obvious extension of the string-matching techniques of [30].

The reader will notice that we have not considered methods for fault diagnosis or the transmission of fault-information. In common with [30] we assume that some underlying mechanism is available to perform these tasks.

4 Some important subgraphs of $B(k, u, v)$

In this section we show that the family of graphs $B(k, u, v)$ contain as subgraphs cycles, trees and meshes, all of large sizes. Cycles (and paths derived from them) are important topologies for solving pipeline-type problems (for example matrix-vector multiplication, recurrence evaluation, some kinds of single input/single output sorting [27]. Trees, particularly complete binary trees, allow efficient algorithms for many problems in the multiplex class of problems (for example evaluation of general arithmetic functions, parallel-input, single-output sorting [27]). Meshes are widely used in numerical and linear-algebraic algorithms [25].

4.1 Cycles

$B(k, u, v)$ contains as edge-disjoint subgraphs $B(k^u, v)$ and $B(k^v, u)$. These are Hamiltonian graphs on k^{uv} vertices (note that Hamiltonian cycles correspond to k^u-ary span v and k^v-ary span u de Bruijn sequences respectively, see [12] for further discussion). Therefore $B(k, u, v)$ contains pairs of edge-disjoint cycles of period k^{uv} (i.e. having k^{uv} vertices). We can improve this result as follows.

Result B. [17] For $k \geq 2$ and every $u \geq 1$, $B(k, u)$ is pancyclic, i.e. contains cycles of period t, for every $1 \leq t \leq k^u$.

An algorithm which generates cycles of arbitrary period in $B(2, u)$ was presented in [9].

Corollary. Suppose $k \geq 2$ and $u, v \geq 1$. Then for every choice of t_1 and t_2 with $1 \leq t_1, t_2 \leq k^{uv}$, the graph $B(k, u, v)$ contains a pair of edge disjoint cycles of periods t_1 and t_2.

The question of finding cycles and paths in $B(k,u)$ in the event of edge failures has recently been examined in [26]. There the following result was proved.

Result C. [26] Suppose k is a prime-power and $u \geq 1$. Then the de Bruijn graph $B(k,u)$ contains k edge disjoint cycles of period $k^u - 1$. Therefore, in the event of up to $k-1$ edge failures, $B(k,u)$ still contains a cycle of period $k^u - 1$.

We can adapt this result to $B(k,u,v)$, once again by using the edge-disjoint subgraphs $B(k^u, v)$ and $B(k^v, u)$.

Corollary. Suppose k is a prime-power and $u, v \geq 1$. Then the graph $B(k,u,v)$ contains $k^u + k^v$ edge-disjoint cycles of period $k^{uv} - 1$. Therefore, in the event of up to $k^u + k^v - 1$ edge failures, $B(k,u,v)$ still contains a cycle of period $k^{uv} - 1$.

The problem of finding large cycles in de Bruijn graphs in the presence of vertex failures was also addressed in [26]. These results can be adapted to the family $B(k,u,v)$ too.

4.2 Trees

Since $B(k,u,v)$ is a directed graph, we consider both trees in which every edge is directed towards the root vertex and in which every edge is directed away from the root vertex. Our results are trivially adapted to give large trees in the undirected version of $B(k,u,v)$ (obtained by removing the orientation of the edges of the graph).

Definition 3. *(c.f. [25]) The* complete k-ary out-directed tree of height n, *denoted $CTO(k,n)$, is a graph with $\frac{k^{n+1}-1}{k-1}$ vertices labelled by the k-ary strings of length at most n. There is a single* root vertex *at level* 0, *labelled by the null string. For $0 \leq t \leq n-1$ and each $0 \leq x_t < k$, there is a directed edge from vertex $(x_0, x_1, \ldots, x_{t-1})$ at level t to vertex $(x_0, x_1, \ldots, x_{t-1}, x_t)$ at level $t+1$. The complete k-ary in-directed tree $CTI(k,n)$ is identical to $CTO(k,n)$ except that the orientation of the edges are reversed.*

It is well-known [27] that the graph $B(2,u)$ contains as subgraphs $CTO(2,u-1)$ and $CTI(2,u-1)$. For example, we can take vertex $(0^{u-1},1)$ as root vertex and directed edges from each vertex $(0^{u-t-1},1,w_1,w_2,\ldots,w_t)$ to vertices $(0^{u-t-2},1,w_1,w_2,\ldots,w_t,0)$ and $(0^{u-t-2},1,w_1,w_2,\ldots,w_t,1)$ to obtain a subgraph isomorphic to $CTO(2,u-1)$. Here 0^j denotes a string of zeros of length j. Similarly, the vertex $(1,0^{u-1})$ is the root vertex of a subgraph isomorphic to $CTI(2,u-1)$ — this time there are edges from vertices of the form $(0,w_1,w_2,\ldots,w_t,1,0^{u-t-2})$ and $(1,w_1,w_2,\ldots,w_t,1,0^{u-t-2})$ to vertex $(w_1,w_2,\ldots,w_t,1,0^{u-t-1})$. These facts are easily extended to the k-ary case to show that $B(k,u)$ contains a set of $k-1$ vertex-disjoint $CTO(k,u-1)$ and a set of $k-1$ vertex-disjoint $CTI(k,u-1)$ as subgraphs: we take as root vertices

$(0^{u-1}, i)$, $1 \leq i < k$ and $(j, 0^{u-1})$, $1 \leq j < k$, respectively. Notice that each of these sets of trees makes use of every vertex of the graph except the vertex (0^u). On considering the subgraphs $B(k^u, v)$ and $B(k^v, u)$ of $B(k, u, v)$ we immediately obtain the following.

Lemma 4. *The graph $B(k, u, v)$ contains a set of $k^u - 1$ vertex-disjoint $CTO(k^u, v-1)$ and a set of $k^u - 1$ vertex-disjoint $CTI(k^u, v-1)$ as subgraphs. $B(k, u, v)$ also contains a set of $k^v - 1$ vertex-disjoint $CTO(k^v, u-1)$ and a set of $k^v - 1$ vertex-disjoint $CTI(k^v, u-1)$ as subgraphs.*

Since the trees in each set in Lemma 4 are vertex disjoint, we can guarantee that such trees can still be found in the presence of faulty vertices. For example, $B(k, u, v)$ still contains a $CTO(k^u, v-1)$ and a $CTI(k^u, v-1)$ in the event of up to $k^u - 2$ vertex failures.

Lemma 4 shows that moderately tall complete trees can be found in $B(k, u, v)$. These trees have quite high degree (namely k^u or k^v). It may be desirable in some applications to have complete trees with small degree available. One way to obtain these is to prune the trees of Lemma 4. With a little more work, we can obtain much taller k-ary trees in $B(k, u, v)$.

We begin with the construction of k-ary out-directed trees in $B(k, u, v)$. For $0 \leq i < u$ and $1 \leq c < k$, let $\mathcal{R}(i, c)$ denote the set of k-ary $u \times v$ matrices with rows $R_1, \ldots, R_u$ where $R_1, \ldots, R_{u-i-1}$ are zero, $R_{u-i}, \ldots, R_{u-1}$ are arbitrary and $R_u = (0^{v-1}, c)$. Choosing any matrix $A \in \mathcal{R}(i, c)$ we write

$$[C_1 \ldots C_v] = \begin{bmatrix} R_{u-i} \\ \vdots \\ R_{u-1} \end{bmatrix}$$

so that $C_1, \ldots, C_v$ are the columns of a submatrix of A.

We describe a construction for a $CTO(k, v-1)$ in $B(k, u, v)$ having root vertex A: A is placed at level 0, while for $1 \leq j \leq v-1$, the vertices in level j of the tree are the matrices of the form

$$\begin{bmatrix} 0 & \ldots & 0 & 0 & 0 & \ldots & 0 \\ \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 0 & \ldots & 0 & 0 & 0 & \ldots & 0 \\ C_{j+1} & \ldots & C_{v-1} & C_v & C_1 & \ldots & C_j \\ 0 & \ldots & 0 & c & w_1 & \ldots & w_j \end{bmatrix} \qquad (0 \leq w_1, \ldots, w_j < k)$$

obtained by repeatedly rotating columns to the left and introducing elements $w_1, \ldots, w_{v-1}$ in position (u, v). It is clear that there are directed edges in E_1 from the vertex in level j with final row $(0^{v-j-1}, c, w_1, \ldots, w_j)$ to the k vertices in level $j+1$ with final rows $(0^{v-j-2}, c, w_1, \ldots, w_j, w_{j+1})$. Supplying these edges we obtain $CTO(k, v-1)$ as a subgraph of $B(k, u, v)$. Notice that if two trees obtained in this way have distinct root vertices, then the trees are vertex disjoint.

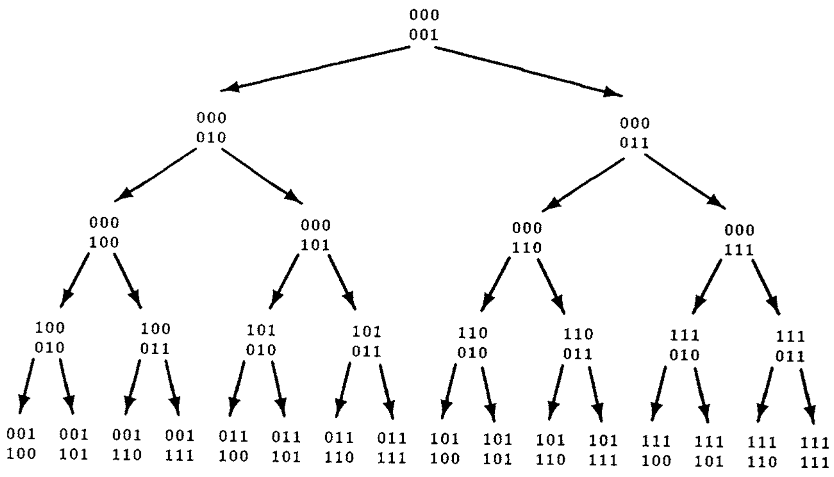


Figure 1. The tree $T(1,1)$ in $B(2,2,3)$

Next we show how to link together subtrees of trees with root vertices in $\mathcal{R}(c_i, i)$, $0 \le i < u$ by edges in E_2 to form a set of vertex disjoint $CTO(k, uv-u)$ in $B(k,u,v)$. To this end, let $(c_0, c_1, \dots, c_{u-1})$ be a k-ary u-tuple with $c_i \ne 0$ for each i. The set $\mathcal{R}(0, c_0)$ contains just one vertex, the array with $u-1$ zero rows and final row $(0^{v-1}, c_0)$. We denote by $T(c_0)$ the $CTO(k, v-1)$ constructed as above having this vertex as root. $T(c_0)$ has leaves of the form

$$\begin{bmatrix} 0 & 0 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & 0 \\ c_0 & w_1 & \dots & w_{v-1} \end{bmatrix} \qquad (0 \le w_0, \dots, w_{j-1} < k).$$

There is an edge in E_2 from such a leaf of $T(c_0)$ to each of the k vertices of the form

$$\begin{bmatrix} 0 & 0 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 0 & 0 \\ c_0 & w_1 & \dots & w_{v-3} & w_{v-2} & w_{v-1} \\ 0 & 0 & \dots & 0 & c_1 & x_1 \end{bmatrix} \qquad (0 \le x_1 < k).$$

These are vertices in level 1 of a tree $T(c_1)$ with root vertex in the set $\mathcal{R}(1, c_1)$. Using the appropriate edge of E_2, we can link the leaf of $T(c_0)$ under consideration to levels 1 up to $v-1$ of the tree $T(c_1)$. The trees $T(c_1)$ that are linked to leaves in this way are vertex disjoint, since their roots are distinct. Hence we have constructed a $CTO(k, 2v-2)$ in $B(k,u,v)$, $T(c_0, c_1)$. It is clear how this process can be continued to generate a $CTO(k, uv - u)$, $T(c_0, c_1, \ldots, c_{u-1})$. As an example, the height 4 tree $T(1,1)$ in $B(2,2,3)$ is shown in Figure 1.

A little more thought shows that distinct choices of u-tuple $(c_0, c_1, \ldots, c_{u-1})$ lead to vertex-disjoint $CTO(k, uv - u)$. Since there are $(k-1)^u$ choices for $(c_0, c_1, \ldots, c_{u-1})$ (this is just one choice in the case $k = 2$) and the construction can be equally well applied to columns instead of rows, we have the following.

Theorem 5. *$B(k,u,v)$ contains a set of $(k-1)^u$ vertex disjoint $CTO(k, uv-u)$ and a set of $(k-1)^v$ vertex disjoint $CTO(k, uv-v)$.*

Thus for example, in the presence of up to $(k-1)^u - 1$ faulty vertices, $B(k,u,v)$ still contains a $CTO(k, uv-u)$.

Using similar ideas, we can construct sets of disjoint $CTI(k, uv-u)$ and $CTI(k, uv-v)$ in $B(k,u,v)$. We sketch the method. For $0 \leq i < u$ and $1 \leq c < k$, we define $\mathcal{R}'(i,c)$ to be the set of of k-ary $u \times v$ matrices with rows $R_1, \ldots, R_u$ where $R_1 = (c, 0^{v-1})$, $R_2, \ldots, R_{i+1}$ are arbitrary and $R_{i+2}, \ldots, R_u$ are zero. For each choice of i, c and $A \in \mathcal{R}'(i,c)$, a $CTI(k, v-1)$ with root vertex A can be obtained (c.f. the construction of $CTI(2, u-1)$ in $B(2,u)$). The set of trees obtained in this way are again vertex-disjoint. For each choice of $(c_0, c_1, \ldots, c_{u-1})$, subtrees of trees with roots in $\mathcal{R}'(i, c_i)$ can be linked using edges in E_2 to the leaves of trees with roots in $\mathcal{R}'(i-1, c_{i-1})$. This gives a $CTI(k, uv-u)$. Analogously to Theorem 5, we have the following.

Theorem 6. *$B(k,u,v)$ contains a set of $(k-1)^u$ vertex disjoint $CTI(k, uv-u)$ and a set of $(k-1)^v$ vertex disjoint $CTI(k, uv-v)$.*

4.3 Meshes

Meshes with the largest possible number of PEs, k^{uv}, can be obtained by considering the combinatorial objects known as Perfect Maps (also known as de Bruijn arrays or de Bruijn tori), for which we give an informal definition below (see [21] for a more detailed development).

Definition 7. *A k-ary $(r,s;u,v)$* Perfect Map (PM) *is a two-dimensional periodic array with periods r and s and symbols drawn from the set $\{0, 1, \ldots, k-1\}$ with the property that every possible $u \times v$ array of symbols occurs exactly once as a sub-array in a period of the array.*

The following necessary conditions on the parameters of a Perfect Map are easily derived.

Lemma 8. *Suppose there exists a k-ary $(r,s;u,v)$ PM. Then*

1. $rs = k^{uv}$,
2. $r > u$ *or* $r = u = 1$,
3. $s > v$ *or* $s = v = 1$.

Our motivation for considering Perfect Maps comes from the fact that a k-ary $(r,s;u,v)$ Perfect Map is equivalent to an embedding of an $r \times s$ directed mesh in $B(k,u,v)$ having the properties that every vertex appears in the mesh, that edges in E_1 are used to move along one dimension of the mesh and that edges in E_2 are used to move along the other dimension. This is analogous to the fact that a k-ary span u de Bruijn sequence is equivalent to a Hamiltonian cycle in $B(k,u)$. We look briefly at the known results on Perfect Maps. A constructive proof of the sufficiency of the conditions of Lemma 8 was given in the case $k = 2$ in [20] and in the case where k is a prime-power in [21]. The latter paper also contains a survey of the other known results on Perfect Maps. The most general result known in the k-ary case was obtained in [22] and can be stated as follows. From Lemma 8, any prime p dividing r or s must also divide k, so we may suppose that k has prime factorisation $k = \prod_{i=1}^{n} p_i{}^{\alpha_i}$ and write $r = \prod_{i=1}^{n} p_i{}^{k_i}$, $s = \prod_{i=1}^{n} p_i{}^{\alpha_i uv - k_i}$ where $0 \leq k_i \leq \alpha_i uv$. The result of [22] is that there exists a k-ary $(r,s;u,v)$ Perfect Map if, for some i, we have

$$p_i{}^{k_i} > u \quad \text{and} \quad p_i{}^{\alpha_i uv - k_i} > v.$$

We illustrate these results by listing the meshes available in two different graphs $B(k,u,v)$.

Example 9. *Consider the graph* $B(2,4,4)$. *From the results of [20], this graph has as subgraphs* 8×8192, 16×4096, 32×2048, 64×1024, 128×512 *and* 256×256 *meshes. Each mesh makes use of all the vertices of the graph.*

Example 10. *By the results of [22], the graph* $B(6,2,2)$ *has as subgraphs* 3×432, 4×324, 6×216, 9×144, 12×108, 18×72, 24×54, 27×48 *and* 36×36 *meshes. Again, each mesh makes use of all the vertices of the graph.*

So far we have examined meshes in $B(k,u,v)$ using all k^{uv} vertices. "Non-maximal" meshes in $B(k,u,v)$ can be derived from other families of arrays having a "window property" (see, for example, [10,18,19]) and allow further variety in the sizes of mesh available.

5 Comparison with other families of graphs

5.1 The directed Moore bound

The directed Moore bound [15] bounds the number of vertices N in a directed graph with maximum out-degree K and diameter D

$$N \leq 1 + K + K^2 + \cdots + K^D.$$

The de Bruijn graph $B(k,u)$ has degree k, diameter u and k^u vertices, so that $\frac{N}{K^D} = 1$ and for k fixed, N has the same asymptotic order as the directed Moore bound as $u \to \infty$. Consider now the performance of the graphs $B(k,u,v)$ relative to the directed Moore bound. For the arguments that follow we assume (without loss of generality) that $u \leq v$. Then $B(k,u,v)$ has k^{uv} vertices, degree $k^u + k^v$ and diameter u. Here

$$K^D = (k^u + k^v)^u = k^{uv}(1 + k^{u-v})^u = N(1 + k^{u-v})^u.$$

When $u = v$, $B(k,u,v)$ has $K^D = N2^u$ so that $\frac{N}{K^D} = 2^{-u}$. This is clearly inferior to the performance of $B(k,u)$. If, on the other hand, $v - u = \Omega(\log(u \log u))$ (for example, if $v = \lfloor cu \rfloor$ for some fixed $c > 1$), then we have

$$\frac{N}{K^D} = (1 + k^{u-v})^{-u} \to 1 \quad \text{as} \quad u \to \infty$$

and $B(k,u,v)$ has the same asymptotic behaviour as the de Bruijn graph $B(k,u)$. Roughly speaking, $B(k,u,v)$ achieves this performance using high degree $(k^u + k^v)$ and low diameter, while $B(k,u)$ has fixed degree and medium diameter. As an illustrative example, consider designing an interconnection network with 2^{16} PEs. $B(2,16)$ has in-degree and out-degree 2 and diameter 16, an order-$(8,8)$ PS network [25] degree 8 and diameter 16, a 16-dimensional hypercube degree and diameter 16, while $B(2,4,4)$ has in-degree and out-degree 32, but diameter only 4. High degree may be the major obstacle limiting the practicality of the graphs $B(k,u,v)$.

5.2 Connectivity and fault-tolerance

We have shown that the connectivity of $B(k,u,v)$ is asymptotically equal to the degree $k^u + k^v$ and have given an efficient fault-tolerant routing algorithm for $B(k,u,v)$. In comparison, even though the graphs introduced in [14,15,24] allow a more flexible choice of parameters and have close to optimal connectivity, it appears that fault-tolerant routing algorithms have yet to be developed for these families.

