Combinatorial Design of Fault-Tolerant Communication Structures, with Applications to Non-Blocking Switches

Thesis by

David Lawrence Schweizer

In Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy

Department of Computer Science
California Institute of Technology
Pasadena, California

1991

(submitted May 24, 1991)

Acknowledgments

This thesis marks not only the culmination of my career at Caltech, but also the conclusion of my formal schooling. It is fitting, therefore, that I thank not only the people who have helped me reach the end of this last project, but also those who helped me get to where I could start it.

My Caltech experience has been shaped most profoundly by my advisor, Yaser Abu-Mostafa. He provided me with a near perfect environment for research; he encouraged me when my work was not going well, and shared my delight when it went better; and he always displayed great confidence in my abilities. The rest of the research group—Jack Lutz, Ruth Erlanson, Amir Atiya, Andy Fyfe, and Allen Knutson—listened patiently and helpfully to numerous reports of work in progress. Dan Ashlock and I had many useful conversations about the material in this thesis when I first started to work on it.

Over the years I have had the good fortune to encounter many very good teachers, and a few truly excellent ones. Without their dedication to their profession, I would not have come this far; without the tools they gave me, I could not have created this thesis.

Finally, I wish to thank my parents for their love and support every step of the way.

Abstract

This thesis is an investigation into structures and strategies for fault-tolerant communication. We assume the existence of some set of nodes—people, telephones, processors—with a need to pass messages—telephone calls, signals on a wire, data packets—amongst themselves.

In Part I, our goal is to create a structure, that is, a pattern of interconnection, in which a designated source node can broadcast a message to (and through) a group of recipient nodes. We seek a structure in which every node has tightly limited fan-out, but which is nonetheless able to function reliably even when challenged with significant numbers of node failures. The structures are described only in terms of their connectivity, and we therefore use the language of graph theory.

Part II is based on the observation that certain transformations of the graphs in Part I produce graphs that look like previously studied structures called non-blocking switches. We show that these transformations, when applied to other graphs, yield new, easier approaches to, and proofs of, some known theorems.

Part III is an independent body of work describing some investigations into possible extensions of the theory of Kolmogorov–Chaitin complexity into the foundations of pattern recognition. We prove the existence of an information theoretic metric on strings in which the distance between two strings is a measure of the amount of specification required for a universal computer to interconvert the strings. We also prove two topological theorems about this metric.

Contents

Part I

Communication Multitrees

Chapter 1

Introduction to the Communication Multitree Problem

1. An Informal Description

Let us suppose that we are inclined to throw extremely large parties on very short notice. By extremely large, we mean parties to which we invite thousands, or even tens of thousands, of friends. By short notice, we mean that the inspiration might strike only hours before we wish the revelries to commence.

This clearly presents some difficulties. If we begin to telephone our friends the night before the party, we will still be making calls at dawn—doubtless having annoyed a number of people by waking them up in the middle of the night—and will be nowhere near finished. There is a well-known approach to this problem: since we know well in advance that we are prone to such urges, we set up a telephone tree. Each friend is sent a postcard that contains a list of names and phone numbers. Each of our friends is instructed that if he or she should receive a call announcing a party, he or she should call all the people on his or her list. So we call six friends, and they call thirty-six, and those thirty-six call two hundred and sixteen, and exponential growth quickly spreads the word.

This scheme can, however, fail quite spectacularly. If some people fail to make their calls, huge groups of our friends will not hear about the party. For example,

if three out of the top six don't make their calls, fully half of our friends won't hear about the party.

The repair we would like to make is to have each person receive several calls, from different sources. The question we will be addressing in this part of this thesis is how to select the names for each postcard to make good use of this fix.

2. The Formal, Mathematical Formulation

The problem we are addressing is not as frivolous as it may seem. We could equally well describe it in the language of multiprocessors, talking about computing nodes passing messages; or in a military metaphor, describing a commander trying to communicate with his or her lower-echelon officers.

We wish to construct a tree-like structure, which we will call a *communication multitree.* The structure must have a distinct source, a notion of direction away from the source, and a width that increases exponentially with distance from the source. Further, we will require that there be two parameters, r and k, such that each person (other than the source) on the multitree makes no more than r calls, and receives no more than k calls.

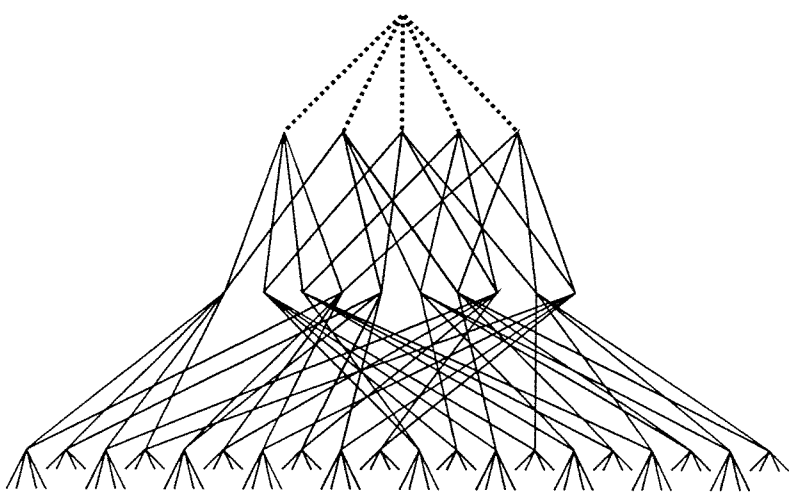

Figure 1.1

In this figure, we have drawn the connections from the source differently from the other connections in the multitree. The source vertex is fundamentally different from the rest of the tree—since it initiates all communication, for example, it is of no use to consider scenarios in which it fails—and therefore will be omitted from most of the analysis that follows. Although we will specify which vertices are to be connected to the source, our focus will be entirely on the rest of the tree. The failure mode that we shall be examining is a catastrophic one. We want the multitree to have the property that even if at some given level there is only one person who makes his or her calls, eventually (if everyone else on the multitree cooperates) there will be a level where everyone gets a call. This is by no means the only interesting model of failure, and it is not even a particularly realistic one, but it puts a strong constraint on the connectivity of the structure. We will argue that a multitree with this recovery property is likely to perform well in the face of other challenges.

Note that if we examine structures that extend infinitely far, then an ordinary tree loses a fraction of its nodes if one fails, while our requirement implies that only a finite number of nodes may be lost.

3. Related Previous Work

The general problem of disseminating information reliably from a source to a set of destinations, often called fault-tolerant broadcasting, has been investigated previously in various models. All of them abstract the problem to the language of graph theory, but they diverge rapidly from there.

Some of the previous emphasis [4, 13] has been on edge failure models and has been concerned with the time required to get a message through to all possible recipients in the presense of a small number of faults. We will be examining vertex failure models, and will be investigating the possibility of getting a message through to a large subset of the possible recipients in the presense of near-catastrophic failures.

Some other investigators [22, 16] have been concerned with minimal size graphs that have very strong fault-tolerance in the presense of even smaller numbers of faults. We will not examine ways to minimize the size of the graphs we construct, as this is assumed in our model not to be a significant concern.