However, in any practical interconnection network, the number of connections at a PE is limited and it is precisely in this situation where our bound on the connectivity of $B(k,u,v)$ is weak. For example, the graph $B(2,4,4)$ has out-degree 32 but Lemma 2 tells us only that the connectivity is at least -2, whereas the subgraph $B(16,4)$ of $B(2,4,4)$ already has connectivity 15. It would be valuable to obtain a better estimate for the connectivity of $B(k,u,v)$ when u and v are small.

5.3 Computationally useful subgraphs

We have shown that in common with the de Bruijn graph, the graphs $B(k,u,v)$ contain attractive classes of complete trees and cycles as subgraphs, even in the presence of faulty edges and vertices. Additionally, $B(k,u,v)$ offers meshes of maximum size as subgraphs. This feature is also enjoyed by the PS networks of [25], but not by other de Bruijn-type networks.

6 Conclusions

We have examined the properties of the graphs $B(k,u,v)$, a generalisation of the de Bruijn graphs to two dimensions. They have high degree, low diameter and, in some cases, good performance relative to the directed Moore bound. We have further obtained an asymptotically optimal bound on the connectivity of these graphs and given simple routing algorithms. We have shown that they admit a range of computationally useful topologies as subgraphs.

Important topics still to be addressed include finding the exact value of the connectivity of $B(k,u,v)$, examining the emulation capabilities of the graphs $B(k,u,v)$ (can they efficiently emulate other networks such as the hypercube or butterfly networks, for example?) and finding efficient VLSI layouts of the graphs $B(k,u,v)$.

Acknowledgement

The author is grateful to Dr. Simon Blackburn for some stimulating discussions on the content of this paper.

References

1. Bermond, J.-C., Bond, J., Paoli, M. and Peyrat, C. (1983). Graphs and interconnection networks: diameter and vulnerability. *London Math. Soc. Lecture Notes*, **83**, *Surveys in Combinatorics*, Editor: E.K. Lloyd, 1–30.

2. Bermond, J.C., Delorme, C. and Quisquater, J.J. (1992). Table of large (Δ, D)-graphs. *Disc. Appl. Math.*, **37/38**, 575–577.

3. Bermond, J.C. and Peyrat, C. (1989). De Bruijn and Kautz networks: a competitor for the hypercube? *Hypercube and Distributed Comp.*, Editors: F. André and J.P. Verjus, North-Holland, Amsterdam.

4. Blackburn, S. *Personal Communication.*

5. Bondy, J.A. and Murty, U.S.R. (1976). *Graph Theory with App.*, Elsevier.

6. Chamberlain, R.M. (1988). Gray codes, fast fourier transforms and hypercubes. *Parallel Comp.*, **6**, 225–233.

7. Du, D.Z. and Hwang, F.K. (1988). "Generalized de Bruijn Digraphs". *Networks*, **18**, 27–38.

8. Esfahanian, A.-H. and Hakimi, S.L. (1985). Fault-tolerant routing in de Bruijn communication networks. *IEEE Trans. Comp.*, **C-34**, 777–788.

9. Etzion, T. (1986). An algorithm for generating shift-register cycles. *Theo. Comp. Sci.*, **44**, 209–224.

10. Etzion, T. (1988). Constructions for perfect maps and pseudorandom arrays. *IEEE Trans. Info. Theo.*, **34**, 1308–1316.

11. Fan, C.T., Fan, S.M., Ma, S.L. and Siu, M.K. (1985). On de Bruijn arrays. *Ars Combinatoria*, **19A**, 205–213.

12. Fredricksen, H. (1982). A survey of full length shift register cycle algorithms. *SIAM Review*, **24**, 195–221.

13. Homobono, N. and Peyrat, C. (1989). Fault-tolerant routings in Kautz and de Bruijn networks. *Disc. Appl. Math.*, **24**, 179–186.

14. Imase, M. and Itoh, M. (1981). Design to minimize diameter on building-block network. *IEEE Trans. Comp.*, **C-30**, 439–442.

15. Imase, M. and Itoh, M. (1983). A design for directed graphs with minimum diameter. *IEEE Trans. Comp.*, **C-32**, 782–784.

16. Imase, M., Soneoka, T. and Okada, K. (1985). Connectivity of regular directed graphs with small diameters. *IEEE Trans. Comp.*, **C-34**, 267–273.

17. Lempel, A. (1971). m-ary closed sequences. (1971). *J. Combinatorial Theo. Series A*, **10**, 253–258.

18. MacWilliams, F.J. and Sloane, N.J.A. (1976). Pseudo-random sequences and arrays. *Proc. IEEE*, **64**, 1715–1729.

19. Nomura, T., Miyakawa, H., Imai, H. and Fukada, A. (1972). A theory of two-dimensional linear recurring arrays. *IEEE Trans. Info. Theo.*, **18**, 775–785.

20. Paterson, K.G. (1994). Perfect maps. *IEEE Trans. Info. Theo.*, **40**, 743–753.

21. Paterson, K.G. New classes of perfect maps I. *J. Combinatorial Theo. Series A*, to appear.

22. Paterson, K.G. New classes of perfect maps II. *J. Combinatorial Theo. Series A*, to appear.

23. Pradhan, D.K. and Reddy, S.M. (1982). A fault-tolerant communication architecture for distributed systems. *IEEE Trans. Comp.*, **C-31**, 863–870.

24. Reddy, S.M., Pradhan, D.K. and Kuhl, J. (1980). Directed graphs with minimal diameter and maximal connectivity. *Technical Report*, School of Engineering, Oakland University.

25. Rosenberg, A.L. (1992). Product-shuffle networks: toward reconciling shuffles and butterflies. *Disc. Appl. Math.*, **37/38**, 465–488.

26. Rowley, R.A. and Bose, B. (1993). Fault-tolerant ring embedding in de Bruijn networks. *IEEE Trans. Comp.*, **C-42**, 1480–1486.

27. Samatham, M.R. and Pradhan, D.K. (1989). The de Bruijn multiprocessor network: a versatile parallel processing and sorting network for VLSI. *IEEE Trans. Comp.*, **C-38**, 567–581.

28. Shibata, Y., Shirahata, M. and Osawa, S. (1994). Counting closed walks in generalized de Bruijn graphs. *Info. Proc. Let.*, **49**, 135–138.

29. Sridhar, M.A. (1991). On the connectivity of the de Bruijn graph. *Info. Proc. Let.*, **27**, 315–318.

30. Sridhar, M.A. and Raghavendra, C.S. (1991). Fault-tolerant networks based on the de Bruijn graph. *IEEE Trans. Comp.*, **C-40**, 1167–1174.

31. Stone, H.S. (1971). Parallel processing with the perfect shuffle. *IEEE Trans. Comp.*, **C-20**, 153–161.

A Graph Theoretic Solution to the Interface Equation

A.D. Pengelly and D.C. Ince

Department of Computer Science, Open University, Milton Keynes

Abstract

The synthesis and subsequent generation of protocol converters can be a time consuming and tedious affair. An automatic means of generating these converters, given a formal description of the interfacing protocols, has been researched by a number of academics and industrialists. While there is a strong standards movement within the protocol community, the output from that process is likely to be slow in terms of its dissemination to the communications world at large. Hence, the development of convertors will be a key issue for many years to come. The work in this paper can be seen as a natural extension of the earlier work of Norris, Martin and Shields. The theory developed adopts a completely novel approach — moving the problem into the domain of graph theory and topology. The resulting theory has spawned a number of interesting results, which include extensions to CCS, further development of the quotient machine concept, the notion of symmetric validation and a further unification between automata and graph theory.

1 Introduction

Far-reaching technological advances in communication networks have enabled the interconnection of heterogeneous systems in order to provide services such as voice, data and video transfer and the sharing of distributed resources. The importance of protocols to the whole field of communications has led to the establishment of a new sub-discipline of communications engineering, namely *protocol engineering* [6,11].

Even in these days of OSI standardisation there is an increased need for the development of protocol converters which act as an interface between two or more possibly widely differing protocols. There are a number of reasons for this. Green [9], for example, gives a convincing set of reasons why OSI will never become a global standard. First, it is already too late, with very large installed bases of SNA, DECnet and TCP/IP networks. Second, computer communication is a relatively young field and is improving all the time, and new technology is likely to destabilise any attempt to converge to a single architecture. Hence protocol conversion will to be an active area — certainly in the medium term.

Protocol converters are both time consuming and complex to develop [9], especially with modern protocols, which are becoming increasingly complicated. Hence, there are potentially large commercial gains to be made by reducing the development time. By reducing the time to delivery, the converter can be released to the field sooner, where the appropriate services can be delivered to the customer and hence generating revenue much more quickly. So not only are development costs reduced, but revenue increased. The automatic synthesis and development of protocol converter specifications could thus lead to significant commercial benefits.

A number of researchers have been active in this field with [4,9,12,22,28] being the most cited works. The common feature of the work that has been carried out is the reliance on some form of formal description technique, usually simple finite state machines or communicating finite state machines. The general design principles employed are discussed in [3], as are the underlying models in [30], which suggest strategies for solving the converter problem in a systematic manner. There are two basic approaches. The first derives converters at the service level by concatenating common services between the given protocols. In doing so, the protocol engineer can ignore the details of the Protocol Data Unit (PDU) sequences and message exchange control. The problem with this is that since PDU level functionality is not always apparent at the service level, the reliability of the resulting protocol converter may be questionable. This is in fact the most common method (as will be seen below). The second is at the protocol data unit (PDU), where the service specification is ignored. The PDU approach has led to more formal techniques and here the influence of automata theory has been evident.

Of particular interest is the body of work referred to as the *interface equation*. An interface equation has the following form:

$$(p \mid r) \setminus A \sim q.$$

Where p, q and r are processes[1], A the restriction set and $\sim$ strong equivalence[2]. The semantics are that if p and r are composed in parallel, with certain actions restricted as defined by A, then the resultant machine or process will be equivalent to q.

A succession of papers from Norris, Shields and Martin mark the progress of what is the most concerted effort to solve the interface equation. Throughout their research, the base specification language is Milner's CCS [18]. While Norris formulated the problem [20], Shields provided a method of solution for the weakly determinate case using the interesting notions of I–completeness and O–completeness [27]. This was referred to as the discarding algorithm and can be seen as a "brute-force" approach to the problem in that every possible state

[1] We will actually use the term "machine" from now on to mean any automata, process etc.

[2] The reader may ask why strong equivalence is used as opposed to observational equivalence. In short, the current theory cannot recover the information lost via reduction in the observational equivalence case. In essence, too much symmetry is lost.

is examined. In general, the algorithm produces maximal solutions, which can nevertheless be reduced using observational equivalence (essentially identifying τ-cycles and contracting them). However, it quickly becomes computationally intractable for anything but small problems. Martin attempted to improve the efficiency of Shields' work [14]. The approach adopted by Martin for the constructive algorithm was quite different [15,16]. The work involved constructing an initial "guess" and then adding (or removing) more structure as needed. While less general than the discarding algorithm, the advantage of this approach is that the solution is sometimes minimal. However, additional processing is needed and hence the problem of computational hardness is not dealt with. Indeed, to make the constructive algorithm completely general would require additional algorithms, in particular merge-split [17].

The interface equation was developed as a generalised approach to systems synthesis [19,21]. However, protocol conversion turns out to be ideally suited for application to the interface equation. The problem with using the interface equation with such problems is size and the NP characteristics of past algorithms. Modern protocols can be of the order of 3000 states or more. With respect to the work of Shields and Martin, this is simply too high. It would take a enormous amount of time to provide a solution, if one were possible. The suspicion has been that this work is rather a brute force approach and misses certain clues that point to a more economical approach. Likewise, the methods developed by other researchers are also limited in terms of their applicability to the practical problem of protocol development [5,13,23,26].

It was to address many of these limitations that the work described in this paper was undertaken. In short, a new approach to the problem was developed, based on the use of graph theory to solve the interface equation.

2 Graph theory

The first observation that needs to be made is that the solution of the interface equation is essentially a quotient problem, that r is related to q modulo p. Following on from this, we can ask the question "are there existing quotient problems in computer science and discrete mathematics that are applicable?". While the notion of quotient in the context of finite state machines and formal languages (such as context free grammars) is well defined, these are not quite what it is needed in the case of the interface equation. It turns out that the most appropriate domain is graph theory.

Graph theory [2] is a mature, established subject area with a number of important results which are ripe for application. Of particular interest is quotient graph theory, where certain classes of problems can be solved via a range of algorithms in polynomial time. Hence, if a formal mapping between CCS and graph theory could be established, the interface equation could be solved entirely within the graph theoretic domain. The actual relationship is in many ways non-trivial. We have had to extend CCS to accommodate the structural requirements of graph analysis, since the bisimulation semantics of CCS are too limiting. The remainder of this paper highlights the key aspects of the theory developed, though the reader should bear in mind that there are a number of subtle issues, which although they have been addressed, cannot be covered here. These include:

1. What is the relationship between bisimularity and topological equivalence? In particular, how can weak notions of equivalence such as observational equivalence be dealt with from a topological perspective?

2. What behavioural properties of the p and q machines, such as safety and liveness, are inherited by the r machine?

3. Is there a functor from the category CCS to the category GRAPH?

The interested reader should refer to [24].

We first show the rather elementary result that Milner's parallel composition operator is closely related to the Graph Cartesian Product (GCP).

Definition 1. *Given two graphs G_1 and G_2 with $V(G_1) \cap V(G_2) = \emptyset$, the graph Cartesian product $G_1 \times G_2$ is the graph G_3 with vertex set $V(G_1) \times V(G_2)$ such that two vertices (u_1, u_2) and (v_1, v_2) of G_3 are adjacent if and only if either $u_1 = v_1$ and the (directed) edge $u_2v_2 \in E(G_2)$ or $u_2 = v_2$ and $u_1v_1 \in E(G_1)$.*

Now consider machines and in particular the state transition graphs associated with them. It is clear that the state transition graphs are no more than labelled digraphs. As such, we can form the product of the state transition graphs using the GCP. It turns out that the GCP is identical, in terms of the resulting product structure, as that obtained if the machines were specified in CCS and composed using Milner's composition operator "|" and where there are no communicating τ events (that is p and r do not talk to each other). To see this, consider Milner's definition of the composition operator (see the Expansion Theorem [18]), where given $p \rightarrow^{\alpha} p'$ andq $\rightarrow^{\beta} q'$ then $p \mid q$ is defined as:

$$\frac{p \rightarrow^{\alpha} p' \mid q \rightarrow^{\beta} q'}{p \mid q \rightarrow^{\alpha} p' \mid q,\ p' \mid q \rightarrow^{\beta} p' \mid q',\ p \mid q' \rightarrow^{\alpha} p' \mid q',\ p \mid q \rightarrow^{\beta} p \mid q'}.$$

It is clear that there is some similarity here between the structure of the product graphs and the composition of the machines. What is interesting is that we are making topological, or structural, observations. From this point of view the behavioural dynamics of the machines are of no consequence and, as such, issues such as nondeterminism, deadlock, livelock or divergence among others, can effectively be ignored. From these observations we deduce the following theorem:

Theorem 2. *Given two arbitrary machines p and q, such that $\forall\alpha \in \Lambda(p) \neg\exists\overline{\alpha} \in \Lambda(q)$, along with their state transition graph representations $\mathcal{G}_{\mathcal{M}_p}$ and $\mathcal{G}_{\mathcal{M}_q}$ respectively, then $\mathcal{G}_{\mathcal{M}_p}|_{\mathcal{M}_q} \equiv \mathcal{G}_{\mathcal{M}_p} \times \mathcal{G}_{\mathcal{M}_q}$.*

Proof. Not given here (see [24]).

3 Tau splitting

This section will provide the necessary syntax and semantics of τ-splitting, beginning with formal definitions and closing with an example. This splitting process is key to the application of the graph quotient algorithms, since it restores the symmetries of the graph product form (remember the *graphs* are the state transition diagrams of machines). Before dealing with the formal definitions, we will use a simple example to highlight the issues involved.

The aim of the τ-splitting transformation is to resolve a τ_c[3] action into its component actions—that is the actions which combined to form it. If $p_1 \rightarrow^{\alpha} p_2$ and $r_1 \rightarrow^{\overline{\alpha}} r_2$ are composed using parallel composition we get$p_1 \mid r_1 \rightarrow^{\tau_c} p_2 \mid r_2$ (assuming the original α and $\overline{\alpha}$ actions are in the restriction set A). The τ-splitting transformation, which is denoted by Υ, takes as its argument the given transition and "splits" the τ_c in order to recover the α and $\overline{\alpha}$ transitions. Hence, the τ_c is removed from the machine and new transitions added. Now $p_1 \mid r_1 \rightarrow^{\tau_c} p_2 \mid r_2$ could be written as $q_1 \rightarrow^{\tau_c} q_2$. Indeed, in the context of the problem domain covered in this paper, this will be the standard format. Now the inference is that this q-machine is in fact the composition of two other machines. We are given one component, in this case the p-machine, but need to find an r-machine such that $p \mid r$ is equivalent in some way to q. Now if q is the composition of two machines then each state of q, q_1 say, is in fact an ordered pair (p_1, r_1). Following the approach adopted by Shields and Martin, we partition the q machine into distinct sets of states, called $K(r)$-sets. The affect of this is to divide the q machine into a number of planes with each plane being associated with a unique state of the r machine. These planes are connected by actions from the r-machine and τ_c actions *and no others.* All the p-machine transitions not involved in communication lie within the $K(r)$-sets. This is the central observation regarding τ-splitting, since the τ_c transitions between $K(r)$-sets arose from communication between the p and r machine. If the communicating actions are removed and the "lost" actions which combined to form it replaced we would find that the transitions within the $K(r)$-sets are all the transitions performed by the p-machine and the transitions between $K(r)$-sets are the transitions performed by the r-machine. The resulting structure is now in a form whereby the quotient algorithms can be used. There are a number of issues here, they are:

[3] We distinguish between a τ action which arises from communication between the p and r machines.

Definition 3. *The τ-splitting relation Υ is defined by the following mapping:*

$$\Upsilon \;:\; R(q) \times \{\tau_c\} \times R(q) \longrightarrow R(q) \times \Lambda(p) \times R(q) \times R(q) \times \overline{\Lambda}(p) \times R(q)$$

$$\Upsilon \;:\; (q_i, \tau_c, q_j) \longmapsto \{< (q_i, \alpha, q_k)_{\dagger r} \mid (q_i, \overline{\alpha}, q_l)_{\dagger p} >:$$

$$(p, r_i) \rightarrow^{\alpha} (p', r_i) \in \Pi^{\sim}(r_i)\; \forall\, r_i \in R(r) \textit{ and}$$

$$(p_i, r) \rightarrow^{\overline{\alpha}} (p_i, r) \in L(p_i)\; \forall\, p_i \in R(p)\}.$$

Given an interface equation $(p \mid r)\; A \equiv q$. Assume there exists a transition $p' \rightarrow^{\alpha} p''$ in p, where $\alpha \in A$. By definition, all actions in A will be restricted in q. However, there will be τ actions in q which have arisen via p—$-r$ communication. Stated more formally, given $p' \rightarrow^{\alpha} p''$ in p with $\alpha \in A$, then $\neg \exists q', q'' : q' \rightarrow^{\alpha} q''$ in q. But $\exists q_k, q_l, r_m, r_n : q_k \rightarrow^{\tau_c} q_l$ with $q_k \equiv (p' \mid r_m)\; A$ and $q_l \equiv (p'' \mid r_n) \backslash A \Rightarrow (p' \mid r_m) \setminus A \rightarrow^{\tau_c} (p'' \mid r_n) \setminus A$. Since $p' \rightarrow^{\alpha} p''$, it follows that $r_m \rightarrow^{\overline{\alpha}} r_n$. Each r' is associated with a $K(r')$-set, hence $r_m \rightarrow^{\overline{\alpha}} r_n \forall p' \in p.K(r_m) \rightarrow^{\overline{\alpha}} K(r_n)$ (note that it may be the case that $r_m = r_n$). By the definition of the GCP and Theorem 2 $p' \rightarrow^{\alpha} p'' \forall r' \in r$. In terms of Υ, $q_k \rightarrow^{\tau_c} q_l$ is split and projected *onto the underlying p and r components as $< (q', \alpha, q_k)_{\dagger r} \mid (q', \overline{\alpha}, q_l)_{\dagger p} >$, where "$\dagger$" indicates that the action is repeated in the plane indicated.*

Tau splitting is a transformation which effectively removes all internal communication. By doing so the machines involved can be represented using graphs and the equation solved via a quotient graph algorithm. As mentioned earlier, this isn't quite the whole story. Issues such as the nature of equivalence, invariants and the extensions to CCS are not discussed here in detail. Three new concepts have been added to CCS, they are:

1. Structural equivalence, denoted "$\overset{S}{=}$", where two machines are structurally equivalent if their associated underlying directed graphs are isomorphic. This notion of equivalence was necessary since the method of solution described here is essentially structural, whilst existing CCS equivalence notions are behavioural.

2. Structural composition, denoted by "$||$", which is a variation on Milner's composition operator.

3. Rendezvous operator, denoted by "$\wr$", which was introduced to address short-comings (within the field of protocol development) of Milner's restriction operator.

See [24] for further details.

4 Algorithms

This section presents a simplified view of the procedures and algorithms to solve the interface equation using the theory so far described. The authors have deliberately left a number of details in order to convey the basic principals of the algorithm. All details can be found in [24].

The procedure consists on two basic steps. The first is to translate the CCS interface equation to the graph domain. The second is to solve the resulting quotient problem.The first requires extensions to CCS, symmetry-based analysis and τ-splitting. The second uses standard graph quotient algorithms as found in [1,7,8,10,29].

The assumption is that we are given an interface equation in the form,

$$(p \mid r)\backslash A \cup B \sim q$$

where

$$\Lambda(p) \cap \overline{\Lambda}(q) \subseteq \{\tau\},$$

$$\Lambda(p) \cap \overline{A} = \emptyset,$$

$$\Lambda(q) \cap (A \cup \overline{A}) = \emptyset,$$

$$\Lambda(r) \cap \Lambda(p) \subseteq \{\tau\}$$

where A is the set of actions involved in communication, B the set of restricted actions not involved in communication.

Step 1a.

The first step is to identify the states of p and q with r (each state of r corresponds to a "plane" of $p - q$ states which we call Π-planes). This is done using the procedure discussed earlier. We derive the K-sets as per Martin's algorithm [14]. The σ mapping ([24]) is then applied to derive the Π-planes, which are submachines of p. This re-labelled q will be referred to as q'.

Step 1b.

It has been found that the application of symmetric closure ([24]) is best done interactively with Step 1a, since the derivation of the K-sets provides vital clues where reduction has taken place. The best indication that symmetry conditions are being violated is the appearance of a distinct q state in more than one K-set, say $K(r_1)$ and $K(r_3)$. Symmetric closure "unfolds" such states and via the addition of new states and actions, restores the symmetries. In practice symmetric closure will not always be successful.

Step 2.

Construction of the Π-planes is achieved by first grouping the states of q' with respect to the r component. The aim here is to derive the structure of the embedded p machine in q. The algorithms used by Shields and Martin to derive the K-sets can be used here with some modification, such as the exclusion of the O-completeness condition[4]. The output from this process is a set of K-sets which are all individually non-A/B reachable.

Step 3.

The connectedness conditions for the algorithm proposed here are weaker (more general) than those of preceding methods in that strong connectedness is not a precondition. The $\tilde{\Pi}$ (r)-sets are constructed by forming the union of all $K(r)$-sets for each r. We will assume here that the internal structure of each $\tilde{\Pi}$-plane is weakly connected.

Notice that at this stage we can begin to use symmetry as a validation tool. The emergent structure should contain certain symmetries, which if not there, are a good indicator to the existence of errors. These errors are typically missing states and actions or incorrect labelling.

Step 4.

The L sets are formed in exactly the same fashion as the Π-planes. However, unlike the Π-planes, it is assumed that each L-set will be non-A/B reachable ([24]).

Step 5.

Let $\mu \in \Lambda(p) - (A \cup \overline{A} \cup B)$, an action of p which does not communicate. Let C be the set of all such actions. These actions provide no additional information in terms of the derivation of the r machine. Construct the machine $q'' = q' \backslash C$. q'' will now contain non-τ actions which are from r and τ actions which are either intrinsic to p or r, or are communicating τ actions. The primary function for removing the C actions is to reduce the processing requirements for quotient extraction. The next step is to distinguish between the τ_i and τ_c actions.

Step 6.

The main steps in distinguishing between τ_i and τ_c actions is as follows:

Pred $= \exists \mu \in \Lambda(p) \cap A : p \rightarrow^{\mu} p'$ and
$\mu \notin \Lambda(q)$ and $\neg\, \exists \tau : L(p'') \rightarrow^{\tau} L(p'''), \forall r \in R(r)$ for some
r or $\neg\, \exists \tau : \tilde{\Pi}\ (r) \rightarrow \tilde{\Pi}\ (r'')\ \forall p \in R(p)$.

[4]It is felt that graph partitioning has a key role to play here, but is left as a subject for future research.

```
For each $(p,q,r) \rightarrow^{\tau} (p',q',r')$ transition do
    If Pred = true then
        Split $\tau$ (Step 7)
    else
        Mark intrinsic
    end if
Repeat until $\widetilde{\Pi} \overset{S}{=} p \, (\Delta(p, \widetilde{\Pi}) = \emptyset)$.
```

The rather complex expression within the **if** statement first checks to see if p did an action in A which q'' does not do. This action will be involved in communication and hence will be associated with a τ_c action. However, even if such an action can be found there are situations where this may not be the correct τ action. Hence, the rest of the expression checks the symmetry properties to see if there are τ transitions in the $R(p) \times R(q)$ or $R(r) \times R(q)$ planes which are symmetric.

Step 7.

From Step 6 identify all $(p,q,r) \rightarrow^{\tau_c} (p',q',r')$ transitions. Identify $\mu \in \Lambda(p) \cap A : p \rightarrow^{\mu} p'$ in p. Add transition to q'' $\forall r \in R(r)$. Create $r \rightarrow^{\overline{\mu}} r'$ transition and add to q'' $\forall p \in R(p)$. Remove $(p,q,r) \rightarrow^{\tau_c} (p',q',r')$ from q'' as described in Definition 3. A q with all its τ_c actions split is labelled $q^{\bullet}$.

Step 8.

We now apply the quotient algorithm.

1. Factor G_q (a weakly connected digraph).
2. $G_q := \mathrm{U}(D(V_q, E_q))$ the underlying graph of $D(V_q, E_q)$.
3. $G_p := \mathrm{U}(D(V_p, E_p))$ the underlying graph of $D(V_p, E_p)$.
4. Find the prime factorisation of G_q (say m-components).
5. Let $A_1, A_2, \ldots, A_m$ be the arc classes corresponding to the edge classes $E_1, E_2, \ldots, E_m$ (associated with G_q).
6. Set $R = \emptyset$.
7. Apply algorithm [7].

```
for each pair of arc classes $A_i, A_j$
    for each pair of adjacent copies $G_i^x, G_i^y$
        for each pair of edges $u - v \in E_i, u - u' \in E_j, u \in G_i^x, u' \in G_i^y$
        if the 4-cycle $u - u' - v' - v - u$ lifts to a subgraph of $D(V_q, E_q)$ that shows
        a conflict between $A_i$ and $A_j$
        then $R := R \cup \{(A_i, A_j)\}$;
```

```
$R^* :=$ the reflexive, transitive closure of $R$;
$p :=$ the number of equivalence classes $R^*$;
for i := 1 to p
begin
    Let $\{A_i\}$ be the $i$th class in $R^*$;
    Let $\{E_i\}$ be the corresponding edge classes in $\mathcal{G}_{\mathcal{M}_q}$;
    $H := G_1^x \cup \cdots \cup G_i^x$ for some $x \in V(G)$;
    Let $D_i$ be the subgraph of $D(V_q, E_q)$ to which $H$ lifts;
    The $i$th prime factor of $D(V_q, E_q)$ is isomorphic to $D_i$;
end
```

8. $D(V_q, E_q) = D_1 \times D_2 \times \cdots \times D_m$.
9. Repeat for $G_p = D(V_p, E_p)$, giving $D(V_p, E_p) = D_1 \times D_2 \times \cdots \times D_n$, where $m \geq n$.

The assertion is that $G_r = D(V_r, E_r)$ is equal to:

$$\begin{aligned} D(V_r, E_r) &\cong \frac{D(V_q,E_q)}{D(V_p,E_p)} \\ &\cong \frac{D_1 \times D_2 \times \cdots \times D_m}{D_1 \times D_2 \times \cdots \times D_n} \\ &\cong D_{m-n} \times D_{m-n+1} \times \cdots \times D_m. \end{aligned}$$

Hence the unlabelled $G_r \cong D_{m-n} \times D_{m-n+1} \times \cdots \times D_m$. This is the unlabelled underlying digraph of r.

Step 9.
The output from Step 8 is the underlying digraph of r which has unlabelled edges, but resides within q''. Labelling of the edges, which corresponds to actions in the machine representation, can be achieved by pattern matching on q'', or via heuristic methods.

Step 10.
Use a tool such as Concurrency Workbench to check that r is a solution of $(p \mid r)\backslash A \cup B \sim q$.

It is interesting to note that under certain conditions the quotient graph algorithm is not needed and we can extract the quotient via indirect means using the L-sets. Clearly enormous processing gains could be made here, but this is a matter for further research.

The algorithm has been validated on a number of examples, including a 600 state q and 45 state p-machine pair.

5 Summary

This paper has presented the basic elements required to solve the interface equation using graph theory. Central to the approach is the use of symmetry and the splitting of the communicating events, followed by quotient extraction. A procedure for generating a solution has been given. Application to a much larger example, which is significantly closer to industrial sized problems than previously encountered in the work by other researchers, is given in [24].

Comparison with other techniques has shown that our approach is able to solve problems which could not be realistically solved via previous techniques and that this class of problems is extensive. On the other hand, our approach needs further work before we can satisfactorily solve the observational equivalence problem (where too much information is lost during the reduction process). An interesting assertion is that the graph theoretic approach, with additional algorithms to solve the observational equivalence problem, is equivalent to the constructive algorithm with merge-split [17]. The strength of the graph theoretic approach is that issues such as deadlock, livelock, divergence, fairness, nondeterminism and weak-connectedness present no difficulties for the theory [25]. The inherent reliance on symmetry proved to be a valuable validation tool, since non-compliance to certain symmetry constraints invariably indicated an error in the protocol specifications.

The authors also assert that the graph theoretic approach is computationally more tractable than previous algorithms and will reduce protocol converter synthesis development times by at least an order of magnitude. This work offers the chance of radically reducing interface development times and effort associated with protocol development.

6 Acknowledgements

The authors are grateful for the contributions of Dr. G.A.R. Martin and Dr. M. Norris at BT Laboratories in the preparation of this paper. This work has been part-funded by BT.

References

1. Aurenhammer, F., Hagauer, J. and Imrich, W. (1992). Cartesian graph factorisation at logarithmic cost per edge. *Computational Complexity*, **2**, 331–349.

2. Behzad, M. and Chartrand, G. (1971). *Introduction to the Theory of Graphs.* Allyn and Bacon.

3. Bochman, G.V. and Mondain-Monval, P. (1990). Design principles for communication gateways. *IEEE Trans. on Selected Areas in Comms.*, **8**.

4. Calvert, K. and Lam, S. (1990). Formal methods for protocol conversion. *IEEE Selected Areas in Comms.*, **8**.

5. Kristol, D. et al. (1993). A polynomial algorithm for gateway generation from formal specifications. *IEEE Trans. on Networking*, **1**, 217–229.

6. Lin, F. and Liu, M. (1992). The rise of protocol engineering. *IEEE Software.*

7. Feigenbaum, J. (1986). Directed Cartesian-product graphs have unique factorisations that can be computed in polynomial time. *Discrete Applied Maths.*, **15**, 105–110.

8. Feigenbaum, J., Hershberger, J. and Schaffer, A. (1985). A polynomial time algorithm for finding the prime factors of cartesian-product graphs. *Discrete Applied Maths.*, 123–138.

9. Green, P. (1986). Protocol conversion. *IEEE Trans. on Comm.*, **34**.

10. Imrich, W. and Zerovnik, J. (1994). Factoring Cartesian product graphs. *J. Graph Theory*, **18**.

11. DeLapeyre, J.F. (1993). Introduction to protocol engineering using specification and description language. *BT Technology J.*, **11**, 7–8.

12. Lam, S. (1986). Protocol conversion: Correctness problems. *Sym. Proc. ACM SIGCOMM'86.*

13. Lam, S. (1988). Protocol conversion. *IEEE Trans. Software Eng.*, **14**.

14. Martin, G. (1987). A general solution to the interface equation. *Technical Report, British Telecom Research and Technology Internal Memorandum Number: R11/87/023.*

15. Martin, G. (1987). Practical considerations of applying the interface equation. *Technical Report, British Telecom Research and Technology Internal Memorandum Number: R11/87/007.*

16. Martin, G. (1989). A study into a constructive method for generating solutions to the interface equation. *Technical Report, British Telecom Research and Technology Internal Memorandum Number: RT31/89/004.*

17. Martin, G. and Pengelly, A. (1993). A note describing the relevance of the merge/split routine to the constructive algorithm for solving the interface equation and ideas for its implementation. *Technical Report, BT Laboratories, Ipswich.*

18. Milner, R. (1980). *A Calculus of Communicating Systems, Lecture Notes in Computer Science*, **92**, Springer-Verlag.

19. Norris, M.T., Everett, R., Martin, G. and Shield, M. (1988). A method for the synthesis of interactive system specifications. *Conf. Proc. Global Telecoms.*

20. Norris, M.T. (1985). The role of formal methods in system design. *BT Tech. J.*, **3**.

21. Norris, M.T., Shields, M.W. and Ganeri, J. (1987). A theoretical basis for the construction of interactive systems. *BT Tech. J.*, **5**.

22. Okumara, K. (1990). Generation of proper adapters and converters from a formal service specification. *INFOCOM'90*, IEEE Computer Society Press.

23. Parrow, J. (1987). Submodule construction as equation solving in CSS. *Technical Report, Laboratory for the Foundations of Computer Science*, University of Edinburgh.

24. Pengelly, A. (1995). The application of graph theory to the synthesis of protocol converters via the interface equation. *PhD Thesis*, Open University.

25. Pengelly, A. and Ince, D. (1995). The solution of the interface equation. *Int. Conf. Appl. Maths.*, ICAM.

26. Shields, M.W. (1989). Implicit system specification and the interface equation. *The Computer J.*, **32**.

27. Shields, M.W. (1986). Solving the interface equation, Technical Report SE/079/2. *Technical Report, Electronic Engineering Laboratories*, University of Kent at Canterbury.

28. Shu, J. and Liu, M. (1990). Protocol conversion between complex protocols. *9th Ann. Int. Conf. Computers and Communcations*, IEEE.

29. Winkler, P. (1987). Factoring a graph in polynomial time. *Euro. J. Combinatorics*, **8**, 209–212.

30. Yao, Y.W., Chen, W.S. and Liu, M.T. (1990). A parallel model for the construction of protocol converters. *Proc. GLOBECOM'90*, **3**.

Uniformly Optimally Reliable Networks for Vertex Failures

D.H. Smith

Department of Mathematics and Computing, University of Glamorgan, Wales

Abstract

Graphs with minimum probability of disconnection for all vertex failure probabilities are said to be *Uniformly Optimally Reliable.* We describe some recent results concerning the existence of these graphs. In particular, we prove a new nonexistence result for graphs of large diameter. We also illustrate the proof method used for sparse graphs by proving nonexistence in one case that has not been explicitly dealt with in the literature.

These results lead us to consider the existence of *Regular Uniformly Optimally Reliable Graphs.* For small vertex failure probabilities we extend our previous results on the number of minimal vertex cut sets. This result, and its associated constructions, are also of practical interest. For large vertex failure probabilities we show that the girth of the graph must be maximal. The combination of these two cases suggests that regular uniformly optimally reliable graphs are rare.

The implications of these results for network designers are discussed.

1 Introduction

The study of network reliability is concerned with the interconnection of various elements in the form of a network, typically a telecommunication, distribution or computer network. The components of the network are represented by the edges and vertices of an underlying directed or undirected graph. The components may be unreliable, but the network should operate as reliably as possible in the presence of component failures.

Work on network reliability has for the most part followed two lines of approach. The first approach is concerned with *methods of calculating* the reliability of a fixed network for a given reliability measure. A second alternative approach attempts to *design* the graph underlying the network, given a fixed number of vertices and edges, so that the network will operate as reliably as possible with respect to a given measure of reliability. We shall concern ourselves with the latter approach.

Many different measures of reliability have been proposed. One of the most natural is the *probability of disconnection* of the network, and we shall concentrate on this reliability measure. Most work on this measure has been concerned with the edge failure case. However the results of Smith [10] show that for small failure probabilities optimal graphs in the vertex failure case are often optimal graphs for mixed and edge failures. Consequently in this paper we study the vertex failure case, using an undirected graph as a model of the network. It can also be argued that the vertex failure case is often of more practical importance. The edges may represent relatively reliable cables or radio links, whereas the vertices may represent complex failure prone systems.

Let G be a finite, simple, undirected, connected graph with N vertices and m edges, which we shall refer to as an (N, m) graph. Let G have connectivity κ and let $k = \lfloor 2m/N \rfloor$. If the vertices of G all have the same probability p of failure (removal with their incident edges) and the failures are assumed independent, then we can write

$$P(G) = \sum_{i=\kappa}^{N-2} N_i \, p^i (1-p)^{N-i},$$

where $P(G)$ is *the probability of disconnection of G*, and N_i is the number of vertex cut sets with i vertices.

If p is sufficiently small then $P(G)$ is minimised if κ is maximised and N_κ is minimised. The maximum value of κ is k and Harary [6] showed that graphs of connectivity k always exist. A number of results are available which construct graphs with κ maximal and N_κ minimal [3,8,9,11] although these results are not comprehensive.

A graph is said to be *uniformly optimally reliable* for vertex failures if $P(G)$ is minimised for all $p, 0 < p < 1$. In the corresponding case for edge failures Boesch et al. [4] showed that uniformly optimally reliable graphs exist for $N - 1 \leq m \leq N - 2$. Boesch [2] conjectured that uniformly optimally reliable graphs always exist, but Myrvold et al. [7] showed that if $N \geq 6$ is even and $m = N(N-2)/2 - 1$ then there does not exist a uniformly optimally reliable graph for edge failures. In the case of vertex failures Amin et al. [1] have studied a measure called pair–connected reliability and shown that for this measure uniformly optimally reliable graphs do not exist for $N \leq m \leq \sim 2N^2/9$, although they do exist for some other values of m. The existence of uniformly optimally reliable graphs for the probability of disconnection in the vertex failure case has been studied in a very recent paper [5].