Chapter 2
Outline of the Construction of the Solution

1. A Basic Overview

The multitrees that we will construct will be built from *identical* blocks, level by level. A block consists of a group of callers, a group of callees, and an arrangement of calls from callers to callees. In any given block, the set of callers and the set of callees are disjoint. A block might have, for example, seven callers, twenty-one callees, and a total of forty-two calls. The callees in the blocks at one level become the callers at the next level. We call these blocks *connectors* to emphasize that they describe the connections from one level to the next.

To build such multitrees, we must answer two questions:

1. How do we choose a good block?
2. How do we regroup the callees when they become callers?

2. Digression: Graph Theory

Basic notation and definitions

A *graph* is a set of vertices, V, and a set of edges, E, where $E \subseteq V \times V$. If (v_i, v_j) is an edge of a graph, we say that v_i is *adjacent* to v_j, and that the edge (v_i, v_j)

is *incident* on v_i (and also on v_j). The *degree* of a vertex is the number of edges incident on that vertex.

In an ordinary graph, we take the edge (v_1, v_2) to be the same as the edge (v_2, v_1), but in a *directed graph* (or *digraph*) these are considered distinct. The *in-degree* of vertex v_i is the number of edges of the form (v_k, v_i), and the *out-degree* is the number of edges of the form (v_i, v_j).

A *hypergraph* also consists of a set of vertices, V, and a set of "edges," E, but the edges are called *hyperedges*, and any particular hyperedge is an element of the i-fold Cartesian product V^i of the vertex set. An *n-hypergraph* is a hypergraph where all of the hyperedges are elements of V^n, for some fixed n. In particular, a graph is a 2-hypergraph.

A *bipartite graph* is a graph whose vertex set can be partitioned into two disjoint subsets, called sides, such that all the edges of the graph have one vertex in one subset and the other vertex in the other. The division is called a *bipartition*, and we will often denote the sides by A and B.

The *adjacency matrix* of a graph is a matrix of zeros and ones with a one at the intersection of the i^{th} row and j^{th} column if and only if (v_i, v_j) is an edge of the graph. The adjacency matrix of a graph is symmetric, that of a digraph may or may not be. Ones along the diagonal indicate edges that connect a vertex to itself: these are sometimes ruled out *a priori*, and none of the graphs we will consider has any such edges. A bipartite graph can be described by a *reduced* adjacency matrix in which we assign the rows to one side of the bipartition, and the columns to the other.

The *incidence matrix* of a graph has rows that correspond to the edges of the graph, and columns that correspond to the vertices; a row with ones in the i^{th} and j^{th} columns corresponds to the edge (v_i, v_j). The notion of incidence matrix carries over unchanged to hypergraphs, but while the incidence matrix of a graph must have row sum 2, the incidence matrix of a hypergraph is not so restricted.

We can summarize all of this: first, the definition of a graph

$$G = (V, E);$$

$$V = \{v_i\}, \qquad E \subseteq V \times V, \qquad e_i = (v_j, v_k);$$

and second, the definitions of adjacency and incidence matrices

$$A = [a_{ij}], \quad a_{ij} = 1 \iff (v_i, v_j) \in E; \qquad \text{(adjacency matrix)}$$

$$B = [b_{ij}], \quad \exists i : b_{ij} = 1 \text{ and } b_{ik} = 1 \iff (v_j, v_k) \in E. \qquad \text{(incidence matrix)}$$

A *Hamilton cycle* is a sequence of edges from the graph that passes through each vertex exactly once, ending where it began. Formally, in a graph with k vertices,

$$H = ((v_{i_1}, v_{i_2}), (v_{i_2}, v_{i_3}), \ldots, (v_{i_k}, v_{i_1})),$$

where $(i_1, \ldots, i_k)$ is a permutation of $0, \ldots, k-1$. We will use the permutation, which lists the vertices of the graph in the order that they are visited, as an abbreviated notation for Hamilton cycles. A *Hamilton cycle decomposition* is a collection of disjoint Hamilton cycles whose union is the entire graph.

We will use *tree* to refer to graphs usually called rooted directed trees. For our purposes, a tree is a digraph whose vertices can be partitioned into *levels*, with a single vertex (called the *root*) with in-degree 0 at level 0, with every other vertex having in-degree 1, and with all edges being directed edges from the vertices at level i to the vertices at level $i + 1$. Such a tree may be finite or infinite in extent.

A *multitree* is a tree with the in-degree 1 restriction dropped. We will describe multitrees as being made up of levels of vertices joined by *layers* of edges.

3. A Pictorial Overview

A connector is a bipartite graph—directed, to be precise, but the direction (from caller to callee) is obvious and the formalism is not used in the proofs. A communication multitree is made of layers of groups of connectors. We can imagine constructing a communication multitree in stages. First, we decide how many vertices will appear at each level. Then we lay down the root. Then we lay down the vertices at level 1, and put in an edge from the root to each of them. Then we lay down the vertices at level 2, and put in edges according to some scheme. Then we lay down the next level of vertices, and so on.

We have already said that the scheme that governs the layers of edges is simple: in any given communication multitree, we use just one selected connector as many times as necessary in each layer. Let us suppose we have a connector, for example, one with four callers, eight callees, and every possible edge, thus:

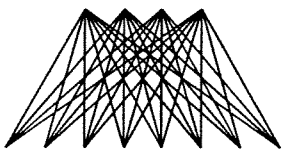

Figure 2.1

We can build a communication multitree in a very direct fashion, as follows:

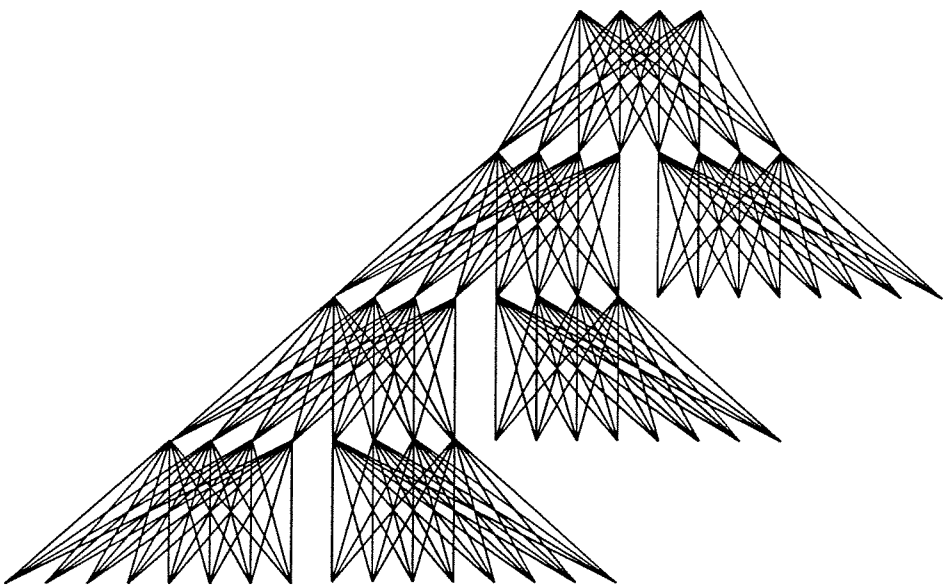

Figure 2.2

But this construction has an obvious drawback: although such a multitree will have some local fault-tolerance, it will not recover globally. Clearly, we do not, in general, want people who are callees together to be callers together. A convenient way to think of the process of selection of a group of callees who will become callers in the same connector to the next level is to imagine permuting the vertices at each

level as we go along, always preserving connectivity. That mixing is a major part of our construction.