In this paper we shall restrict attention to the probability of disconnection $P(G)$ in the vertex failure case. We shall see that uniformly optimally reliable graphs do not exist if G has diameter > 4 and, in fact, do not exist in most cases when the graph is sparse.

This leads us to study the same problem for regular graphs: does there exist a regular $(N, kN/2)$ graph with $P(G)$ minimised for all $p, 0 < p < 1$ in the vertex failure case? We improve the result in [11] which is used for constructing optimal graphs in the case of small p. In this case an optimal graph will usually have girth 4. On the other hand for p close to 1 an optimal graph has maximal girth. Thus regular uniformly optimally reliable graphs for vertex failures are also rare.

In the concluding section we discuss the implications of these results for network designers.

2 Degree sequence dominance

Let P_3 denote a path with three vertices and, following [1], we let $\#(G, P_3)$ and $*(G, P_3)$ denote respectively the number of subgraphs of G isomorphic to P_3 and the number of induced subgraphs of G isomorphic to P_3.

Denote by C_i the number of connected induced subgraphs of G with i vertices. Then $N_i = \binom{N}{i} - C_{N-i}$ and $C_2 = m$. Thus N_N, N_{N-1}, N_{N-2} are fixed for an (N, m) graph and for sufficiently large p, $P(G)$ is minimised if C_3 is maximised. If $\#(G, K_3)$ denotes the number of triangles in G then

$$\begin{aligned} C_3 &= *(G, P_3) + \#(G, K_3) && (2.1) \\ &= \#(G, P_3) - 2\#(G, K_3) && (2.2) \end{aligned}$$

as each triangle contributes $3P_3$s which are not induced subgraphs.

Definition 1. *Let $D(G) = (d_1, d_2, ..., d_N), (d_1 \geq d_2 \geq ... \geq d_N)$ and $D(H) = (d'_1, d'_2, ..., d'_N), (d'_1 \geq d'_2 \geq ... \geq d'_N)$ denote the degree sequences of (N, m) graphs G and H respectively. Then $D(H)$ is said to* dominate *$D(G)$ if for all $j = 1, 2, ..., N$*

$$\sum_{i=1}^{j} d'_i \geq \sum_{i=1}^{j} d_i$$

with strict inequality for at least one j.

It follows from 2.2 that

$$C_3 = \sum_{i=1}^{N} \binom{d_i}{2} - 2\#(G, K_3) \qquad (2.3)$$

The following Lemma is straightforward [1].

Lemma 2. *If $D(H)$ dominates $D(G)$ then $\#(H, P_3) > \#(G, P_3)$.*

Lemma 3. *Let G be an (N, m) graph and let G_c be the complement of G. Then G minimises $P(G)$ for p large if and only if $\#(G, K_3) + *(G_c, P_3)$ is the minimum over all (N, m) graphs.*

Proof. Let e be an edge of G incident with vertices u and v. Denote by $n_0(e), n_1(e)$ and $n_2(e)$ the number of vertices adjacent to exactly 0, 1, and 2 vertices in $\{u, v\}$ respectively. Then $n_0(e)$ is the number of induced P_3s in G_c with u and v as end vertices, $n_1(e)$ is the number of induced P_3s in G containing the edge e and $n_2(e)$ is the number of K_3s in G containing the edge e. Also

$n_0(e) + n_1(e) + n_2(e) = N - 2$. Summing over all edges and noting that in G each induced P_3 is counted twice and each K_3 is counted 3 times, we have

$$\begin{aligned} m(N-2) &= \sum_{e \in G} (n_0(e) + n_1(e) + n_2(e)) \\ &= *(G_c, P_3) + 2 * (G, P_3) + 3\#(G, K_3). \end{aligned}$$

Then

$$\begin{aligned} C_3 &= *(G, P_3) + \#(G, K_3) \\ &= (m(N-2) - *(G_c, P_3) - \#(G, K_3))/2 \end{aligned}$$

and the result follows.

3 Graphs with large diameter

As noted in the introduction, an (N, m) graph G which minimises $P(G)$ for p small will have connectivity $\kappa = k = \lfloor 2m/N \rfloor$ which will equal the minimum degree of G.

Theorem 4. *If G has minimum degree $k = \lfloor 2m/N \rfloor$ and no vertex y such that $d(x, y) \le 2$ for all vertices x, then G does not minimise $P(G)$ for large p.*

Proof. Choose a vertex x_1 adjacent to a vertex x_2 of degree k. Choose another vertex x_3 such that $d(x_3, x_1) > 2$. Construct a new graph G' by removing the edge $(x_1 x_2)$ and inserting an edge $(x_1 x_3)$. Since no new triangle is created and the degree sequence of G' dominates the degree sequence of G, it follows from Equation 2.2 and Lemma 2 that C_3 is not maximised in G.

In particular, if G has *diameter* > 4 or if $m = Nk/2$ (so G is regular) and $N > 1 + k^2$ then G is not uniformly optimally reliable.

4 Graphs with k/N small

We can extend Theorem 4 by following closely the method of [1] for pair connected reliability, which makes crucial use of Lemma 3. However, these results appear to have been superceded by the results in [5] which include the following:

Theorem 5. *Given N and m, let $2m = Nk + s$ $(0 \le s < N/2, 4 \le k \le 2N/5 - 8)$. Then there does not exist a uniformly optimally reliable (N, m) graph.*

The condition $(0 \le s < N/2)$ is easily extended to $(0 \le s < N)$ and, in fact, Goldschmidt et al. [5] claim, without giving full details, that Theorem 5 can be extended as follows:

Theorem 6. *Given N and m, let $2m = Nk + s$ $(0 \leq s < N, 2 \leq k \leq N/2 - 6, m > N)$. Then there does not exist a uniformly optimally reliable (N,m) graph.*

The methods used to prove these theorems are very similar to those in [1] and are basically as follows. Find an upper bound for C_3 for an (N,m) graph of connectivity k. Then find a graph with a larger value of C_3, normally a complete bipartite graph plus or minus some edges. To illustrate the method we follow closely the method in [1] to give a simple proof in the case $k = 2, N \geq 8$.

Proof in the case k=2,N ≥8. Let G denote an optimal (N,m) graph which minimises $P(G)$ for p small. G will have connectivity 2 and $m = N + \theta$ $(0 \leq \theta < N/2)$.

As $C_3 = \#(G,P_3) - 2\#(G,K_3)$ we see that C_3 cannot exceed $2\binom{\theta+2}{2} + (N-2)\binom{2}{2}$ corresponding to a graph with no triangles and degree sequence $(\theta+2, \theta+2, 2, 2, ..., 2)$ which dominates the degree sequence of all other 2-connected $(N, N+\theta)$ graphs.

Now let G_1 denote the graph obtained from $K_{2,N-2}$ by removing $N-\theta-4$ edges from a vertex in the part with two vertices. Then

$$C_3(G_1) = \binom{N-2}{2} + \binom{\theta+2}{2} + \theta + 2$$

and

$$\begin{aligned} C_3(G_1) - C_3(G) &\geq (N-2)(N-3)/2 - (\theta+2)(\theta+1)/2 + \\ &\quad (\theta+2) - (N-2) \\ &\geq (N-2)(N-5)/2 - (\theta+2)(\theta-1)/2 \\ &> 0 \text{ for } N \geq 8 \text{ as } N > \theta + 4. \end{aligned}$$

Thus G is not uniformly optimally reliable.

5 Regular uniformly optimally reliable graphs

Uniformly optimally reliable graphs for the probability of disconnection in the presence of vertex failures certainly exist, for example $K_{3,3}$ is easily seen to have the minimum value of N_i for $i = 1, 2, ..., 4$. In fact complete multipartite graphs can be shown to be uniformly optimally reliable (see for example [5]). However, the results of the previous sections show that other examples are likely to be rare. In addition, the case when the probability p of vertex failure is close to 1 is rather anomalous. Minimising N_{N-3} may force the graph to be nonregular and the connectivity to be smaller than it need be. This conflicts with the usual requirement to maximise the connectivity of a network. If the value of p is not known in advance, it is possible that a very sub-optimal graph will be obtained if too much weight is given to values of p close to 1. It is therefore of interest to remove this oddity by focusing attention on regular graphs.

Definition 7. *A regular graph with N vertices and degree k is said to be* regular uniformly optimally reliable *if $P(G)$ is minimised over all such graphs for all $p, 0 < p < 1$.*

Before considering the existence of regular uniformly optimally reliable graphs we shall improve somewhat the results in [11] for the case of p close to 0.

6 Small probabilities of vertex failure

Smith and Doty [11] noted that, for sufficiently small p, $P(G)$ is a minimum for regular graphs of degree k if G has connectivity k and N_k is minimised. They proved that if $N/k > 5/2$ then $N_k \geq \lceil 5N/2k \rceil$ and constructed families of graphs meeting this bound. In this section we will remove the need for the condition $N/k > 5/2$ and characterise graphs meeting the bound in certain cases.

Let $\Gamma(u)$ denote the set of vertices adjacent to u in G. Define an equivalence relation $\sim$ on the vertices of G by $u \sim v$ if and only if $\Gamma(u) = \Gamma(v)$. Let $\Gamma(G)$ be a graph with the equivalence classes of $\sim$ as vertices. Vertices $[u]$ and $[v]$ of $\Gamma(G)$ are adjacent if and only if u and v are adjacent in G. We refer to $\Gamma(G)$ as the quotient graph of G. Label the vertices $[v]$ of $\Gamma(G)$ with integer weights $L[v]$, the weight giving the number of vertices in the equivalence class.

Theorem 8. *If G is a noncomplete regular graph with $N > 3$ vertices, degree k, connectivity k and with q vertex cut sets with k vertices, then either G is a complete bipartite graph K_{kk} with $q = 2$, $\Gamma(G)$ is a cycle C_3 and $q = 3$, $\Gamma(G)$ is a cycle C_5 and $q = 5$, or $q \geq \lceil 5N/2k \rceil$.*

In order to prove this theorem we note first that the quotient graph of G can only have vertices of degree 1 if G is complete bipartite. Assume that G satisfies the conditions of the theorem. If $\Gamma(G)$ is not a cycle then $\Gamma(G)$ must have at least 2 vertices of degree > 2.

Lemma 9. *If $\Gamma(G)$ is a cycle then $\Gamma(G) = C_3$, $\Gamma(G) = C_5$ or $q \geq \lceil 5N/2k \rceil$.*

Proof. $\Gamma(G)$ cannot be C_4 by the definition of quotient graph. If $\Gamma(G)$ is a cycle $C_i (i > 5)$, then $q > \lceil 5N/2k \rceil$.

The next lemma is essentially Lemma 1 in [11]. The condition $N/k > 5/2$ stated in [11] is not used in the proof.

Lemma 10. *Suppose that $\Gamma(G)$ is not a cycle. Let $\Gamma(G)$ have at least two vertices of degree > 2, and at least two vertices of degree 2 which are adjacent. Let $[a], [c_1], [c_2], ..., [c_r], [b] (r \geq 2)$ be a path in the quotient graph such that $[a], [b]$ have degree > 2 and the vertices $[c_i] (i = 1, 2, ..., r)$ have degree 2. Then*

$$L([c_1]) = L([c_2]) = \cdots = L([c_r]) = s,$$

with $s \leq k/2$ and $[a], [b]$ are not adjacent. (In fact, if $r \geq 3$, k is even and $s = k/2$).

Lemma 11. *Suppose that the quotient graph $\Gamma(G)$ has vertices of degree > 2. Let S denote the set of vertices of G corresponding to vertices of degree 2 in $\Gamma(G)$ and S' denote $V(G) \setminus S$. Suppose that $\Gamma(G)$ has $e \geq 0$ edges that join two vertices of degree 2. Then $|S'| \geq N/2 - ke/4$.*

Proof. It follows from Lemma 10 that G has at most $ek^2/4$ edges joining vertices of S and so

$$|S|.k - e.k^2/2 \leq \text{ number of edges joining S to S}' \leq k|S'|$$

so

$$|S'| - |S| \geq -ek/2$$

and the result follows from $|S'| + |S| = N$.

The following Lemma is essentially Lemma 3 in [11]. The condition $N/k > 5/2$ stated in [11] is not used in the proof.

Lemma 12. *Suppose that the quotient graph $\Gamma(G)$ has vertices of degree > 2 and that there are e edges of $\Gamma(G)$ that join two vertices of degree 2. Then G has at least $2e$ vertex cut sets that are not neighbour sets of a vertex.*

Proof of Theorem 8. If $\Gamma(G)$ is a cycle the result follows from Lemma 9. Otherwise, let q' denote the number of vertex cut sets of G with k vertices that are vertex neighbour sets. Let e, S, S' be as defined in Lemma 11. Counting in two ways the vertices in the vertex cut sets of G that are neighbour sets, we have

$$q'.k = \sum_{[v_i] \in \Gamma(G)} d([v_i]).L([v_i])$$

Let T denote the set of vertices of $\Gamma(G)$ corresponding to vertices of S and $T' = V(\Gamma(G)) \setminus T$.

$$\begin{aligned} q'.k &= \sum_{[v_i] \in T} 2.L([v_i]) + \sum_{[v_i] \in T'} d([v_i]).L([v_i]) \\ &\geq 2 \sum_{[v_i] \in \Gamma(G)} L([v_i]) + \sum_{[v_i] \in T'} L([v_i]) \\ &\geq 2N + |S'| \\ &\geq 5N/2 - ke/4 \text{ (from Lemma 11).} \end{aligned}$$

From Lemma 12 we have

$$\begin{aligned} q &\geq q' + 2e \\ &\geq 5N/2k + 7e/4 \end{aligned}$$

so $q \geq \lceil 5N/2k \rceil$.

Notice that if $q = \lceil 5N/2k \rceil$ we must have $e = 0$. If in addition $5N/2k$ is an integer we have $|S'| = |S| = N/2$ and $d([v_i]) = 3$ for $[v_i] \in T'$. We see from the proof of Lemma 11 that each vertex of degree 3 in $\Gamma(G)$ is adjacent to three vertices of $\Gamma(G)$ of degree 2. Thus $\Gamma(G)$ is obtained from a graph of degree 3 by inserting a vertex of degree 2 on every edge. Examples of optimal graphs with $\Gamma(G)$ of this type are given in [11]. If $k > 3$ then such an optimal graph, if it exists, must have girth 4. These optimal graphs certainly exist if $N/k \geq 4$ is an even integer and $k \geq 8$ [11]. Thus in this case at least, optimal regular graphs for small probabilities of vertex failure have girth 4. The quotient graph approach gives evidence for the conjecture that optimal regular graphs for small probabilities of vertex failure will have girth 4 in most other cases where the number of edges is not too large.

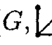
7 Large probabilities of vertex failure

As noted for the general case in Section 2, if C_i denotes the number of connected induced subgraphs of G with i vertices then $N_i = \binom{N}{i} - C_{N-i}$, $C_1 = N$ and $C_2 = m$. Thus for sufficiently large p, $P(G)$ is minimised if C_3 is maximised. In general, if regular graphs G, G' have $C_i = C_i'$ $(i = 3, 4, ..., j-1)$ and $C_j < C_j'$, then $P(G')$ is smaller than $P(G)$ for sufficiently large p.

In the regular case we have from Equation 2.2

$$C_3 = Nk(k-1)/2 - 2\#(G, K_3).$$

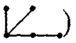
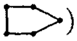
If G has girth > 3 we have

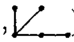
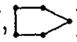
$C_4 = *(G, P_4) + \#(G,$  $) + \#(G, \square)$

$= \#(G, P_4) + \#(G,$ $) - 3\#(G, \square)$

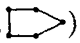
$= Nk(k-1)^2/2 + Nk(k-1)(k-2)/6 - 3\#(G, \square).$

Similarly, if G has girth > 4

$C_5 = *(G, P_5) + \#(G,$  $) + \#(G,$ $) + \#(G,$  $)$

$= \#(G, P_5) + \#(G,$  $) + \#(G,$ $) - 4\#(G,$  $)$

$= Nk(k-1)^3/2 + N(k(k-1)(k-2)/6).3(k-1) +$

$Nk(k-1)(k-2)(k-3)/24 - 4\#(G,$  $)$

and in general if G has girth $> s - 1$

$$\begin{aligned} C_s &= *(G, P_s) + \#(G, \text{trees with } s \text{ vertices} \neq P_s) + \#(G, \text{s} - \text{cycles}) \\ &= \#(G, \text{trees with } s \text{ vertices}) - (s-1)\#(G, \text{s} - \text{cycles}). \qquad (7.1) \end{aligned}$$

The girth condition implies that the number of each type of tree in G counted in Equation 7.1 depends only on N and k. To see this, fix either the centre of the tree in N ways or the bicentre in $Nk/2$ ways. In either case the number of ways of completing the tree, given the girth condition, depends only on k.

It follows from Equation 7.1 that if G, G' are regular of degree k and girth $G >$ girth G' then G will be more reliable than G' for sufficiently large p.

8 Conclusion

We have seen that in the regular case as well as in the general case, the conditions for small p and for large p tend to conflict. In the regular case many of the optimal graphs for small p can be constructed using quotient graphs and have girth 4. In the case of large p we must attempt to maximise girth. Again in the regular case, the example $K_{3,3}$ shows that regular uniformly optimally reliable graphs do exist, but we conjecture that they are rare.

It appears that a network designer who wishes to design an optimal or near optimal network, particularly when vertex failures are important, should have a good estimate of p available in advance. Otherwise, if the estimate of p is wrong, there may be a danger of choosing an apparently optimal network whose reliability is in fact very poor.

References

1. Amin, A.T., Siegrist, K.T. and Slater, P.J. (1993). On uniformly optimally reliable graphs for pair connected reliability with vertex failures. *Networks*, **23**, 185–193.

2. Boesch, F.T. (1986). On unreliability polynomials and graph connectivity in network synthesis. *J. Graph Theo.*, **10**, 339–352.

3. Boesch, F.T. and Felzer, A. (1971). On the invulnerability of the regular complete k-partite graphs. *SIAM J. App. Math.*, **20**, 176–182.

4. Boesch, F.T., Li, X. and Suffel, C. (1991). On the existence of uniformly optimally reliable networks. *Networks*, **21**, 181–194.

5. Goldschmidt, O., Jaillet, P. and LaSota, R. (1994). On reliability of graphs with node failures. *Networks*, **24**, 251–259.

6. Harary, F. (1962). The maximum connectivity of a graph. *Proc. Nat. Acad. Sci. USA*, **48**, 1142–1146.

7. Myrvold, W. Cheung, K.H., Page, L.B. and Perry, J.E. (1991). Uniformly-most reliable graphs do not always exist. *Networks*, **21**, 417–419.

8. Smith, D.H. (1984). Graphs with the smallest number of minimum cut sets. *Networks*, **14**, 47–61.

9. Smith, D.H. (1991). Optimally reliable networks. *Annals of Operations Res.*, **33**, 107–112.

10. Smith, D.H. (1993). Optimally reliable graphs for both vertex and edge failures. *Combinatorics, Probability and Comp.*, **2**, 93–100.

11. Smith, D.H. and Doty, L.L. (1990). On the construction of optimally reliable graphs. *Networks*, **20**, 723–729.

Multi–Function Coding and Modulation for Spread Spectrum and CDMA with Inherent Security

S. Shepherd

Department of Electronic and Electrical Engineering, University of Bradford

1 Introduction

Currently, spreading sequences for spread–spectrum communications and sequences for CDMA (code division multiple access) systems typically use codes whose auto– and cross–correlation properties are optimized for the characteristics of the channel in question. If security is desired, this is added as a further layer of coding. Sequences exist, however, which combine both desirable modulation characteristics and strong cryptographic properties.