The big picture

Suppose that every vertex in our communication multitree is identified by an ordered pair (i, j), where i is the level it is in, and j is its index in that level. Assume we have a connector with v callers and b callees, and that b in an integer multiple of v. Also assume that we have some sequence of permutations, σ_i, where σ_n is a permutation on $v\left(\frac{b}{v}\right)^{n-1}$ objects. We can describe the construction recursively. First, connect the root to every vertex in level 1. Second, assume that the multitree has been constructed up to level i. Insert a connector with callers

$$(i, 0), (i, 1), \ldots, (i, v-1)$$

and callees

$$(i+1, 0), (i+1, 1), \ldots, (i+1, b-1),$$

and another connecting the next v vertices in level i to the next b vertices in level $i+1$, and so on. Now permute the vertices at level $i+1$, moving $(i+1, j)$ to $(i+1, \sigma_{i+1}(j))$. The following figure shows a communication multitree with parameters

$$v = 5, b = 10, r = 4, k = 2,$$

and is isomorphic to the one shown in Figure 1.1.

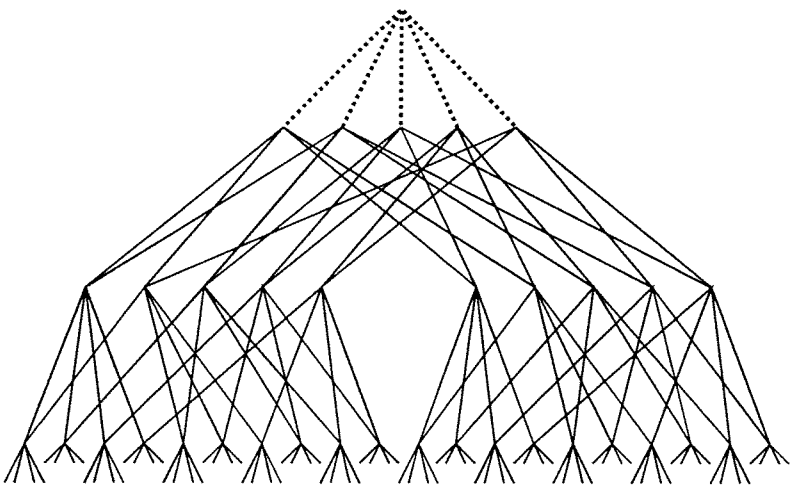

Figure 2.3

The way we construct the permutations is as follows. Each permutation σ_i is actually the composition of four other permutations, three of them arising from the specific connector used and one solely related to the parameter $\frac{b}{v}$. We will call these four permutations, in the order they are applied, α, π, α^{-1}, and β. The situation is illustrated in the next figure, where two vertices shown at opposite ends of a sequence of dashed lines are identified; that is, they are actually a single vertex before and after permutation.

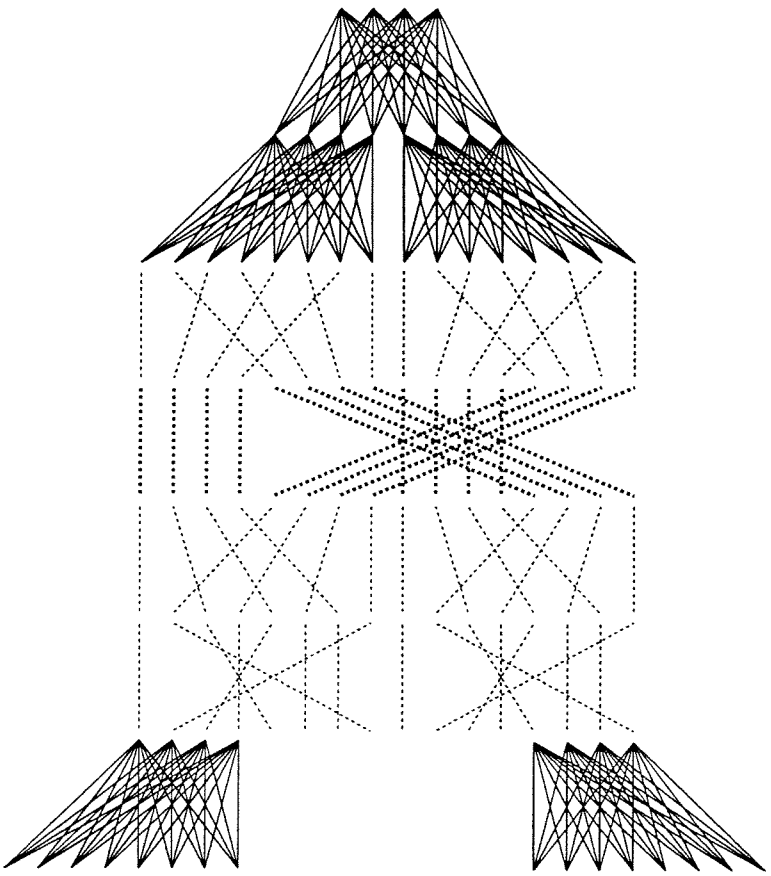

Figure 2.4

Thus, the natural division of our exposition is not, as one might guess from the questions asked in §I.2.1, into connectors and permutations, but rather into connectors with the α and β permutations, and then the permutations π. The next two chapters will develop these subjects individually, and the subsequent chapter will combine them.

Chapter 3

Blocks, and the Permutations α & β

1. Basic Ideas

Our basic building block (in any particular instance) is a bipartite graph with $|B| = \frac{r}{k}|A| = \frac{b}{v}|A|$, where, in view of the exponential growth requirement of §I.1.2, r is a multiple of k. We will denote such a connector by $C_{|A|,|B|}$. As they are, in general, not fully connected, the notation is ambiguous. However, as the connectivity is a major thrust of this work, the basic structure will be clear in any given instance, leaving the values of the parameters as the quantities of interest.

As we have said, the layers of a multitree are sets of disjoint connectors. We will use "ancestors" to refer to the vertices when they are viewed as inputs, and "descendants" when we speak of their dual role as outputs. In the connector pictured in Figure 3.1, we have marked the seven descendants of a set of two inputs, or, from the other perspective, two of the ancestors of a set of seven outputs.

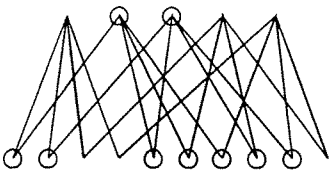

Figure 3.1

One of our basic requirements is that the direct descendants of a given (proper) subset of ancestors appear together as inputs in a single block as infrequently as possible. In fact, we want the generations of their descendants to interact as little as possible. (There are—at least—first and second order interactions, but the first-order effects dominate most of the time, and are always easier to investigate. So our emphasis will be on these, but we will point out where the second-order effects enter into the picture.)

2. Digression: Design Theory

Balanced incomplete block designs have been widely studied. [34. 3. 36] They are usually called (v, b, r, k, λ) designs. A design is composed of v symbols grouped into b blocks, with each block a k-subset of the set of symbols, every symbol appearing in exactly r blocks, and every pair of symbols occurring together in exactly λ different blocks. These conditions imply two relations between the parameters of the design:

$$vr = bk,$$
$$\lambda(v-1) = r(k-1).$$

The *incidence matrix* of a (v, b, r, k, λ) design is a matrix of zeros and ones with v columns, b rows, and a one at the intersection of column i and row j if symbol v_i occurs in block b_j. All the rows of the incidence matrix of a design sum to k, and all the columns sum to r.

The complete graph on n vertices, K_n can be viewed as a

$$(n, \binom{n}{2}, n-1, 2, 1)$$

design. This design is an example of the class of *irreducible* designs [34]; others[3] call them complete designs, but we will reserve that term for another use.

We will also be interested in matrices of zeros and ones that are not incidence matrices of designs, but which may be combined to yield a design. In particular, we will be considering square matrices of zeros and ones with all row and column sums equal. We will call such a matrix a *balanced pattern.* A *near-balanced pattern* is one with equal row sums and column sums that deviate from that value by at most ± 1.

A *connected balanced pattern* is a balanced pattern that has the property that any division of the columns into two nonempty parts will include at least one row which has ones in both parts of the division.

If the rows of the incidence matrix of a design can be partitioned into groups such that each group is a balanced pattern, then we say we have a *balanced pattern decomposition* of the design.

For example, the three matrices:

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

are connected balanced patterns, and together are a balanced pattern decomposition of a $(7, 21, 6, 2, 1)$ design—also known as K_7.

3. Examples Derived from Complete Graphs ($k = 2$)

Suppose we take the incidence matrix of K_p, the complete graph on some odd prime number of vertices. This graph has p vertices and $\binom{p}{2}$ edges. Thus, the matrix has p columns and $\binom{p}{2}$ rows. We now view this matrix as the reduced adjacency matrix of a bipartite graph. This yields a mapping

$$K_p \longrightarrow C_{p,\binom{p}{2}}.$$

Observe that any partition of the edge set of K_p induces a partition of the B side of the bipartition of $C_{p,\binom{p}{2}}$.

If we use a Hamilton cycle decomposition of K_p, we will have $\frac{r}{k}$ blocks (as required). Further, the edges are very evenly distributed through the partition: two of the edges incident upon each vertex go into each class of the partition.

Lemma 3.1 A Hamilton cycle decomposition of K_p is equivalent to a connected balanced pattern decomposition of the corresponding design.

Proof Any Hamilton cycle is clearly a balanced pattern: every edge connects two vertices (thus implying row sum 2) and every vertex occurs in exactly two edges (guaranteeing column sum 2). Since the Hamilton cycle is a connected graph, the balanced pattern must be connected. Thus each cycle induces a balanced pattern. Since the cycles are edge disjoint, the patterns must also be disjoint, and thus form a decomposition. ∎

For example, the three matrices exhibited in §I.3.2 correspond to the following Hamilton cycle decomposition of K_7 :

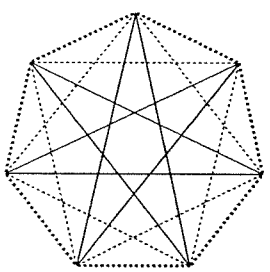

Figure 3.2

The first problem we face is how to decompose K_p into Hamilton cycles. We begin with a theorem that unifies some of the possible descriptions.

Theorem 3.2 Let $1 \leq k \leq \frac{p-1}{2}$, and take all arithmetic to be mod$p$. Then the following are equivalent descriptions of a single Hamilton cycle in K_p :

1. The cycle described by the permutation of the vertices:

$$-(\frac{p-1}{2})k,\ -(\frac{p-1}{2}-1)k,\ \ldots,\ -2k, -k, 0,\ k,\ 2k,\ \ldots,\ (\frac{p-1}{2}-1)k,\ (\frac{p-1}{2})k.$$

2. All the edges (v_i, v_j), where $|i-j| = k$.
3. The subgraph whose incidence matrix is the circulant formed by circulating

$$\underbrace{1\overbrace{0\ldots0}^{k-1}100\ldots0}_{p}.$$

Proof In addition to proving that the three descriptions are equivalent, we need to prove that they describe a Hamilton cycle. First, we prove that (1.) describes a Hamilton cycle. Observe that we are working in the group $(\mathbb{Z}_p, +)$, and therefore k is a generator. This means that the sequence

$$0,\ k,\ 2k,\ 3k,\ \ldots,\ k(\frac{p-1}{2}-1),\ k(\frac{p-1}{2}),\ \ldots,(p-2)k,\ (p-1)k$$

contains every element of the group. But if we rewrite the $(p-n)k$ terms as $-nk$ terms (i.e., write $-2k$ for $(p-2)k$, etc.), this clearly becomes a permuted version of (1.). Thus (1.) is indeed a Hamilton cycle.

We turn now to proving the equivalence of the three descriptions. First, to show the equivalence of the first and second, observe that since the difference between successive terms in the sequence (1.) is k, by definition that sequence describes edges (v_i, v_j) with $|i-j| = k$. For any edge (v_i, v_j) with $|i-j| = k$, the vertex i must occur somewhere in the sequence (1.), and clearly j must either immediately precede or immediately follow it. To show that the second description is equivalent to the third, notice that since p is an odd prime, the numbers of zeros on each side of the second 1 in the circulant (3.) are different. Thus, the circulant has p distinct rows. Since our definition of an incidence matrix requires that the columns of the circulant correspond to $v_0, v_1, \ldots, v_{p-1}$, each row defines an edge (v_i, v_j) with $|i-j| = k$. ∎

Corollary 3.3 Letting k range over all possible values $1, 2, \ldots, \frac{p-1}{2}$, yields a decomposition of the graph K_p into Hamilton cycles.

Proof We need to prove two things: that distinct values of k yield distinct cycles, and that the union of these cycles is the entire graph. For the first part, assume that there are two different values, k_1 and k_2 that yield the same cycle. Pick an edge (v_i, v_j) in that cycle. Since we cannot have $i - j = k_1$ and $i - j = k_2$ simultaneously, we must have $i - j = k_1$ and $j - i = k_2$. But this says that $-k_2 = p - k_2 = k_1$. However, for any value of k, $1 \leq k \leq \frac{p-1}{2}$, $p - k$ is outside the range. The second part is immediate: we have $\frac{p-1}{2}$ disjoint cycles, each containing p edges. They must exhaust the $\binom{p}{2}$ edges in the graph. ∎

We call the decomposition thus generated the *principal* Hamilton cycle decomposition of K_p. The reason for the restriction to prime values is now clear: in the composite case, the additive group has subgroups, and these would lead to disconnected small cycles for some values of k.