The most promising of these appears to be the sequences produced by the Blum Generator, which has been shown to possess a very high degree of randomness [1]. This generator has already been used to provide the key sequence for a public key stream cipher radio scheme [2] and so a natural extension of this is the use of such sequences as spreading codes, which as well as offering the advantages of spread spectrum communications, also exhibit the unbreakable cryptographic properties of the Vernam one–time–pad. In addition, the combination of the modulation encoding with security offers considerable savings in processing and significantly enhances system efficiency. The main problem to be solved in this area now is the synchronization of the sequences. Knowledge of the secret key (the factors of the generator modulus), as well as allowing decryption of the data, is also expected to allow an efficient synchronization algorithm to be implemented.

2 Background

Ideally, a spreading code should possess the following qualities [3]:

- Optimal cross–correlation properties to minimize symbol error rate due to interference.
- Good synchronization (auto–correlation) properties for reliable recovery.
- High algorithmic complexity to prevent unauthorized data recovery.
- Uniform spectral shape of transmitted pulses for design reasons.

A preliminary investigation into the mathematical properties of Blum sequences from the security point of view has already been carried out [4]. The main approach of the current research program described in this paper is to extend the work to an analysis of the spreading and coding properties of these sequences, especially with regard to synchronization and redundancy. This three year program of work is supported by an EPSRC grant.

Existing spreading sequences use arithmetic from integer fields, notably GF(2), GF(q) and GF(2^q). Little appears to have been done on the use of integer rings for these purposes although they exhibit a number of potential advantages:

- The computation of the multiplicative inverse in a finite ring (with sufficiently large modulus) is intractable without knowledge of the factors of the modulus, whereas in a field it is trivial. This is the theoretical basis of all public key ciphers and hence offers inherent cryptographic properties "for free" along with the modulation scheme.
- Given a knowledge of the factors, operations in a ring are easier owing to the Chinese Remainder Theorem, for example the Quisquater–Couvreur speedup algorithm for RSA decryption.
- A ring is less structured than a field, and hence a greater number of sequences of sufficient orthogonality may be possible from a ring of given order than from a field of similar order.

This paper focuses on an investigation of the cross–correlation and auto–correlation properties of sequences generated from integer rings of proven cryptographic strength to assess their usefulness and applicability to CDMA and spread–spectrum communications. Some of the more obvious aspects of the theoretical analysis have already been addressed heuristically, for example, the analysis of the probability of another user (by chance) starting to transmit using the same seed for his spreading code generator as that reached by another user in the process of an existing transmission. This critically determines the symbol error rate. Based on pure chance alone, clearly if the sequence length is of the order of, say, 2^{250} bits then the probability of synchronizing with another user is 2^{-250}, i.e. vanishingly small!

3 Spreading codes

The ideal CDMA system uses a constellation of totally orthogonal codes. The receiver has a complete set of correlators, there is no co–channel interference and the near–far resistance problem does not arise. Totally orthogonal codes, however, are only possible for a fixed timing offset in a completely synchronous system. In practice, systems will generally be asynchronous, and such codes tend to have poor cross–correlation characteristics if synchronization fails. Consequently, codes that are "sufficiently" orthogonal (i.e. have some finite low cross–correlation) are used instead. The spreading codes can be generated by many different methods. For convenience, linear feedback shift registers (LFSR) are often used for simplicity. Algorithmically simple generators, however, can lead to security weaknesses. Should an eavesdropper be sufficiently near to the transmitter to be able to receive the spreading signal, then the interception of a comparatively short block of code can allow recovery of the LFSR feedback arrangement and the complete transmission can be correlated and recovered.

4 Blum sequences

It would clearly be advantageous to use a spreading code with better security properties and it is suggested that such a code might be that produced by the x^2 mod n or Blum generator [1,2,4,5]. This generator uses a modulus n which is a strong Blum integer (i.e. the product of two strong primes each congruent to 3 mod 4) and a seed value x_0. The seed is successively squared mod n and the $\log_2 n$ least significant bits of the result appended to the output bit stream. It has been shown that such a sequence exhibits such excellent randomness properties that, given an unlimited length of the bitstream (less than the period) there is no more chance of guessing whether the next bit at either end is a one or a zero than by tossing a fair coin. Cryptographically, the sequence is thus equivalent to a truly random one and yet is completely deterministic and easily regenerated by the legitimate receiver.

There are only two ways for an eavesdropper to determine the Blum sequence:

- Factorize the modulus n. For sufficiently large n, this is an intractable problem in number theory. Using the most efficient general–purpose factoring algorithms (multiple polynomial quadratic sieve), estimated times for factoring on a Cray II are given in Table 1.

- Cycle forwards through the sequence until it starts to repeat. A Blum generator, for a suitable choice of seed and modulus, has a maximum length sequence (MLS) of at least $\lambda(\lambda(n))$ where λ is the number–theoretic Carmichael function. Since $\lambda(n)$ is of the order of n, so is $\lambda(\lambda(n))$. Thus, the sequence period can easily be arranged to be sufficiently long to preclude this attack. For example, if n is a 256–bit quantity, the period of an MLS would be of the order of 2^{250} bits. Using fast hardware to produce a residue in 10 microseconds then 10^5 residues could be computed per second. Thus, computing the entire cycle would take 2^{245} seconds or over 10^{62} years. Clearly, attempting to factorize the modulus is preferable!

Table 1. Integer factorization times

Size of n in Bits	Factorization Time
64	10 seconds
128	20 seconds
192	11 minutes
256	11 hours
320	25 days
384	4.2 years
448	253 years
512	16,000 years
585	1 million years

5 Main research areas

The key points of the research are:

- An analysis of the auto–correlation properties of individual blocks of a Blum sequence to determine whether reliable recovery and hence an acceptable symbol error rate is possible.
- An analysis of the cross–correlation properties between different blocks of a Blum sequence to determine whether co–channel rejection and hence an acceptable symbol error rate is possible. This will depend on the probability distribution of the cross–correlations. These two areas have been addressed initially by some simulations. Given the highly random nature of the sequences, it is likely that the correlation properties will be acceptable and this has been confirmed by the simulation results.
- An analysis of the algorithmic complexity of the Blum generator. This is important for operational reasons as the algorithmic complexity limits the rate at which the residues can be generated and hence limits the chip rate of the system. Typical speeds that have been attained are 1000 residues per second in software (on a PC) and 100,000 residues per second in custom hardware.
- Confirmation of the uniformity of Blum sequence spectral shape. Given the fact that the Blum sequences exhibit all the characteristics of a totally random sequence, it is likely that this requirement will be satisfied but it is important to analyze second and higher order statistics as well in order to avoid adjacent channel interference and other problems.

- An investigation of the problems of receiver synchronization. This is probably the most important and difficult problem. With the very long codes involved, it is necessary to reduce the search range. However, the property of Blum sequences that (the receiver's) knowledge of the factors of the modulus allows random access to any point in the sequence, is the key to synchronization. During the call setup, a starting seed and offset may be openly exchanged. Knowledge of the factors allows the receiver to rapidly jump to the new point in the sequence which might be, say, 10^{60} chips downstream. An eavesdropper, however, has no option but to generate the intervening 10^{60} chips until he reaches the correct point in the sequence. This clearly infeasible. Given a typical propagation delay spread of 20:1, the receiver only has to search 20 or so correlations until the correct one is found.

- An analysis of possible schemes whereby "good" seeds that produce sequences with the desirable properties described can be selected. The ability to do this without undue effort is important for operational reasons. Although the maximum length sequences produced by the Blum generator are of period $\lambda(\lambda(n))$, it is possible to find certain "pathological" seeds that generate very much shorter sequences. Trivially, 1 would clearly be a poor choice! It is important to avoid these bad seeds for all the reasons given above.

6 Applications

Secure spreading codes have wide application in a number of areas:

- **Personal Communications.** The possible use of CDMA techniques for personal communications is the subject of a very large current LINK research project. The use of spreading codes with cryptographic properties would considerably enhance the value-added attractions of such a system from the users' point of view.

- **Frequency Hopping.** Up to now, only direct spreading applications have been considered. The ideas described could also be applied to frequency hopping systems. Here, the carrier frequency is selected from a range of, say, 256 possible frequencies and changed every so often under the control of the spreading sequence. This is typically every 10 microseconds for fast hopping and every millisecond for slow hopping. The slow hopping rate is possible in software on a PC. Given the current state of fast multiplier hardware technology, a 256-bit residue can be generated in 10 microseconds. Thus 10^5 residues can be generated per second, each contributing 8 bits to the spreading sequence. Thus, a maximum spreading rate of 800 kbps is possible for fast hopping. It may be possible to combine the Blum sequence with code inversion of a LFSR sequence on, say, a 16:1 basis to gain the advantage of a higher spreading rate (12.8 Mbps) while retaining the randomness of the Blum sequence for security.

References

1. Blum, L., Blum, M. and Shub, M. (1986). A simple unpredictable pseudo-random number generator. *SIAM J. Comp.*, **15**, 364–383.

2. Shepherd, S. (1994). Public key stream ciphers. *IEE Colloquium on Security and Crypt. App. to Radio Sys.*, Digest 1994/141, Institution of Electrical Engineers, London.

3. Vajda, I. (1990). On random code-hopping DS/SSMA system. *IEEE Internat. Sym. Spread Spectrum Techniques and Apps.*, London.

4. Shepherd, S., Sanders, P. and Stockel, C. (1993). The quadratic residue cipher and some notes on implementation. *Cryptologia*, **XVII**, 264–282.

5. Shepherd, S. (1992). A distributed security architecture for large scale systems. *PhD thesis*, University of Plymouth.

Multi–Stage Scheduling Problems with Precedence Constraints

V.A. Strusevich

University of Greenwich, London

1 Introduction

In the shop scheduling models, we are given a set $\mathcal{M} = \{M_1, M_2, \dots, M_m\}$ of machines, $m \geq 2$, and a set $\mathcal{J} = \{J_1, J_2, \dots, J_n\}$ of jobs. Each job J_j consists of m_j operations $O_{1j}, O_{2j}, \dots, O_{m_j j}$ each to be processed on a specified machine of set $\mathcal{M}$. The processing time of an operation O_{ij} equals p_{ij} time units. It is assumed that each machine processes at most one job at a time, and each job is processed on at most one machine at a time.

The most well–known shop scheduling model is the *flow shop.* Here, each job J_j is first processed on machine M_1, then on machine M_2, and so on, until it is processed on machine M_m. In what follows, the order in which a job has to pass the machines is called the *processing route.* Thus, in the flow shop, all jobs are given the same processing route $(M_1, M_2, \dots, M_m)$.

A more general situation is known as *job shop.* For this model, the processing routes of the jobs are assumed to be known in advance, but those need not be the same. Moreover, for the job shop, a job need not visit all machines of set $\mathcal{M}$, however, it may visit some machines more than once.

Many practical situations lead to the necessity of studying so–called *open shops.* The point of difference between this model and the flow shop is that, in the open shop, the order of processing operations of a job is immaterial and must be found, different jobs being allowed to get different orders.

These three basic models can be generalized by combining some or all of them.

For example, combining the flow shop and the open shop, we obtain the model which is known as the *mixed shop.* More precisely, for the mixed shop, it is assumed that the set $\mathcal{J}$ of jobs is partitioned into two non–empty subsets $\mathcal{J}_0$ and $\mathcal{J}_1$. The jobs of the set $\mathcal{J}_0$ have non–fixed orders of their operations (as in an open shop), while all jobs of the set $\mathcal{J}_1$ have the processing route $(M_1, M_2, \dots, M_m)$ (as in a flow shop).

One of the most general shop scheduling models which covers all the previous ones is called the *super shop.* This is obtained as a result of combining the open shop and the job shop. According to that model, set $\mathcal{J}$ is partitioned into $r+1$ non–empty subsets $\mathcal{J}_0, \mathcal{J}_1, \mathcal{J}_2, \dots, \mathcal{J}_r, r \geq 2$. The jobs of set $\mathcal{J}_0$ have non–fixed orders of processing their operations (as in an open shop), while the jobs of a set $\mathcal{J}_q, 1 \leq q \leq r$, have the processing route L_q; some machines of set $\mathcal{M}$ may not occur in a sequence L_q, while, on the other hand, some of them may occur more than once (as in a job shop).

For any shop scheduling model, preemption may or may not be allowed. If preemption is not allowed, then there is no interruption in the processing of each operation. On the other hand, if preemption is allowed, then the processing of any job on any machine may be interrupted at any time and resumed later. If the operations of a job have to be processed in the prescribed order, then, irrespective of whether preemption is allowed the next operation of the job can only start after the previous operation has been completed. If the order of the operations of a job is immaterial, then, in between an interruption and a resumption of the processing of an operation any other unfinished operation of this job may be processed. In any case, the total length of all time intervals in which an operation O_{ij} is processed is equal to the given processing time p_{ij}.

In this paper, it is assumed that the jobs are not independent, and there is a precedence relation imposed over set $\mathcal{J}$. We distinguish between two types of precedence constraints, given by the relations $\rightarrow$ and $\Rightarrow$, respectively.

For two jobs J_j and J_k, we write $J_j \rightarrow J_k$ and say that job J_j precedes job J_k if and only if job J_k cannot start on *any* machine until job J_j is completed on *all* machines it has to be processed on. The relation $\rightarrow$ will be called the *precedence relation of the first type.* If $J_j \rightarrow J_k$ and there is no job J_i such that $J_j \rightarrow J_i \rightarrow J_k$, then job J_j is said to *directly precede* job J_k, and this is denoted by $J_j \xrightarrow{d} J_k$.

On the other hand, for two jobs J_j and J_k, we write $J_j \Rightarrow J_k$ and say that job J_j precedes job J_k if and only if for any $i, 1 \leq i \leq m$, job J_k cannot start on machine M_i until job J_j is completed on M_i. The relation $\Rightarrow$ will be called the *precedence relation of the second type.* If $J_j \Rightarrow J_k$ and there is no job J_i such that $J_j \Rightarrow J_i \Rightarrow J_k$, then job J_j is said to *directly precede* job J_k, and this is denoted by $J_j \overset{d}{\Rightarrow} J_k$.

We assume that precedence constraints are given by a so-called *reduction graph.* We denote the reduction graphs corresponding to relations $\rightarrow$ and $\Rightarrow$ by $\vec{G}$ and $\overset{\Rightarrow}{G}$, respectively. The set of vertices of the reduction graph coincides with the set of jobs, and there is an arc going from a vertex J_j to vertex J_k if and only if job J_j directly precedes job J_k.

The remainder of this paper is organized as follows. Section 2 gives notation and presents a short overview of the complexity of shop scheduling problems with no precedence constraints. The problems under precedence constraints of the first type are considered in Section 3. Section 4 is devoted to studying the problems under precedence constraints of the second type. Some concluding remarks are contained in Section 5.

Because of lack of space, we normally give only sketches of the proofs. Missing details can be found in [19].

2 Preliminaries

To make an easy reference to a problem, we follow standard notation for scheduling problems based on the three–field classification scheme of the form $\alpha|\beta|\gamma$, see, for example [11]. Here, the first position α stands for the machine environment, or the type of a shop. We use $\alpha \in \{F, J, O, X, S\}$ in the following way: F – flow shop; J – job shop; O – open shop; X – mixed shop; S – super shop.

As a rule, the number of machines in a shop is indicated explicitly. For example, $\alpha = F2$ denotes the two–machine flow shop. If the number m of machines is not shown, it is assumed to be variable.

The second position β of the classification scheme describes the processing conditions. If this position is left empty, this implies that preemption is forbidden, and no precedence constraints are imposed. We write "*pmtn*" in this position if preemption is allowed. Also, we write either "$prec_1$" or "$prec_2$" to indicate that there are precedence constraints either of the first or of the second type, respectively.

Let m_j denote the number of operations of a job J_j. We write "$m_j \leq m'$" in the position β if the number of operations of any job does not exceed a given number m'.

The third position γ specifies the objective function. In what follows, $\gamma = C_{\max}$, i.e, we minimize the *makespan* or the maximum completion time of all jobs on all machines. A schedule that minimizes the makespan is called (*time*–) *optimal.*

In our study, we often deal with the two–machine shop scheduling systems of two–stage processing. In that case, special notation is used. The machines are denoted by A and B, respectively. For a job J_j, the processing times on A and B are a_j and b_j, respectively. The set of jobs $\mathcal{J}$ is divided into three subsets $\mathcal{J}_{AB}, \mathcal{J}_{BA}$ and $\mathcal{J}_O$. Here, each job of set $\mathcal{J}_{AB}$ is first processed on machine A and then on machine B, i.e., has the processing route (A, B); each job of set $\mathcal{J}_{BA}$ has the processing route (B, A); set $\mathcal{J}_O$ contains the jobs for which the processing route can be either (A, B) or (B, A) and is not fixed in advance. The jobs that have to be processed on exactly one of the machines are also assumed to belong to the set $\mathcal{J}_O$.

For a non–empty set of jobs Q, we define

$$a(Q) = \sum_{J_j \in Q} a_j, b(Q) = \sum_{J_j \in Q} b_j,$$

while $a(\emptyset) = b(\emptyset) = 0$.

Our main goal is to provide a sharp borderline between "easy", i.e., polynomially solvable problems, and "difficult", i.e., NP–hard problems. For the later problems, the existence of a polynomial–time algorithm is unlikely.

First, recall some results on relevant scheduling problems with no precedence constraints imposed.

The $F2||C_{\max}$ and $J2|m_j \leq 2|C_{\max}$ problems are solvable in $O(n \log n)$ time due to Johnson [9] and Jackson [8], respectively. The $O2||C_{\max}$ problem can be solved in $O(n)$ time due to Gonzalez and Sahni [6]. Masuda et al. [13] give an $O(n \log n)$ time algorithm for solving the $X2||C_{\max}$ problem. Note that for all these problems preemption, if allowed, does not reduce the makespan, i.e., the corresponding algorithms also solve the preemptive counterparts of the mentioned problems.

Both $S2|m_j \leq 2|C_{\max}$ and $S2|m_j \leq 2, pmtn|C_{\max}$ problems are solvable in $O(n \log n)$ time due to Strusevich [19,20]. For this model, an optimal preemptive schedule may be shorter than a non–preemptive one.

Several polynomial–time algorithms are known for the $O|pmtn|C_{\max}$ problem (see, for example, [6]).

We now give a brief overview of "difficult" shop scheduling problems, the terminology can be found in [4].

Both $F3||C_{\max}$ and $F3|pmtn|C_{\max}$ problems are NP–hard in the strong sense, as proved by Garey et al. [5] and by Gonzalez and Sahni [7], respectively. Both $J3|m_j \leq 2|C_{\max}$ and $J2|m_j \leq 3|C_{\max}$ problems, as well as their preemptive counterparts are NP–hard in the strong sense (see [7,12]). The $O3||C_{\max}$ problem is NP–hard in the ordinary sense [6], while the general $O||C_{\max}$ is NP–hard in the strong sense due to J.K. Lenstra (see, for example, [11]).

In this paper, to prove the NP–hardness of a scheduling problem, we use two well–known problems, PARTITION and 3–PARTITION. The former problem is NP–complete in the ordinary sense, while the later is NP–complete in the strong sense.