Since a decomposition of K_p into Hamilton cycles is by definition a partition of the edge set of K_p, we immediately get an induced partition of the B side of the bipartition of $C_{p,\binom{p}{2}}$. We can now define the α permutation. It is simply any† permutation that takes points in B that correspond to edges on a single Hamilton cycle to consecutive points. In the next figure, we show a Hamilton cycle decomposition of K_5, the induced partition of $C_{5,10}$, and a possible choice of the permutation α. In later chapters, we will use α to refer not only to the permutation, but also to the partition.

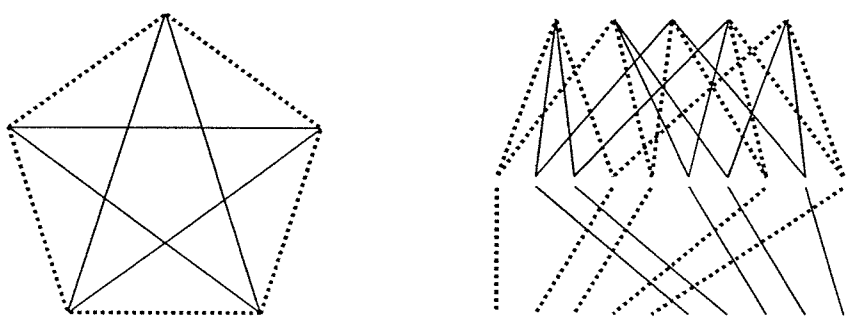

Figure 3.3

† There are many ways of doing this, and there appears to be some second order sensitivity to the choice.

There is another way to decompose K_p into Hamilton cycles. We begin with a description of a single cycle.

Theorem 3.4 Let $1 \leq k \leq \frac{p-1}{2}$, and take all arithmetic to be mod$p$. Then the cycle described by the permutation of the vertices:

$$\pm(\frac{p-1}{2})k,\ \mp(\frac{p-1}{2}-1)k,\ \ldots,\ 2k,\ -k, 0,\ k,\ -2k,\ \ldots,\ \pm(\frac{p-1}{2}-1)k,\ \mp(\frac{p-1}{2})k$$

is a Hamilton cycle in K_p.

Proof The terms of this cycle differ from those of the first formulation in Theorem 3.2 only in sign. In that cycle, every term to the left of zero has negative sign, and every term to the right has positive sign. Since that cycle is (disregarding sign) symmetric about zero, alternating signs simply permutes the elements. But the proof of Theorem 3.2 did not depend on the order of the vertices in the cycle, and hence alternating signs also yields a Hamilton cycle. ∎

Corollary 3.5 Letting k range over all possible values $1, 2, \ldots, \frac{p-1}{2}$, yields a decomposition of the graph K_p into Hamilton cycles.

Proof Again consider the cyclic group $(\mathbb{Z}_p, +)$, but this time write the elements as

$$\alpha^0, \alpha^1, \alpha^2, \ldots, \alpha^{p-1}.$$

Suppose that some edge e occurs in two different cycles, say, the cycles generated by α^r and α^s. Let e be the edge connecting α^{mr} to $\alpha^{-(m+1)r}$. Then either e is the edge connecting α^{ns} to $\alpha^{-(n+1)s}$ or the edge connecting α^{-ns} to $\alpha^{(n+1)s}$. In the first case, we get

$$\alpha^{mr}\alpha^{-(m+1)r} = \alpha^{mr-mr-r} = \alpha^{-r} = \alpha^{ns}\alpha^{-(n+1)s} = \alpha^{-s},$$

which contradicts the assumption that r and s were distinct. In the second case, the same products yield

$$\alpha^{-r} = \alpha^{s},$$

but this is impossible with r and s both less than or equal to $\frac{p-1}{2}$. Thus all the cycles are edge disjoint, and we have a decomposition. ∎

This decomposition, called the *skew-principal,* decomposition defines the permutation β in the same manner as the principal decomposition defines α. Here, for example, is the skew-principle decomposition of K_7.

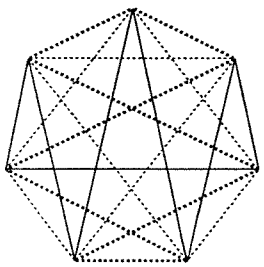

Figure 3.4

If we compare the preceeding figure to Figure 3.2, we observe that the edges in the partition classes of the principal decomposition are nicely scattered throughout the skew-principal. In fact, this is no accident:

Theorem 3.6 The edges of K_p in each partition of the principal Hamilton cycle decomposition are as evenly distributed through the partitions of the skew-principal Hamilton cycle decompositions as possible, that is, of the p edges in any principal partition, there are two in every skew-principal partition but one, and that one has three. Also, each skew-principal partition has three edges from exactly one principal partition.

Proof Consider the second description in Theorem 3.2: a principal cycle is one consisting of all edges (v_i, v_j), where $|i - j| = k$. We can think of the value k as the length of the edge, and thus every partition class of the principal decomposition contains p edges of a certain length, and no edges of any other length. We will now show that every partition class of the skew-principal decomposition contains

three edges of one length, and two of every other length. Write the skew-principal decomposition as:

$$\pm(\frac{p-1}{2})k,\ \mp(\frac{p-1}{2}-1)k,\ \ldots,\ 2k,\ -k, 0,\ k,\ -2k,\ \ldots,\ \pm(\frac{p-1}{2}-1)k,\ \mp(\frac{p-1}{2})k.$$

The edges incident on vertex 0 are both of length k, as is the edge connecting the first and last edges on the list. (We cannot specify the signs without knowing the value of p mod 4, but we do know that they are opposite.) Now consider the absolute values of the differences of successive terms in the list. (These values are the lengths of the edges joining the vertices listed.) Clearly, they are symmetric: this gives us at least two edges of each length. But the terms of the sequence $k, 3k, 5k, 7k, \ldots, (p-2)k$ are distinct, and that gives us exactly two edges of each other length. ∎

4. Digression: Orthogonality

Lemma 3.6 describes a kind of orthogonality relation, and can be extended to the general definitions.

In general, we will say that two connected balanced pattern decompositions of a (v, b, r, k, λ) design in which $\left(v/\frac{b}{v}\right)$ is an integer are *orthogonal* if each class of one partition is made up of $v/\frac{b}{v}$ rows from each class of the other.

If $\left(v/\frac{b}{v}\right)$ is not an integer, then we will say the decompositions are orthogonal if each class of one partition is made up of $\lfloor v/\frac{b}{v} \rfloor + 1$ rows from one class and $\lfloor v/\frac{b}{v} \rfloor$ rows from each of the other classes of the other partition. This relationship is clearly symmetric: it doesn't matter which partition we look at first.

This definition simplifies the statement of the last theorem:

Lemma 3.7 The principal and skew-principal Hamilton cycle decompositions of K_p, where p is an odd prime, are orthogonal.

Proof This is an immediate consequence of Theorem 3.6 and the definition. ∎

Orthogonal connected balanced pattern decompositions decompositions of designs with k greater than 2 (but less than v) are extremely hard to treat. Fortunately, however, the construction we are pursuing does not require the full strength of balanced patterns: it will suffice, at least in the examples we will show, to use near-balanced patterns. As these objects, which we are forced to describe as orthogonal connected near-balanced pattern decompositions of designs, have unequal column sums, they do not appear to be combinatorially interesting in their own right, but are very useful here.