PARTITION. Given r positive integers $e_i, i \in R = \{1, 2, \ldots, r\}$, and an integer E such that $\sum_{i \in R} e_i = 2E$, does there exist a partition of set R into two subsets R_1 and R_2 such that $\sum_{i \in R_1} e_i = \sum_{i \in R_2} e_i = E$?

3–PARTITION. Given $3r$ positive integers $e_i, i \in R = \{1, 2, \ldots, 3r\}$, and an integer E such that $\sum_{i \in R} e_i = rE$ and $E/4 < e_i < E/2$, does there exists a partition of set R into r three–element subsets R_j such that $\sum_{i \in R_j} e_i = E$ for all $j = 1, 2, \ldots, r$?

To prove that a scheduling problem to minimize the makespan is NP–hard, we consider the corresponding *decision* problem, i.e. the problem of determining whether there exists a schedule S_0 such that $C_{\max}(S_0) \leq y$ for a given y. We transform either PARTITION or 3–PARTITION to a specific instance of the decision counterpart of a scheduling problem in question, and show that in the resulting problem schedule S_0 exists if and only if there exists the desired partition (or 3–partition). To prove the NP–hardness in the ordinary sense, we start with PARTITION and show that the required transformation takes time bounded by a polynomial in r. To prove the NP–hardness in the strong sense, we start with 3–PARTITION and show that the required transformation takes time bounded by a polynomial in r and E.

3 Precedence relation of the first type

In this section we study the complexity of the $S|prec_1|C_{\max}$ and $S|prec_1, pmtn|C_{\max}$ problems under various assumptions on the structure of set $\mathcal{J}$ and the reduction graph $\vec{G}$.

We only consider reduction graphs of a chain-like structure, which happens to be enough to give a complete complexity classification of the problems under consideration. Some of the results presented below can be found in [20].

3.1 Linear order

We start with considering the trivial case, assuming that the set of jobs is *linearly ordered*, i.e, the reduction graph $\vec{G}$ is a single chain.

It is obvious that at any time at most one operation can be processed. Therefore, a non-preemptive schedule S^* that is optimal for both $S|prec_1|C_{\max}$ and $S|prec_1, pmtn|C_{\max}$ problems can be found as follows. The jobs are processed in the prescribed order and according to their processing routes; for the jobs of set $\mathcal{J}_0$ the processing route can be arbitrary, for example, $(M_1, M_2, \dots, M_m)$; each operation starts as soon as possible. It is clear that the time required for finding schedule S^* is $O(n\sum_{j=1}^{n} m_j)$.

3.2 Single chain constraints

We now consider a more general situation, assuming that the reduction graph $\vec{G}$ contains a single chain as well as a number of isolated vertices. In this case, we denote the linearly ordered subset of jobs corresponding to the chain by $\vec{\mathcal{J}}$, while the subset of non-ordered jobs is denoted by $\tilde{\mathcal{J}}$.

The $F2|prec_1|C_{\max}$ problem is proved to be NP-hard in the ordinary sense even if just two jobs are ordered, see [12]. The proof can easily be extended to show that in fact both $F2|prec_1|C_{\max}$ and $F2|prec_1, pmtn|C_{\max}$ problems are NP-hard in the strong sense if set $\mathcal{J} = \mathcal{J}_{AB}$ is split into two subsets $\mathcal{J}_{AB}^1$ and $\mathcal{J}_{AB}^2$, such that one of them is linearly ordered and the other is not ordered, i.e.,

$$\mathcal{J} = \mathcal{J}_{AB} = \mathcal{J}_{AB}^1 \cup \mathcal{J}_{AB}^2, \vec{\mathcal{J}} = \mathcal{J}_{AB}^1, \tilde{\mathcal{J}} = \mathcal{J}_{AB}^2.$$

The $O2|prec_1|C_{\max}$ problem is considered in [1], see also [22], Chapter 3, Section 4.1. The problem is shown to be NP-hard in the strong sense if

$$\mathcal{J} = \mathcal{J}_O = \mathcal{J}_O^1 \cup \mathcal{J}_O^2, \vec{\mathcal{J}} = \mathcal{J}_O^1, \tilde{\mathcal{J}} = \mathcal{J}_O^2.$$

These results imply that in the non-preemptive case any two-machine problem to minimize the makespan is NP-hard in the strong sense, if some (but not all) of the jobs having the same fixed route (or non-fixed route) are linearly ordered while the other jobs are not ordered.

Suppose now that preemption is allowed and consider the $S|prec_1, pmtn|C_{\max}$ problem, provided that set $\mathcal{J}$ is composed as follows. The set $\mathcal{J}_0$ of the jobs with non-fixed routes is split into two subsets $\mathcal{J}_0^1$ and $\mathcal{J}_0^2$, so that the jobs of $\mathcal{J}_0^2$ are not ordered, while all other jobs are linearly ordered, i.e.,

$$\mathcal{J}_0 = \mathcal{J}_0^1 \cup \mathcal{J}_0^2, \vec{\mathcal{J}} = \mathcal{J}_0^1 \cup \mathcal{J}_1 \cup \ldots \cup \mathcal{J}_r, \tilde{\mathcal{J}} = \mathcal{J}_0^2. \qquad (3.1)$$

It can be shown that there exists an optimal schedule in which the jobs of set $\vec{\mathcal{J}}$ are processed without preemption.

Theorem 1. *The $S|prec_1, pmtn|C_{\max}$ problem is solvable in polynomial time, provided that set $\mathcal{J}$ is composed as in Equation 3.1.*

Proof. Denote a non-preemptive time-optimal schedule for processing the jobs of set $\vec{\mathcal{J}}$ by $S^*(\vec{\mathcal{J}})$. This schedule can be found as described in Section 3.1.

Let the total number of operations of the jobs of set $\vec{\mathcal{J}}$ be equal to W. Assume that the operations are numbered in such a way that in schedule $S^*(\vec{\mathcal{J}})$ they are processed in the sequence $1, 2, \ldots, W$. Since in $S^*(\vec{\mathcal{J}})$ these operations do not overlap, we may number the corresponding time intervals by the integers $1, 2, \ldots, W$, assuming that an operation V is processed in the interval $V, 1 \leq V \leq W$. Also, if an operation O_{qk} is numbered as V, we refer to machine M_q as $M(V)$, and to job J_k as $J(V)$.

We show that schedule $S^*(\vec{\mathcal{J}})$ for processing the jobs of set $\vec{\mathcal{J}}$ can be extended to an optimal schedule S^* for processing all jobs of set $\mathcal{J}$.

Let $x_{ij}(V)$ denote the total time of processing job $J_j \in \tilde{\mathcal{J}}$ on machine M_i in the time interval $V, 1 \leq V \leq W$. Also, for a positive C, let y_{ij} be the total time for processing job $J_j \in \tilde{\mathcal{J}}$ on machine M_i in the time interval $[C_{\max}(S^*(\vec{\mathcal{J}})), C_{\max}(S^*(\vec{\mathcal{J}})) + C]$.

Consider the following linear programming problem:

$$\text{minimize } C$$
$$\text{s.t.}$$

$$\sum_{i=1, i\neq q}^{m} x_{ij}(V) \le p_{qk},$$

$$J_j \in \tilde{\mathcal{J}}, J_k = J(V), M_q = M(V), V = 1, 2, \ldots, W;$$

$$\sum_{J_j \in \tilde{\mathcal{J}}} x_{ij}(V) \le p_{qk},$$

$$i = 1, 2, \ldots, m, i \neq q, J_k = J(V), M_q = M(V), V = 1, 2, \ldots, W;$$

$$\sum_{i=1}^{m} y_{ij} \le C, J_j \in \tilde{\mathcal{J}}; \sum_{J_j \in \tilde{\mathcal{J}}} y_{ij} \le C, i = 1, 2, \ldots, m;$$

$$\sum_{V=1}^{W} x_{ij}(V) + y_{ij} = p_{ij}, i = 1, 2, \ldots, m; J_j \in \tilde{\mathcal{J}};$$

$$C \ge 0; y_{ij} \ge 0, i = 1, 2, \ldots, m; J_j \in \tilde{\mathcal{J}};$$

$$x_{ij}(V) = 0, J_j \in \tilde{\mathcal{J}}, M_i = M(V), V = 1, 2, \ldots, W;$$

$$x_{ij}(V) \ge 0, J_j \in \tilde{\mathcal{J}}, i = 1, 2, \ldots, m, M_i \neq M(V), V = 1, 2, \ldots, W.$$

Suppose that $\widehat{C}$ is the optimal value of C, while $\hat{x}_{ij}(V)$ and $\hat{y}_{ij}$ are the values of variables $x_{ij}(V)$ and y_{ij}, respectively, for which the value $C = \widehat{C}$ is achieved, $J_j \in \tilde{\mathcal{J}}$, $i = 1, 2, \ldots, m, V = 1, 2, \ldots, W$.

For each $V, 1 \le V \le W$, solve the $O|pmtn|C_{\max}$ problem of processing the jobs of set $\tilde{\mathcal{J}}$ in the time interval V, assuming that the processing times are equal to $\hat{x}_{ij}(V)$,$J_j \in \tilde{\mathcal{J}}, i = 1, 2, \ldots, m$. If $\widehat{C} > 0$, then solve the $O|pmtn|C_{\max}$ problem of processing the jobs of set $\tilde{\mathcal{J}}$ in the time interval $[C_{\max}(S^*(\vec{\mathcal{J}}))$, $C_{\max}(S^*(\vec{\mathcal{J}})) + \widehat{C}]$, assuming that the processing times are equal to $\hat{y}_{ij}, J_j \in \tilde{\mathcal{J}}$. Concatenating the corresponding partial schedules, we obtain the desired optimal schedule S^* with $C_{\max}(S^*) = C_{\max}(S^*(\vec{\mathcal{J}})) + \widehat{C}$.

Notice that finding the schedule $S^*(\vec{\mathcal{J}})$ takes polynomial time. The linear programming problem can be solved in polynomial time. The $O|pmtn|C_{\max}$ problem is polynomially solvable, and the number of these problems to be solved depends polynomially on the number of operations of jobs in set $\vec{\mathcal{J}}$. Therefore, we conclude that the problem under consideration is solvable in polynomial time. The theorem is proved.

We now consider a special case of the previous problem with $m = 2$. We use the notation accepted for the two–machine shop problems. Thus, we assume that set $\mathcal{J}_O$ is split into two subsets $\mathcal{J}_O^1$ and $\mathcal{J}_O^2$, so that $\vec{\mathcal{J}}= \mathcal{J}_{AB} \cup \mathcal{J}_{BA} \cup \mathcal{J}_O^1$ and $\tilde{\mathcal{J}}= \mathcal{J}_O^2$.

Theorem 2. *The $S2|prec_1, pmtn, m_j \leq 2|C_{\max}$ problem is solvable in $O(n)$ time, if $\vec{\mathcal{J}}= \mathcal{J}_{AB} \cup \mathcal{J}_{BA} \cup \mathcal{J}_O^1$ and $\tilde{\mathcal{J}}= \mathcal{J}_O^2$.*

Proof. Suppose that $| \vec{\mathcal{J}} | = n_1$ and that the jobs of the linearly ordered set $\vec{\mathcal{J}}$ are numbered in such a way that $J_j \xrightarrow{d} J_{j+1}, j = 1, 2, \ldots, n_1 - 1$. Assume that the jobs in $\tilde{\mathcal{J}}$ are numbered arbitrarily and the jobs of set $\vec{\mathcal{J}}$ are processed with no preemption.

For any feasible schedule S, we have

$$C_{\max}(S) \geq T = \max\{a(\mathcal{J}), b(\mathcal{J}), a(\vec{\mathcal{J}}) + b(\vec{\mathcal{J}}), \max\{a_j + b_j | J_j \in \tilde{\mathcal{J}}\}\}.$$

Thus, if we find a schedule that meets this lower bound, this schedule will be optimal.

First, we find a non–preemptive schedule $S^*(\vec{\mathcal{J}})$ for processing the jobs in set $\vec{\mathcal{J}}$ as described in Section 3.1. This requires $O(n_1)$ time.

We need an algorithm for scheduling the jobs of set $\tilde{\mathcal{J}}$.

The desired algorithm can be derived from the algorithm by Lawler et al. [10] for solving the two–machine preemptive open shop scheduling problem to minimize maximum lateness. That algorithm actually finds a preemptive open shop schedule with no late jobs with respect to specially defined deadlines. Our super shop problem can also be viewed as that of finding a schedule with no late jobs, provided that a job $J_k \in \vec{\mathcal{J}}$ is given the deadline $D_k = \sum_{j=1}^{k} (a_j + b_j), k = 1, 2, \ldots, n_1$, while all jobs of set $\tilde{\mathcal{J}}$ are given the common deadline $D = T$. A slight difference between this problem and that from [10] is that in our case some jobs may have fixed routes.

In general the running time of the algorithm from [10] is $O(n \log n)$. However, if the deadlines are sorted in non–decreasing order, the algorithm runs in $O(n)$ time. Since in our case the jobs are numbered in the required way beforehand, we conclude that the problem under consideration is solvable in $O(n)$ time. The theorem is proved.

We now consider the $S2|prec_1, m_j \leq 2|C_{\max}$ and $S2|prec_1, pmtn, m_j \leq 2|C_{\max}$ problems, provided that exactly one of the sets $\mathcal{J}_{AB}$, $\mathcal{J}_{BA}$ or $\mathcal{J}_O$ is empty, one of the remaining sets is linearly ordered, and the other set is not ordered.

Theorem 3. *The* $S2|prec_1, m_j \leq 2|C_{\max}$ *and* $S2|prec_1, pmtn, m_j \leq 2|C_{\max}$ *problems are* NP*-hard in the strong sense if* $\tilde{\mathcal{J}} = \mathcal{J}_{AB}$ *and either* $\vec{\mathcal{J}} = \mathcal{J}_{BA}, \mathcal{J}_O = \emptyset$ *or* $\vec{\mathcal{J}} = \mathcal{J}_O, \mathcal{J}_{BA} = \emptyset$.

Proof. We give a sketch of a reduction scheme of 3–PARTITION to the problems under consideration.

Let the set $\mathcal{J}$ consist of $n = 4r$ jobs divided into two groups: U–jobs denoted by $U_i, i = 1, 2, \ldots, 3r$, and V–jobs denoted by $V_j, j = 1, 2, \ldots, r$. The processing times are set equal to $a_{U_i} = b_{U_i} = e_i, i = 1, 2, \ldots, 3r; a_{V_j} = b_{V_j} = E, j = 1, 2, \ldots, r$.

Define $\tilde{\mathcal{J}} = \mathcal{J}_{AB} = \{U_1, U_2, \ldots, U_{3r}\}$ and $\vec{\mathcal{J}} = \{V_1, V_2, \ldots, V_r\}$. The precedence relation $\rightarrow$ is defined in such a way that $V_j \xrightarrow{d} V_{j+1}, j = 1, 2, \ldots, r-1$.

It can be shown that if either $\vec{\mathcal{J}} = \mathcal{J}_{BA}, \mathcal{J}_O = \emptyset$ or $\vec{\mathcal{J}} = \mathcal{J}_O, \mathcal{J}_{BA} = \emptyset$, then in the constructed problem a schedule S_0, preemptive or not, such that $C_{\max}(S_0) \leq y = 2rE$ exists if and only if 3–PARTITION has a solution.

Theorem 4. *The* $S2|prec_1, m_j \leq 2|C_{\max}$ *problem is* NP*-hard in the strong sense if* $\vec{\mathcal{J}} = \mathcal{J}_{AB}, \tilde{\mathcal{J}} = \mathcal{J}_O$.

Proof. Use the same reduction of 3–PARTITION to the corresponding decision problem as in the proof of the previous theorem. The only difference is that here we define $\vec{\mathcal{J}} = \mathcal{J}_{AB} = \{V_1, V_2, \ldots, V_r\}$ and $\tilde{\mathcal{J}} = \mathcal{J}_O = \{U_1, U_2, \ldots, U_{3r}\}$.

3.3 Summary

The above results provide the complete complexity classification of the super shop scheduling problems to minimize the makespan under precedence constraints of the first type.

Table 1.

$\vec{\mathcal{J}}$	$\tilde{\mathcal{J}}$	Complexity	Reference
$\mathcal{J}_O^1$	$\mathcal{J}_O^2$	NP	[1,22]
$\mathcal{J}_{AB}^1$	$\mathcal{J}_{AB}^2$	NP	[12]
$\mathcal{J}_O$	$\mathcal{J}_{AB}$	NP	Theorem 3
$\mathcal{J}_{BA}$	$\mathcal{J}_{AB}$	NP	Theorem 3
$\mathcal{J}_{AB}$	$\mathcal{J}_O$	NP	Theorem 4

Table 2.

$\overrightarrow{\mathcal{J}}$	$\widetilde{\mathcal{J}}$	Complexity	Reference
$\mathcal{J}_{AB} \cup \mathcal{J}_{BA} \cup \mathcal{J}_O^1$	$\mathcal{J}_O^2$	$O(n)$	Theorem 2
$\mathcal{J}_{AB}^1$	$\mathcal{J}_{AB}^2$	NP	[12]
$\mathcal{J}_O$	$\mathcal{J}_{AB}$	NP	Theorem 3
$\mathcal{J}_{BA}$	$\mathcal{J}_{AB}$	NP	Theorem 3

If the set of jobs is linearly ordered, then both $S|prec_1|C_{\max}$ and $S|prec_1, pmtn|C_{\max}$ problems are solvable in polynomial time.

If the reduction graph $\overrightarrow{G}$ contains exactly one chain and possibly a number of isolated vertices, then the $S|prec_1, pmtn|C_{\max}$ problem is solvable in polynomial time by linear programming, provided that the structure of set $\mathcal{J}$ satisfies Equation 3.1 (see Theorem 1). The complexity results for the $S2|prec_1, m_j \leq 2|C_{\max}$ problem and the $S2|prec_1, pmtn, m_j \leq 2|C_{\max}$ problem are given in Tables 1 and 2, respectively. In these tables, we write "NP" to indicate that the corresponding problem is NP-hard in the strong sense.

Suppose that the reduction graph $\overrightarrow{G}$ contains $p \geq 2$ chains and possibly a number of isolated vertices (see Figure 1).

It follows from Section 3.2 that we only need to examine the $S2|prec_1, pmtn, m_j \leq 2|C_{\max}$ problem in which the set $\mathcal{J}_O^1 \neq \emptyset$ is not ordered while the set $\mathcal{J} = \mathcal{J}_{AB} \cup \mathcal{J}_{BA} \cup \mathcal{J}_O^2$ is ordered. However, the $O2|prec_1, pmtn|C_{\max}$ problem under precedence constraints of the required structure is proved to be NP-hard in the strong sense by Borodich and Strusevich [2] (see also [22], Chapter 3, Section 4.5).

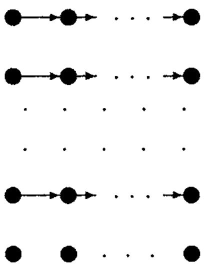


Figure 1.

4 Precedence relation of the second type

In this section we study the complexity of the super shop scheduling problems under precedence constraints of the second type. Recall that these constraints are specified by the relation $\Rightarrow$, and we write $J_j \Rightarrow J_k$ to indicate that job J_k cannot be started on any machine until job J_j is completed on that machine. The reduction graph of this relation is denoted by $\overset{\Rightarrow}{G}$.