There is an additional condition, related to orthogonality, that we will want later. Ideally, our partitions will be chosen so that every input of the connector will be connected to at least one output in each class of the product partition of the two partitions α and β. This is not always possible—for example, it cannot be done for the $k = 2$ examples—but is very useful. The next two sections will show examples that do allow this property, which we will call the *product partition requirement.*

This requirement is in fact extremely useful. If our connector has α and β partitions for which this holds, then (provided that the partitions are connected patterns) the entire proof of recovery depends only on a property of the permutations we will explore in the next chapter. This suggests, however, that this sufficient condition is probably not necessary, as it is a very blunt instrument.

5. Examples Derived from Complete Block Designs ($k = v$)

A *complete* block design is one in which every symbol occurs in every block. This is a degenerate class of irreducible designs, themselves already a degenerate class of designs, and is therefore not found in the literature. However, as they satisfy the requirements of our constructions and allow simple proofs, they are well worth considering here.

A complete block design corresponds to a complete bipartite graph, that is, a bipartite graph in which every vertex on each side of the bipartition is connected to every vertex on the other side. The incidence matrix of a complete block design (or the reduced adjacency matrix of the corresponding graph) contains no zeros at all.

Here is an example, which we will later refer to as the $2 \rightarrow 4$ complete connector, along with the α and β permutations.

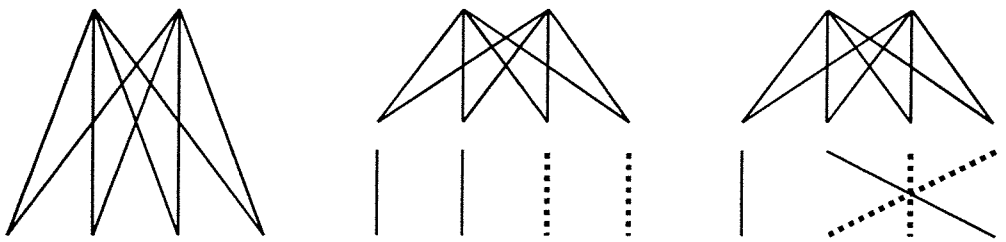

Figure 3.5

This example is deceptively simple. In particular, it has the property that every input is connected to at least one output in each class of the product partition. Although this is trivially true, it is crucially important.

We could have said that in the $2 \rightarrow 4$ complete connector example, every input is connected to exactly one output in each class of the product partition. This is not true of every complete connector. In particular, if $v < \frac{b}{v}$ we do not have enough outputs connected to each input to cover all the blocks of the partitions, and if $v > \frac{b}{v}$, we have more than enough. This symmetry destroying surplus again creates second-order effects that remain to be explored. We will require that we have $\frac{v}{b/v} \geq 1$ in these constructions, which follows from the facts that the permutation α creates $\frac{b}{v}$ blocks of size v, and then β selects $\frac{v}{b/v}$ from each. In the case $v = \frac{b}{v}$, we will say that the constructions is *exact* For example, the cases: $2 \rightarrow 4, 6 \rightarrow 36$, and $7 \rightarrow 49$ are all exact, while $5 \rightarrow 10$ and $6 \rightarrow 30$ are not.

6. Examples from Other Designs ($k \geq 3$)

We will examine a characteristic example, namely, the

$$13, 26, 6, 3, 1$$

design. The incidence matrix of this design can be written as:

1	0	1	0	0	0	0	0	1	0	0	0	0
0	1	0	1	0	0	0	0	0	1	0	0	0
0	0	1	0	1	0	0	0	0	0	1	0	0
0	0	0	1	0	1	0	0	0	0	0	1	0
0	0	0	0	1	0	1	0	0	0	0	0	1
1	0	0	0	0	1	0	1	0	0	0	0	0
0	1	0	0	0	0	1	0	1	0	0	0	0
0	0	1	0	0	0	0	1	0	1	0	0	0
0	0	0	1	0	0	0	0	1	0	1	0	0
0	0	0	0	1	0	0	0	0	1	0	1	0
0	0	0	0	0	1	0	0	0	0	1	0	1
1	0	0	0	0	0	1	0	0	0	0	1	0
0	1	0	0	0	0	0	1	0	0	0	0	1
1	0	0	1	1	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	0	0	0	0	0	0
0	0	1	0	0	1	1	0	0	0	0	0	0
0	0	0	1	0	0	1	1	0	0	0	0	0
0	0	0	0	1	0	0	1	1	0	0	0	0
0	0	0	0	0	1	0	0	1	1	0	0	0
0	0	0	0	0	0	1	0	0	1	1	0	0
0	0	0	0	0	0	0	1	0	0	1	1	0
0	0	0	0	0	0	0	0	1	0	0	1	1
1	0	0	0	0	0	0	0	0	1	0	0	1
1	1	0	0	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	0	0	0	0	1	0
0	0	1	1	0	0	0	0	0	0	0	0	1

This matrix can clearly be decomposed into two connected balanced patterns: the circulant that makes up the first thirteen rows, and the one that makes up the second thirteen.

The connector derived from this design is shown in the next figure, along with the α permutation that corresponds to dividing the matrix into its two circulants.

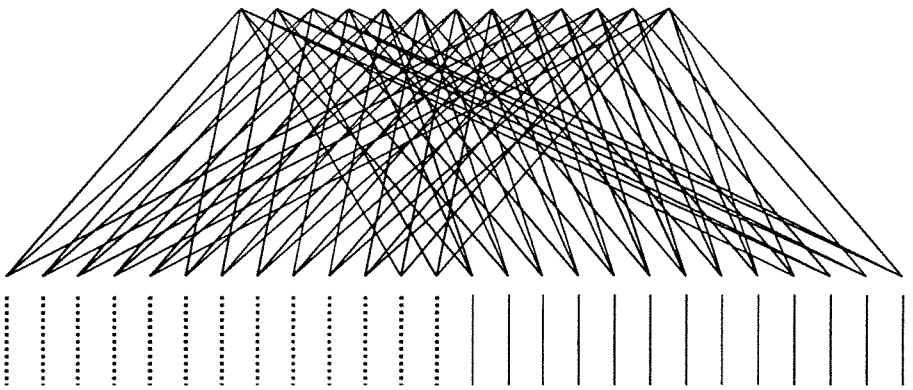

Figure 3.6

The second decomposition has in its first block rows

$$1, 2, 5, 6, 7, 10, 11, 14, 16, 19, 21, 24, 26,$$

that is, the rows of the matrix

$$\begin{array}{cccccccccccccc}
1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}$$

and in its second block, rows

$$3, 4, 8, 9, 12, 13, 15, 17, 18, 20, 22, 23, 25,$$

that is, the rows of the matrix

$$\begin{array}{cccccccccccc}
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{array}$$

The first block has all column sums equal to three except for two columns with sum two, and two with sum four. Since all the columns of the entire design sum to six, the other block has all column sums three except for two fours and two twos. Although this example was constructed by hand, there is sufficient regularity in the procedure to be quite confident that other examples are relatively commonplace. The decomposition can be found by a simple backtracking process, and, as we have three ones in each column of each block of the first balanced pattern decomposition and need to divide them into only two classes, the system is not as tightly constrained as in the Hamilton cycle example.

The critical observation about this design is that it has the property that each input has three descendants in each block of the α partition, and at least one of these descendants is in each block of the β partition. Because of this, this design can be analyzed in exactly the same way as the complete $C_{2,4}$ construction.

Chapter 4

Self-Similar Permutations Based on Latin Squares

1. Basic Ideas

We originally wanted to solve a problem that arose from some characteristics of trees. With the material that has been developed thus far, we can build a structure (such as that in Figure 2.2) that has a significant amount of redundancy in its connectivity, but is nonetheless still unable to guarantee finite losses. The connectors in Figure 2.2 are joined like this:

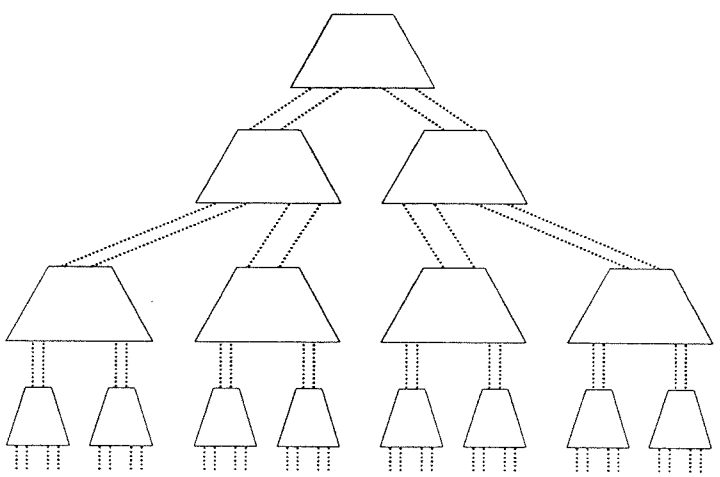

Figure 4.1

This structure looks very much like a tree, and, indeed, it is only slightly better than a tree. While it can tolerate the loss of a single vertex, if all the inputs to any single connector fail, we lose a fraction of the entire structure.

It is clear from this picture that we need to break up the outputs from the connectors and regroup them so that the inputs to each connector are outputs of several different connectors. We already have a way to divide the outputs of a connector into blocks of equal size, each with as many vertices as there are inputs to a connector. In fact, we have specified two different ways to do this: α and β .

The regrouping will have to be done at each level, and will have to be done in such a way that the structure cannot be divided into two or more parts that have no edges crossing the division. If it could be divided in that way, then the loss of all the vertices at some level in one part would again lead to the loss of a fraction of the entire structure.

In this chapter, we will develop series of permutations that can be used in combination with the partitions of the previous chapter to build highly fault-tolerant structures.

2. Digression: Latin Squares

A *Latin square of order* n is an n by n arrangement of integers. In the usual definition, each of the integers 1 to n appears exactly once in each row and once in each column. For our purposes, we prefer a slight variant, and demand instead that the integers range from 0 to $n-1$. Latin squares have been studied extensively; see, for example, [12].

First, Latin squares exist for all orders. Second, there are, in general, many different (non-isomorphic under permutations of rows and columns) Latin squares of a given order. Third, a Latin square may or may not be symmetric about its main diagonal, and that main diagonal may or may not include each of the integers 0 to $n-1$ exactly once.

We show two examples of order three that illustrate the idea, but caution the reader against extrapolating too far from them.

0	1	2
1	2	0
2	0	1

0	1	2
2	0	1
1	2	0

3. The Permutations

Notation

We will be examining sequences of permutations on n^i objects. Without loss of generality, these objects are numbered, starting with 0. To avoid cascading subscripts, we will assume that at any given moment the value of n is clear from the context, and simply denote a permutation by π_i. For a given n, then, π_i operates on the integers 0 through $n^i - 1$, and we denote the image of k under π_i by $\pi_i(k)$.

Our sequences of permutations will always start with a permutation on n^2 objects. For clarity, we will denote these permutations by π_2.

Base cases

The base cases of the recursive definitions of the permutations we are developing in this chapter may be described in two ways. One description is simply a pair of criteria that define a class of permutations, the other is an algorithmic description based on Latin squares. The criteria may seem arbitrary at this point, but later they will appear quite naturally in the construction of communication structures. At this point, however, we will simply present the two characterizations as *mathematika ex machina*, and prove their equivalence.

We begin by assuming that we are interested in permutations on k^2 objects. Suppose that the objects are labeled 0 through $k^2 - 1$, and that they are *a priori* grouped into blocks, with the first block consisting of the first k objects, the second

block consisting of the second k objects, and so on. Our first characterization requires only that our base permutations satisfy the following two criteria:

1. The relative positions of objects within blocks are preserved, that is, $\pi(x) \equiv x \pmod{k}$.
2. No two items from a single block are mapped into the same destination block. We observe that the second criterion immediately requires, by the pigeonhole principle, that exactly one object per block be fixed by the permutation.

Let us now consider a different characterization of some permutations. Suppose again that we have k^2 objects labeled and grouped as above. Suppose further that we have a Latin square of order k. We will use the Latin square to generate a permutation thus: interpret the j^{th} entry in the i^{th} row of the Latin square as the index of the destination block for the j^{th} object of the i^{th} block, and require that no object be mapped out of its original relative position within a block.

It is an immediate lemma that the two characterizations are equivalent. The key point to consider in the proof is that every row of a Latin square of order n is a permutation of the integers 0 to $n-1$.

The recursive definition

We will now define the sequence of permutations $\pi_2, \pi_3, \ldots$, based on a given Latin square of order n. The definition is recursive, so we assume that we already know π_2 through π_{i-1} as we define π_i.

The idea is very simple. The permutation we are defining will operate on n^i objects. Once we have defined π_2 directly from the Latin square, as above, there are two steps in the recursion:

1. Divide the objects into n^2 blocks. Don't do anything sneaky: the first block is 0 through $n^{i-2}-1$, the second is n^{i-2} through $2n^{i-2}-1$, etc. Permute these blocks using π_2.
2. Now divide the objects into n^3 blocks. Apply π_{i-1} to blocks of size n^{i-1}. This will require n applications of that permutation.

An example is in order. Let us suppose that we wish to permute 7^{11} objects. Our base permutation is on 49 objects, and arises from some Latin square of order 7.

We first divide the objects into 49 blocks of 7^9 objects. We use our base permutation to shift them around. (This shifting, it should perhaps be noted, is assumed not to disturb the order of objects within a block.) Now we regroup the objects into 343 ($= 7 \times 49$) blocks of size 7^8. We take these blocks 49 at a time. There are two ways to view what happens next: either we simply apply the permutation of 7^{10} objects to each of 7 groups; or, to keep the inductive nature of the definition visible, we apply the base permutation to each of 7 groups of 49 blocks of 7^8 objects.