4.1 Linear order

We start with the simplest situation, in which the set of jobs is linearly ordered, i.e., the graph $\overset{\Rightarrow}{G}$ is a chain. Assume that the jobs are numbered in such a way that

$$J_1 \overset{d}{\Rightarrow} J_2 \overset{d}{\Rightarrow} \dots \overset{d}{\Rightarrow} J_n.$$

First, consider the $S|prec_2|C_{\max}$ problem with $\mathcal{J}_0 = \emptyset$, i.e., the $J|prec_2\ |C_{\max}$ problem. Since there is a unique feasible sequence of the jobs for each machine, and each job is defined by a unique sequence of operations, we conclude that an optimal schedule can be found in $O(n \sum_{j=1}^{n} m_j)$ time by starting each operation as soon as possible. Moreover, observe that in this case preemption, if allowed, may not reduce the makespan, therefore the $J|prec_2, pmtn|C_{\max}$ problem with the linearly ordered set of jobs is also polynomially solvable.

Now consider the $Om|prec_2|C_{\max}$ problem with the linearly ordered set of jobs. By interchanging the set of jobs and the set of machines, we see that the original problem is equivalent to the $Fn||C_{\max}$ problem of processing the jobs $M_1, M_2, \dots, M_m$ on the machines $J_1, J_2, \dots, J_n$ in this order. Similarly, the $Om|prec_2, pmtn|C_{\max}$ problem with the linearly ordered set of jobs becomes equivalent to the $Fn|pmtn|C_{\max}$ problem of processing the jobs $M_1, M_2, \dots, M_m$ on the machines $J_1, J_2, \dots, J_n$.

Sotskov [18] proves that both $F||C_{\max}$ and $F|pmtn|C_{\max}$ problems are NP–hard in the ordinary sense if the number of jobs is three and the number of machines is variable. This implies that if the set of jobs is linearly ordered, then both $O3|prec_2|C_{\max}$ and $O3|prec_2, pmtn|C_{\max}$ problems have the same complexity.

Consider the $O2|prec_2|C_{\max}$ and $O2|prec_2, pmtn|C_{\max}$ problems with the linearly ordered set of jobs. As usual, the machines are denoted by A and B, and the processing times of job J_j on these machines are equal to a_j and b_j, respectively. By interchanging the set of jobs and the set of machines, the original problems become equivalent to the $Fn||C_{\max}$ and $Fn|pmtn|C_{\max}$ problems, respectively, in which two jobs A and B are to be processed on the machines $J_1, J_2, \dots, J_n$ in this order. The last two problems are solvable in polynomial time. In fact, Sotskov [18] proves that essentially more general problems with two jobs are polynomially solvable in both the preemptive and non–preemptive cases. See also [3]. Applying Sotskov's algorithm to the flow shop problems with two jobs, we obtain that if the set of jobs is linearly ordered, then the $O2|prec_2|C_{\max}$

and $O2|prec_2, pmtn|C_{\max}$ problems are solvable in $O(n \log n)$ and $O(n^2)$ time, respectively. Note that for these problems preemption, if allowed, may reduce the makespan.

This approach can be extended to the $S2|prec_2, m_j \leq 2|C_{\max}$ or $S2|prec_2, pmtn, m_j \leq 2|C_{\max}$ problems with linearly ordered set of jobs. Here we have that $\mathcal{J}_{AB} \cup \mathcal{J}_{BA} \neq \emptyset$, therefore in the two–job flow shop problem arising after interchanging the set of jobs and the set of machines, the set of operations is partially ordered.

We skip the details of a slight modification of Sotskov's algorithm that is required to handle this situation. Note that the required changes do not affect the running time. Thus, the following statement holds:

Theorem 5. *If the set of jobs is linearly ordered, the $S2|prec_2, m_j \leq 2|C_{\max}$ and $S2|prec_2, pmtn, m_j \leq 2|C_{\max}$ problems are solvable in $O(n \log n)$ and $O(n^2)$ time, respectively.*

4.2 Single chain constraints

We now consider a more general situation, assuming that the reduction graph $\overset{\Rightarrow}{G}$ contains a single chain as well as a number of isolated vertices. In this case, we denote the linearly ordered subset of jobs corresponding to the chain by $\overset{\Rightarrow}{\mathcal{J}}$, while the subset of non–ordered jobs is denoted by $\tilde{\mathcal{J}}$.

The $O2|prec_2|C_{\max}$ problem with

$$\mathcal{J} = \mathcal{J}_O = \mathcal{J}_O^1 \cup \mathcal{J}_O^2, \overset{\Rightarrow}{\mathcal{J}} = \mathcal{J}_O^1, \tilde{\mathcal{J}} = \mathcal{J}_O^2.$$

is shown to be NP–hard in the strong sense [2], see also [22], Chapter 3, Section 4.6.

Note that the $O2|prec_2, pmtn|C_{\max}$ and the $F2|prec_2|C_{\max}$ problems are polynomially solvable even under more general precedence constrains. See Sections 4.3 and 4.4.

In this section we establish the NP–hardness of several two–machine shop scheduling problems.

Theorem 6. *Both $S2|prec_2, m_j \leq 2|C_{\max}$ and $S2|prec_2, pmtn, m_j \leq 2|\ C_{\max}$ problems are NP-hard in the strong sense if $\mathcal{J} = \mathcal{J}_{BA} \cup \mathcal{J}_{AB} = \mathcal{J}_{BA} \cup \mathcal{J}_{AB}^1 \cup \mathcal{J}_{AB}^2$, $\overset{\Rightarrow}{\mathcal{J}} = \mathcal{J}_{BA} \cup \mathcal{J}_{AB}^1$, $\tilde{\mathcal{J}} = \mathcal{J}_{AB}^2 \neq \emptyset$.*

Proof. Starting with 3–PARTITION, define the following instance of the problems under consideration. Without loss of generality, assume that in the formulation of 3–PARTITION the integer E is even.

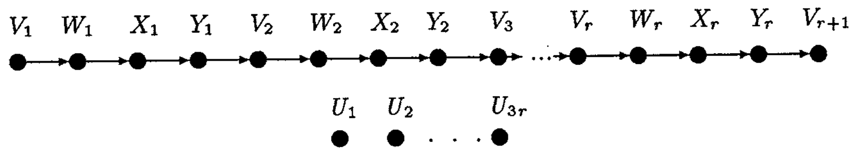


Figure 2.

Let the set $\mathcal{J}$ consist of $n = 7r + 1$ jobs divided into five groups:

U–jobs denoted by $U_i, i = 1, 2, \dots, 3r$;

V–jobs denoted by $V_j, j = 1, 2, \dots, r + 1$;

W–jobs denoted by $W_j, j = 1, 2, \dots, r$;

X–jobs denoted by $X_j, j = 1, 2, \dots, r$;

Y–jobs denoted by $Y_j, j = 1, 2, \dots, r$.

The processing times are set equal to

$a_{U_i} = b_{U_i} = e_i, = 1, 2, \dots, 3r$;

$a_{V_j} = E/2, b_{V_j} = E, j = 1, 2, \dots, r + 1$;

$a_{W_j} = E, b_{W_j} = E/2, j = 1, 2, \dots, r$;

$a_{X_j} = E, b_{X_j} = E/2, j = 1, 2, \dots, r$;

$a_{Y_j} = E/2, b_{Y_j} = E, j = 1, 2, \dots, r$.

Define $\tilde{\mathcal{J}} = \mathcal{J}^2_{AB} = \{U_1, U_2, \dots, U_{3r}\}$ and $\mathcal{J}^1_{AB} = \{V_1, V_2, \dots, V_{r+1}, W_1, W_2, \dots, W_r\}$, $\mathcal{J}_{BA} = \{X_1, X_2, \dots, X_r, Y_1, Y_2, \dots, Y_r\}$, $\overset{\Rightarrow}{\mathcal{J}} = \mathcal{J}_{BA} \cup \mathcal{J}^1_{AB}$. The precedence relation $\Rightarrow$ is defined as shown in Figure 2.

It can be shown that in the constructed problem a schedule S_0, preemptive or not, such that $C_{\max}(S_0) \le y = (4r + 3/2)E$ exists if and only if 3–PARTITION has a solution.

Theorem 7. *The $S2|prec_2, pmtn|C_{\max}$ problem is NP–hard in the ordinary sense if $\mathcal{J} = \mathcal{J}_{AB} \cup \mathcal{J}_O$, $\overset{\Rightarrow}{\mathcal{J}} = \mathcal{J}_O$, $\tilde{\mathcal{J}} = \mathcal{J}_{AB}$.*

Proof. Starting with PARTITION, define the following instance of the problem under consideration.

Let the set $\mathcal{J}$ consist of $n = r + 5$ jobs divided into two groups: U–jobs denoted by $U_i, i = 1, 2, \dots, r$, and V–jobs denoted by $V_j, j = 1, 2, \dots, 5$.

The processing times are set equal to

$$a_{U_i} = b_{U_i} = 2e_i, = 1, 2, \dots, r;$$

$$a_{V_j} = 2E, b_{V_j} = 4E, j \in \{1, 3, 5\};$$

$$a_{V_j} = 6E, b_{V_j} = E, j \in \{2, 4\}.$$

Define $\tilde{\mathcal{J}} = \mathcal{J}_{AB} = \{U_1, U_2, \dots, U_{3r}\}$ and $\overset{\Rightarrow}{\mathcal{J}} = \mathcal{J}_O = \{V_1, V_2, \dots, V_5\}$. The precedence relation $\Rightarrow$ is defined in such a way that job $V_j \overset{d}{\Rightarrow} V_{j+1}, j = 1, 2, 3, 4$.

It can be shown that in the constructed problem a schedule S_0, such that $C_{\max}(S_0) \le y = 20E$ exists if and only if PARTITION has a solution.

Theorem 8. *The $S2|prec_2|C_{\max}$ problem is NP-hard in the ordinary sense if $\mathcal{J} = \mathcal{J}_{AB} \cup \mathcal{J}_O$, $\overset{\Rightarrow}{\mathcal{J}} = \mathcal{J}_O$, $\tilde{\mathcal{J}} = \mathcal{J}_{AB}$.*

Proof. We present the reduction scheme of PARTITION to the decision counterpart of the problem under consideration. Without loss of generality, assume that in the formulation of PARTITION the integer E is even.

Let the set $\mathcal{J}$ consist of $n = r + 10$ jobs divided into two groups: U-jobs denoted by $U_i, i = 1, 2, \dots, r$, and V-jobs denoted by $V_j, j = 1, 2, \dots, 10$.

The processing times are set equal to

$$a_{U_i} = b_{U_i} = e_i, = 1, 2, \dots, r;$$

$$a_{V_j} = E/2, b_{V_j} = 3E, j \in \{1, 4, 5, 8, 9\};$$

$$a_{V_j} = 3E, b_{V_j} = E/2, j \in \{2, 3, 6, 7, 10\}.$$

Define $\tilde{\mathcal{J}} = \mathcal{J}_{AB} = \{U_1, U_2, \dots, U_{3r}\}$ and $\overset{\Rightarrow}{\mathcal{J}} = \mathcal{J}_O = \{V_1, V_2, \dots, V_{10}\}$. The precedence relation $\Rightarrow$ is defined in such a way that job $V_j \overset{d}{\Rightarrow} V_{j+1}, j = 1, 2, \dots, 9$.

It is not difficult to show that in the constructed problem a schedule S_0, such that $C_{\max}(S_0) \le y = 20E$ exists if and only if PARTITION has a solution.

4.3 Series–parallel constraints

Let $G(X, U)$ be a (di)graph, where X is the set of vertices and U is the set of arcs.

A graph $G(X, U)$ is said to be *a parallel composition* of two graphs $G_1(X_1, U_1)$ and $G_2(X_2, U_2)$ such that $X_1 \cap X_2 = \emptyset$, if $X = X_1 \cup X_2$ and $U = U_1 \cup U_2$.

A graph $G(X, U)$ is said to be *a series composition* of two graphs $G_1(X_1, U_1)$ and $G_2(X_2, U_2)$ such that $X_1 \cap X_2 = \emptyset$, if $X = X_1 \cup X_2$ and $U = U_1 \cup U_2 \cup \tilde{U}$, where $\tilde{U}$ is the set of arcs from each vertex of the graph G_1 with zero outdegree to each vertex of the graph G_2 with zero indegree.

A graph G is called *series–parallel* (or *SP–graph*) if either it consists of only one vertex, or it can be obtained from a set of single–vertex graphs by a subsequent application of the operations of series and/or parallel composition.

The monograph [21] by Tanaev et al. provides a systematic exposition of the theory of optimization of so–called priority–generating functions over partially ordered sets. Some related facts can also be found in [16] by Monma and Sidney. As described in [21], any priority–generating function can be minimized over the set of permutations of n elements that are feasible with respect to a series–parallel graph in $O(n \log n)$ time.

In particular, this implies that the $F2|prec_2|C_{\max}$ problem assuming that the reduction graph $\overset{\Rightarrow}{G}$ is series–parallel is solvable in $O(n \log n)$ time. See [14,17] for algorithms of the same running time especially designed for solving this problem.

We now consider the $S2|prec_2, m_j \leq 2|C_{\max}$ and $S2|prec_2,\ pmtn,\ m_j \leq 2|C_{\max}$ problems under precedence constraints of the following structure. The reduction graph $\overset{\Rightarrow}{G}(\mathcal{J}) = \overset{\Rightarrow}{G}(\mathcal{J}, U)$ is a parallel composition of three graphs $\overset{\Rightarrow}{G}(\mathcal{J}_{AB}) = \overset{\Rightarrow}{G}(\mathcal{J}_{AB}, U_{AB})$, $\overset{\Rightarrow}{G}(\mathcal{J}_{BA}) = \overset{\Rightarrow}{G}(\mathcal{J}_{BA}, U_{BA})$ and $\overset{\Rightarrow}{G}(\mathcal{J}_O) = \overset{\Rightarrow}{G}(\mathcal{J}_O, U_O)$, where each is the first two graphs is series–parallel while for the third graph $U_O = \emptyset$.

It can be shown that the algorithm from [19] for solving the $S2|m_j \leq 2|C_{\max}$ and $S2|pmtn,\ m_j \leq 2|C_{\max}$ problems can be modified in order to solve $S2|prec_2$, $m_j \leq 2|C_{\max}$ and $S2|prec_2$, $pmtn$, $m_j \leq 2|C_{\max}$ problems with the graph $\overset{\Rightarrow}{G}(\mathcal{J})$ of the described structure so that the following statement holds.

Theorem 9. *The* $S2|prec_2,\ m_j \leq 2|C_{\max}$ *and* $S2|prec_2, pmtn,\ m_j \leq 2|C_{\max}$ *problems under series–parallel precedence constraints imposed over set* $\mathcal{J}_{AB}$ *and over set* $\mathcal{J}_{BA}$ *are solvable in* $O(n \log n)$ *time.*

4.4 Summary

We now summarize the complexity results for the super shop scheduling problems to minimize the makespan under precedence constraints of the second type.

If the set of job is linearly ordered, then both $J|prec_2|C_{\max}$ and $J|prec_2$, $pmtn|C_{\max}$ problems are solvable in polynomial time; the $S2|prec_2|C_{\max}$ and $S2|prec_2,\ pmtn|C_{\max}$ problems are solvable in $O(n \log n)$ and $O(n^2)$ time, respectively; both $O3|prec_2|C_{\max}$ and $O3|prec_2, pmtn|C_{\max}$ are NP–hard in the ordinary sense. See Section 4.1.

If the reduction graph $\overset{\Rightarrow}{G}$ contains exactly one chain and possibly a number of isolated vertices, then the complexity results for the $S2|prec_2, m_j \leq 2|C_{\max}$ problem and the $S2|prec_2,\ pmtn,\ m_j \leq 2|C_{\max}$ problem are given in Tables 3 and 4, respectively. In these tables, we write "NP" to indicate that the corresponding problem is NP–hard in the strong sense, and "NP?" to indicate that the problem is proved to be NP–hard in the ordinary sense and it is unknown if this is NP–hard in the strong sense. See Section 4.2.

Table 3.

$\overset{\Rightarrow}{\mathcal{J}}$	$\overset{\sim}{\mathcal{J}}$	Complexity	Reference
$\mathcal{J}_O^1$	$\mathcal{J}_O^2$	NP	[2,22]
$\mathcal{J}_{BA} \cup \mathcal{J}_{AB}^1$	$\mathcal{J}_{AB}^2$	NP	Theorem 6
$\mathcal{J}_O$	$\mathcal{J}_{AB}$	NP?	Theorem 8

Table 4.

$\overset{\rightarrow}{\mathcal{J}}$	$\overset{\sim}{\mathcal{J}}$	Complexity	Reference
$\mathcal{J}_{BA} \cup \mathcal{J}_{AB}^1$	$\mathcal{J}_{AB}^2$	NP	Theorem 6
$\mathcal{J}_O$	$\mathcal{J}_{AB}$	NP?	Theorem 7

If the reduction graph $\overset{\Rightarrow}{G}$ contains $p \geq 2$ chains and possibly a number of isolated vertices (see Figure 1), then the $O2|prec_2, pmtn|C_{\max}$ problem is solvable in $O(n^2)$ time (see [20] and [22], Chapter 3, Section 4.7).

The $S2|prec_2, m_j \leq 2|C_{\max}$ and $S2|prec_2, pmtn, m_j \leq 2|C_{\max}$ problems under series–parallel precedence constraints imposed over set $\mathcal{J}_{AB}$ and over set $\mathcal{J}_{BA}$ are solvable in $O(n \log n)$ time. See Section 4.3.

The $F2|prec_2|C_{\max}$ problem under arbitrary precedence constraints of the second type is NP–hard in the strong sense due to Monma [15].

5 Conclusion

In this paper, we have presented the complexity results of the shop scheduling problems under precedence constraints of two types.

In the case of the precedence relation $\rightarrow$ (the first type), the complete classification is obtained.

For the relation $\Rightarrow$ (the second type) the complexity status of several problems listed below is still unknown.

1. Determine the complexity of the $O2|prec_2, pmtn|C_{\max}$ problem if the reduction graph is a tree, a series–parallel graph, an arbitrary circuit–free graph.

2. Does there exist a reduction graph, more general than a series–parallel graph such that the $F2|prec_2|C_{\max}$ problem is polynomially solvable?

3. Are the $X2|prec_2|C_{\max}$ and $X2|prec_2, pmtn|C_{\max}$ problems under single chain constraints with $\overset{\Rightarrow}{\mathcal{J}} = \mathcal{J}_O$ and $\overset{\sim}{\mathcal{J}} = \mathcal{J}_{AB}$ NP–hard in the strong sense?

It is also an interesting research goal to determine the complexity of shop scheduling problems under precedence constraints with the objectives different from makespan.

References

1. Borodich, S.A. and Tuzikov, A.V. (1985). On the complexity of constructing time–optimal schedules for some two–stage service systems with non–fixed routes. *Methods, Algorithms and Programs for the Solution of Extremal Problems*, Institute of Engineering Cybernetics, Minsk, 76–85. (In Russian.)

2. Borodich, S.A. and Strusevich, V.A. (1986). Scheduling a partially ordered set of jobs for a class of servicing systems. *Izvestiya Akademii Nauk BSSR, Seriya Fiziko–matematicheskikh Nauk*, N3, 19–22. (In Russian.)

3. Brucker, P. (1988). An efficient algorithm for the job–shop problem with two jobs. *Computing*, **40**, 353–358.

4. Garey, M.R. and Johnson, D.S. (1979). *Computers and Intractability. A Guide to The Theory of NP–Completeness*, Freeman, San Francisco.

5. Garey, M.R., Johnson, D.S. and Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, **1**, 117–129.

6. Gonzalez, T. and Sahni, S. (1976). Open shop scheduling to minimize finish time. *J. Association for Computer Machinery*, **23**, 665–669.

7. Gonzalez, T. and Sahni, S. (1978). Flowshop and jobshop schedules: complexity and approximation. *Operations Research*, **26**, 36–52.

8. Jackson, J.R. (1956). An extension of Johnson's results on job lot scheduling. *Naval Research Logistics Quarterly*, **3**, 210–203.

9. Johnson, S.M. (1954). Optimal two and three production schedules with set up times included. *Naval Research Logistics Quarterly*, **1**, 61–68.

10. Lawler, E.L., Lenstra, J.K. and Rinnooy Kan, A.H.G. (1981). Minimizing maximum lateness in a two–machine open shop. *Mathematics of Operations Research*, **6**, 153–158.

11. Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G. and Shmoys, D.B. (1993). Sequencing and scheduling: algorithms and complexity. *Handbooks in Operations Research and Management Science*, **4**, *Logistics of Production and Inventory*, Editor: S.C. Graves et al., North–Holland, Amsterdam, 445–522.

12. Lenstra, J.K., Rinnooy Kan, A.H.G. and Brucker, P. (1977). Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, **1**, 343–362.

13. Masuda, T., Ishii, H. and Nishida, T. (1985). The mixed shop scheduling problem. *Discrete Applied Mathematics*, **11**, 175–186.

14. Monma, C.L. (1979). The two-machine maximum flow-time problem with series-parallel precedence constraints: an algorithm and extensions. *Operations Research*, **27**, 792–797.

15. Monma, C.L. (1980). Sequencing to minimize the maximum job cost. *Operations Research*, **28**, 942–951.

16. Monma, C.L. and Sidney, J.B. (1979). Sequencing with series-parallel precedence constraints. *Mathematics of Operations Research*, **4**, 215–234.

17. Sidney, J.B. (1979). The two-machine maximum flow-time problem with series-parallel precedence relation. *Operations Research*, **27**, 782–791.

18. Sotskov, Y.N. (1991). The complexity of shop-scheduling problems with two or three jobs. *European J. Operational Research,* **53**, 326–336.

19. Strusevich, V.A. (1991). *Complexity Aspects of Shop Scheduling Problems*, Ph.D. Thesis, Erasmus University, Rotterdam.

20. Strusevich, V.A. (1991). Two machine super shop scheduling problem. *J. Operational Research Society*, **42**, 479–492.

21. Tanaev, V.S., Gordon, V.S. and Shafransky, Y.M. (1994). *Scheduling Theory, Single-Stage Systems*, Editors: Dordrecht et al., Kluwer Academic Publishers.

22. Tanaev, V.S., Sotskov, Y.N. and Strusevich, V.A. (1994). *Scheduling Theory, Multi-Stage Systems.*, Editors: Dordrecht et al., Kluwer Academic Publishers.

Frequency Assignment for Cellular Radio Networks

D.A. Youngs

Vodafone Limited, Berkshire

Abstract

Assigning frequencies to cells in a cellular radio network has traditionally been carried out either by hand or, more recently, using algorithms derived from the theory of abstract graphs. This second approach is realised by modelling the problem as a simple graph colouring problem, where vertices of the graph represent the cells and edges correspond to potentially interfering cell pairs. The algorithms used were originally developed to colour unweighted graphs. In effect this means that they utilise only "hard" interference data between cells.

In this note we describe a graph colouring algorithm that can be applied quite naturally to edge weighted graphs, and thus also to frequency assignment problems using "soft" interference data. Prior to this we review a number of existing colouring algorithms widely used today. We also discuss issues surrounding the generation of interference data, and how to measure the quality of a given frequency plan.

1 Introduction

The basic *frequency assignment* problem is to assign a single radio frequency to each cell in a network (the initial motivation for this work came from mobile telecommunications networks). Two cells in close proximity are likely to interfere with each other if they are assigned the same frequency. The difficulty then, comes in assigning the frequencies so that this interference is minimised. By modelling the problem as a graph whose vertices represent the cells, and whose edges correspond to potentially interfering cell pairs, the problem can be solved effectively using elementary graph colouring techniques. These techniques date back to at least the 1960's through the work of Welsh and Powell [1], Szekeres and Wilf [2], and Matula et al. [3]. An early indication of their applicability to frequency assignment was given by Hale [4].

The basic problem is prone to a number of further complications. First, it may be necessary to assign more than one frequency to a cell. This merely leads to an increase in the size of the problem instance, and can also be solved effectively using the same graph colouring techniques. Secondly, simply assigning different frequencies to two cells may not be sufficient to avoid interference, in which case further *frequency separation* between the cells is necessary. This results in what is still a fairly straightforward problem, and little further work is required for the usual graph colouring algorithms to apply. Finally, most colouring algorithms attempt to minimise the the number of colours used, whereas in frequency assignment, the number of frequencies is usually specified in advance, and the objective is to make the best possible use of them. Whilst the colouring algorithms can usually be adapted to cope with this situation, the resulting algorithms are rarely as elegant or as effective.

The problem as it now stands is interesting enough, and most present-day frequency assignment is carried out by applying standard graph colouring algorithms to this problem (although promising alternative approaches, such as Kunz's neural networks [5], have been proposed). Nevertheless, there is one further aspect that should be taken into account, one that sets it apart from elementary graph colouring. The problems encountered so far are characterised by the fact that the *exclusions* (or edges) are "hard"—either they exist or they do not. In practice we have available far more information than this, usually in the form of an *interference* value between each pair of cells that lies in a specified continuous range. Thus, we can remodel our problem by considering an *edge weighted graph* whose weights correspond to interference values. To solve this problem requires a significant digression from the usual graph colouring techniques.

In the next section we review a number of well known sequential colouring algorithms and briefly discuss their application to frequency assignment problems. Following this, in Section 3 we present a non-sequential colouring algorithm that can be applied very easily to edge weighted graphs with a specified number of colours, and thus also to frequency assignment problems with soft (or continuous) interference data.

Once a solution (i.e. a frequency plan) has been found, the next question is: how good is it? How to measure the "quality" of a particular solution is a difficult and possibly subjective matter. The measure we introduce and use is a crude one based on the residual interference the network experiences after the frequency assignment. This is considered further in Section 4.

A major factor influencing the quality of solutions produced by our, or any other, frequency assignment algorithm, is the information content of the input data. The important data in our case are the weights between pairs of cells. These weights should correspond to the importance of avoiding interference. The generation of interference data is discussed in Section 5.

2 Sequential algorithms

Perhaps the most obvious way to colour a graph (or assign frequencies) is to use a sequential or "greedy" algorithm. In this case we simply consider the vertices (cells) one at a time, successively assigning allowable colours (frequencies) as we go, until we have either coloured all vertices or run out of colours.

The main point where frequency assignment differs from the traditional colouring problem, is that a frequency plan must use only a specified number of frequencies. Therefore, the second terminating condition is unacceptable for frequency assignment. To combat this a "fix" is normally used whereby disallowed colours are assigned to the remaining uncoloured vertices.

Perhaps the most important factor affecting the quality of solutions generated by this method is how the next vertex is chosen. In addition, an initial ordering of the vertices for resolving uncertainties from the "choice of next vertex" is important, as is the method by which we choose a colour for each vertex. We may therefore generate a whole series of algorithms, each with the same underlying modular structure based on the three components

- initial ordering,
- choice of next vertex,
- assignment of colour.

The most popular algorithms are formed through some combination of the following components

Initial ordering

- random order,
- largest vertex degree first (excluding vertices already ordered),
- smallest vertex degree last (excluding vertices already ordered),
- largest vertex degree first (including vertices already ordered).

Choice of next vertex

- next vertex in the initial ordering (excluding vertices already coloured),
- vertex of smallest degree (excluding vertices already coloured),
- vertex with fewest colours available to it,
- vertex whose assignment results in the minimum degree of the uncoloured vertices being maximised,
- vertex whose assignment results in the minimum number of colours available to uncoloured vertices being maximised.

Assignment of colour

- smallest available colour (assuming we initially order the colours),
- least used available colour,
- most used available colour.

Frequency separation can easily be built into the algorithm at the "assignment of colour" stage.

It is possible to generalise the algorithms to cope with weighted edges (or soft interference data). This can be achieved at the "choice of next vertex" stage. To be more precise, where a summation of edges is involved (for example calculation of the degree of a vertex), this could be replaced by a summation of edge weights.

One of the major problems that can occur with weighted edge versions of the algorithms, is that the terminating conditions may become ineffective. This is because, using soft interference data usually leads to considerably more edges being involved (hard data corresponds to the inclusion of only those edges whose weights exceed a given threshold), which in turn means that the graph is harder to colour and a large number of vertices remain uncoloured when the algorithm terminates. To remedy this, a large number of vertices may have to be assigned previously disallowed colours. Careful consideration then needs to be given regarding how these colours are assigned.

The family of sequential algorithms described above are generally acceptable for use in practical situations. Nevertheless, very little formal analysis has appeared in the published literature regarding the quality of the solutions they produce.

3 An alternative algorithm

In this section we describe an algorithm having an entirely different character to those of the previous section. The fundamental difference lies in the fact that this algorithm is not sequential in nature. Since our primary application is to frequency assignment, we consider it appropriate to describe the algorithm in terms of frequency assignment rather than graph colouring.

3.1 Input data

The algorithm requires the following input data

Cell set
This is simply the set C of identities of cells to be assigned frequencies.

Multiple frequencies
A positive integer $m(c)$ is associated with each cell c. This corresponds to the number of frequencies to be assigned at that cell. In effect, when the algorithm runs, each cell is divided into $m(c)$ "sub-cells", one for each desired frequency.

Frequency number
This value k corresponds to the number of different frequencies available. The set of frequencies themselves are usually defined as $F = \{1, 2, ..., k\}$.

Interference
A weight or potential interference $i(c,d)$ is associated with each pair of (sub-) cells c and d. This corresponds to the perceived potential interference between the cells, or more precisely to the importance of avoiding interference between the two cells. The weights may take values in any given range, usually some closed interval of real numbers or discrete 0-1 values. Note that $i(c,d) = 0$ whenever c and d do not interfere.

Frequency separation
A positive integer $ds(c,d)$ is associated with each pair of (sub-) cells c and d. This value corresponds to the desired frequency separation between the cells. Note that $ds(c,d) = 0$ whenever c and d do not interfere, and $ds(c,d) = 1$ whenever it is sufficient for c and d to be assigned merely distinct frequencies.

Initial frequency plan
For each (sub-) cell c we are given an initial frequency assignment $if(c)$ for c. For simplicity we usually assume $if(c) = 1$ for all c. Nevertheless, if can be set to other values such as a randomly generated frequency assignment, or a frequency assignment generated by some other means (for example by another frequency assignment algorithm).

Predefined frequencies
A subset of the cells with preassigned fixed frequencies may also be prescribed.

3.2 Notation

Some further notation is required before we describe the algorithm itself. At any point during the algorithm, a frequency plan $f : C \rightarrow F$ will be in place. The actual frequency separation $as_f(c,d)$ between two cells c and d with respect to f is given by

$$as_f(c,d) = |f(c) - f(d)|.$$

We may then define the deficiency $D_f(c)$ of a cell c with respect to f to be

$$D_f(c) = \sum_{d:d \neq c} \max\{0, ds(c,d) - as_f(c,d)\} i(c,d).$$

This can be viewed simply as the sum, taken over all cells d distinct from c, of the interference between c and d, weighted according to how well they achieve their desired frequency separation.

Finally, suppose f is a frequency plan and c is a cell. We define a set $F(f,c)$ which consists of frequency plans that are identical to f except possibly in their assignment at the cell c. Notice that $|F(f,c)| = k$ (where k is the number of frequencies available). The net deficiency $E_f(c)$ of c with respect to f is then defined as

$$E_f(c) = \max_{g \in F(f,c)}\{D_f(c) - D_g(c)\}.$$

This can be viewed as the amount by which the deficiency of c can be reduced by assigning a different frequency. Clearly $E_f(c)$ is always non-negative, and is zero whenever the current frequency assigned to c has the smallest deficiency. If g is a frequency plan for which $E_f(c)$ achieves its maximum, then g is termed a preferred plan for c (relative to f), and $g(c)$ a preferred frequency for c.

3.3 The algorithm

The algorithm assigns the desired number of frequencies to each cell. It does this in a way that minimises, as far as possible, the residual interference experienced in the network. Precisely how this residual interference is calculated is the subject of Section 4. The remainder of this section is concerned with how the solution is generated.

Initially we construct a larger network of cells from our initial set, so that each cell requires precisely one frequency to be assigned. Frequency "1" is then assigned to all cells to give our initial frequency plan. Thereafter we continually change the frequencies assigned to cells until a stable plan is reached. At each stage of the algorithm we have a current frequency plan. Upon termination of the algorithm we take the stable current plan as our final solution.

Formally, the steps of the algorithm are as follows

Step 1 (expansion): Divide each cell c into $m(c)$ sub-cells, one for each of the desired frequencies in that cell, each one incident with the same neighbouring cells as the original, and also with the other newly formed sub-cells.

Step 2 (initialisation): Set the frequency plan *current* = *if* (here we assume $if(c) = 1$ for all c). Set the number of available frequencies *freq* = *2*.

Step 3 (reassignment): While cells exist with net deficiency $E_{current} > 0$, choose a cell c such that $E_{current}(c)$ is a maximum (taken over all cs). Set the frequency plan current equal to a preferred plan for c.

Step 4 (increment): If the total number of available frequencies $freq < k$, then set $freq = freq + 1$ and return to Step 3. Otherwise STOP.

At this point a few explanatory remarks are in order. The new edges introduced between pairs of sub-cells in Step 1 must be assigned suitable interference values (also determined by the factors discussed in Section 5). All frequency plans considered in Step 3 are allowed to range over frequencies 1,2,..., *freq* only. A simple derivation of the algorithm is obtained by setting $freq = k$ at Step 1 and ignoring Step 4. This is useful if *if* is not the trivial all "1"s assignment. If a cell has a prescribed fixed frequency, then we always set its net deficiency equal to zero. This prevents its assigned frequency from being changed.

It is left as an easy exercise for the reader to show that the algorithm always terminates.

4 Quality measure

Suppose that for each cell c we are given the frequency assignment $f(c)$ of c, and for each pair c and d of cells, we are also given the interference $i(c,d)$, the desired frequency separation $ds(c,d)$, and the actual frequency separation $as_f(c,d)$. We assume here that in the case of multiple frequencies, the set of cells has been split as described in Step 1 of the algorithm (i.e. we treat them as separate cells and have precisely one frequency assigned in each cell).

We define the residual interference $I = I(f, i, ds)$ of the frequency plan f to be

$$\begin{aligned} I &= \textstyle\sum_{c,d:c\neq d} \max\{0, ds(c,d) - as_f(c,d)\} i(c,d) \\ &= \tfrac{1}{2} \textstyle\sum_c D_f(c) \end{aligned}$$

This can be viewed simply as the sum, taken over all pairs of cells, of the interference between them, weighted according to how well they achieve their desired frequency separation. Notice that the summation is over all unordered pairs of cells.

It is possible that a better measure than I can be found to assess the quality of a given frequency assignment. Nevertheless, the function I does possess most of the essential qualities required of such a measure, so it is likely that a good measure will not differ to any great extent from I. Two of the basic properties of I are as follows

- I achieves its maximum value with the worst possible frequency assignment, i.e. when all cells are assigned identical frequencies.
- I achieves its minimum value ($= 0$) if and only if the frequency assignment is perfect, i.e. when no two cells interfere with each other.

The function I also possesses other desirable properties, such as its apparent "monotonicity" (i.e. for intermediate values of I, a lower value seems to indicate a "better" plan), and its "dependence" on all of the input data.

Of course, even if I were deemed to be an acceptable measure, there are still aspects of it that require further study. In particular, there is the question of whether the interference is a linear function of the frequency separation achieved.

A very important assumption made thus far is that all information (interference data and frequency separation) concerns merely "pairs of cells". The problem becomes far more complicated if "triples of cells" and so on are to be considered. Indeed it then becomes impossible to even model the problem in a graph theoretic setting (it requires a more general "hypergraph" or "set system" theory), let alone trying to solve it using techniques from the theory.

Finally, we reiterate that there is a very important distinction to be made between the applicability of the function I, and how interference values are assigned to pairs of cells. For example, factors such as "cell size" and "cell usage" can be taken into account through the interference value assigned between cell pairs, and not in the way that quality is measured. Further discussion on this matter may be found in Section 5.

5 Interference data

As described previously, a weight or interference value is assigned between each pair of cells in the network. These values should reflect the importance of assigning distinct (or further separated) frequencies, and not just the level of interference experienced when assigning the same frequency. Of course the amount of interference caused is an important factor, but there are other equally important factors that up to now have been largely ignored in practice.

Other factors that should have a bearing on the weight are cell loading and geographical location. Obviously if a cell is heavily loaded with users then it is more important than a cell that is hardly used at all. Similarly, in certain geographical locations such as Inner London it is desirable to reduce interference to an absolute minimum, not only because the cells situated there are heavily used, but also because there may be competition from other networks, or perhaps the users may expect a superior service. Another factor is that of "cell size": two large cells interfering with one another will result in interference over a larger geographical area than two small cells interfering.

How to calculate weights between cells is an area where considerable further work is required. At present the interference value for a pair of cells is often derived simply from data that estimates how much (which is itself a fairly loosely defined term) interference is experienced between the cells when they are assigned identical frequencies. Other factors such as those mentioned above should be incorporated into the weight calculations.

6 Conclusions

Frequency assignment problems in cellular radio environments are commonly solved using techniques borrowed from the theory of abstract graphs, more specifically, graph colouring. These techniques tend to aim at minimising the number of colours used to colour a graph, whereas frequency assignment can be viewed as making maximum use of a predefined set of colours. In addition, the data available for use in practical frequency assignment is generally "soft" in nature, representing the level of potential interference between cells. In spite of these two distinguishing features, graph colouring algorithms have up to now, been used only in their traditional form, minimising the number of frequencies (colours) and handling hard input data.

In this note we have attempted to show how graph colouring techniques can be used to tackle frequency assignment problems in thier natural setting with predefined sets of available colours and soft input data. As well as giving a brief indication of how traditional sequential algorithms can be generalised to cope with soft input data, we have described a new non-sequential algorithm eminently suitable for use with soft input data.

Measuring the quality of solutions offered by traditional graph colouring techniques is a trivial exercise, it amounts to counting the number of colours used. For frequency assignment the problem becomes far more difficult—we are given a set of frequencies, and must make "best" use of them. in this note we have proposed a simple quality measure for frequency assignment. Nevertheless, until comparable algorithms (i.e. based on fixed sets of frequencies and soft data) are forthcoming, the relative merits of the algorithm presented here cannot be assessed in any meaningful way.

References

1. Welsh, D.J.A. and Powell, M.B. (1967). An upper bound for the chromatic number of a graph and its application to timetabling problems. *Computer J.*, **10**, 85–86.

2. Szekeres, G. and Wilf, H.S. (1968). An inequality for the chromatic number of a graph. *J. of Combinatorial Theo.*, **4**, 1–3.

3. Matula, D.W., Marble, G. and Isaacson, J.D. (1972). Graph coloring algorithms. *Graph Theo. and Computing*, Academic Press, New York, 109–122.

4. Hale, W.K. (1980). Frequency assignment: theory and applications. *Proc. IEEE*, **68**, 1497–1513.

5. Kunz, D. (1990). Practical channel assignment using neural networks. *Proc. IEEE Conf. Vehicular Technology*, 652–655.

9 780198 511922