4. Rapid (Digit-by-digit) Calculation

Having defined our class of permutations π_i, we are faced with a practical problem: how difficult is it to compute the value of $\pi_i(n)$? If we try to use the definition as our algorithm, we find that we are computing the entire permutation in an extremely inefficient fashion. This is not, however, the only option, and we now turn to a description of an efficient method to compute the value of $\pi_i(n)$ for arbitrary instances of the permutations we have been examining. We will first state the algorithm, and then prove that it does in fact compute what we want.

The algorithm is extraordinarily simple. Let us suppose we have used a Latin square of order n to define a sequence of permutations. We wish to compute $\pi_i(k)$. Write k in base n, including leading zeros if necessary so that we have an i digit number. Suppose this representation of k is $d_{i-1}d_{i-2}\ldots d_2d_1d_0$. Then the base n representation of $\pi_i(k)$ is:

$$(d_{i-1} \diamond d_{i-2})(d_{i-2} \diamond d_{i-3}) \cdots (d_2 \diamond d_1)(d_1 \diamond d_0)d_0,$$

where $a \diamond b$ is the digit found at the intersection of the a^{th} row and b^{th} column of the Latin square.

For example, suppose we wish to compute $\pi_7(1292)$, given that our base Latin square is:

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

We write 1292 as 0020132_5, use our Latin square as a function table, and calculate

$$\pi_7(1292) = 0241032_5 = 8892.$$

The proof of the correctness of this algorithm is actually quite simple. If we consider the first step of the recursive definition of the permutations in terms of the base n representation of k, we note that permuting n^2 blocks in the manner specified by π_2 corresponds exactly to changing only the highest order digit in the representation, and, further, that changing it in a way that depends only on its old value and the value of the next highest order digit.

The second step of the recursive definition of the permutation leaves all the highest order digits unchanged since it is operating on blocks of size n^{i-1}. Altogether, the recursive definition simply specifies that k be moved through a series of intermediate steps; as those steps are

$$(d_{i-1} \diamond d_{i-2})d_{i-3} \ldots d_2 d_1 d_0, d'_{i-1}(d_{i-2} \diamond d_{i-3})d_{i-3} \ldots d_2 d_1 d_0,$$
$$d'_{i-1}d'_{i-2}(d_{i-3} \diamond d_{i-4})d_{i-4} \ldots d_2 d_1 d_0, \ldots, d'_{i-1}d'_{i-2}d'_{i-3} \ldots d'_2(d_{i-1} \diamond d_{i-0})d_0,$$

we can simplify the calculation by computing these values just for the single item k, rather than for the entire set of n^i objects.

5. Non-adjacency of Descendants

We have said that we want to reduce the interactions of the generations of descendants of a single vertex as much as possible. Since any vertex is in some block (after the first decomposition), if we can show that our permutations π separate the blocks containing descendants of a single vertex, we will have moved a great deal closer to the proofs of recovery that we want. We will now examine the general behavior of the permutations, behavior that is independent of our choice of α and β.

Pick some π_i based on a Latin square of order j. Define the following operations on a set of integers, X :

$$\sigma : X \longrightarrow \{x_k, x_k + 1, \ldots, x_k + j - 1 | x_k \in X, \mathrm{mod} j\},$$

and

$$j : X \longrightarrow \{jx_k | x_k \in X\}.$$

Further let

$$\rho_i = \sigma \circ \pi_{i+1} \circ \sigma \circ j,$$

that is,

$$\rho_i(X) = \sigma(\pi_{i+1}(\sigma(j(X))))$$

It is easiest to understand what this operation is doing if we consider the base j representation of the x_i, and think of x_i as indexing a block of input vertices. Select a particular x_i. In base j, it is written $d_{i-1}d_{i-2}\ldots d_2d_1d_0$. When we multiply by j, we simply get $d_{i-1}d_{i-2}\ldots d_2d_1d_00$. The operation σ yields a set of numbers that index all the blocks (after α but before π) that could possibly contain a descendant of a vertex in block x_i. (Whether or not they actually do contain such a vertex depends on the specific connector we are using.) These blocks are then moved around by π. The second application of σ yields those blocks that might, after the β partition, contain a descendant of some vertex in the original block.

We can now state a fundamental theorem about the possible interaction of these descendants:

Theorem 4.1 No two blocks containing descendants of a single vertex can be brought by a permutation π_i to positions where their vertices can be mingled by β unless they occur in a level where every block might contain a descendant of the given vertex. If $i \geq 1$ and $n \leq i-1$ then:

$$|\rho_{i+n}(\rho_{i+n-1}(\cdots(\rho_{i+1}(\rho_i(\{x\})))\cdots))| = j^{2(n+1)}.$$

Proof We will proceed by induction on n, and will be interested not only in the size of the sets involved, but in their composition.

Suppose we have a set of j^{2n} $i+n$-digit numbers where the first $i-n$ digits of all the numbers are identical, and every possible combination of the last $2n$ digits occurs. That is,

$$X = \underbrace{\{\underbrace{d_{i+n-1}d_{i+n-2}\ldots d_{2n+1}d_{2n}}_{i-n}\underbrace{*\,*\cdots*}_{2n}\}}_{i+n}.$$

where $*$ may be any digit. We wish to show that if we apply ρ_{i+n} to X, the result is the set

$$X = \{\underbrace{\underbrace{d'_{i+n}d'_{i+n-1}\cdots d'_{2n+3})d'_{2(n+1)}}_{i-n-1}\underbrace{*\,*\cdots *}_{2(n+1)}}_{i+n+1}\}.$$

Note that this is a set of size $j^{2(n+1)}$, which is what the statement of the theorem requires, and that it is of the correct form to allow an induction.

We begin with $n = 0$. The set X consists of a single element, which we can write as $d_{i-1}d_{i-2}\ldots d_2d_1d_0$. This set satisfies our condition, and now we need to determine $\rho_i(X)$. First,

$$j(X) = \{d_id_{i-1}\ldots d_2d_10\}.$$

Note that the subscripts have changed because we now have an $i+1$-digit number. Next,

$$\sigma(j(X)) = \{d_id_{i-1}\ldots d_2d_1*\}.$$

Each element of this set ends in some digit k, and when we apply π_{i+1} to such a number, we get

$$(d_i \diamond d_{i-1})(d_{i-1} \diamond d_{i-2})\cdots(d_2 \diamond d_1)(d_1 \diamond k)k.$$

Since k ranged over every possible value, $(d_1 \diamond k)$ also ranges over every possible value. Each value occurs, however, with a single value of k. The final application of σ creates a group of numbers with every possible final digit for each value of k, and we get

$$\rho_i(X) = \{d'_id'_{i-1}\ldots d'_2**\}.$$

This is in the required form, and is of size j^2.

Now we turn to the inductive step. This will be much easier, as it is essentially the same as the base case. The first two operations, $\sigma \circ j$, extend the string and add a $*$ onto the end. The permutation moves things around, fixing the lowest order bit, but converting the last d_j before the $*$s to a $*$. Then the final application of σ turns the lowest order bit into a $*$, and we have the inductive result.

The theorem as stated is simply an observation about the size of the sets involved. We point out, however, that as $j^{2(n+1)}$ is the upper bound on how large

these sets could possibly be, the theorem as stated is sufficient to imply that all the elements are distinct. This is what we actually want for recovery. ∎

Chapter 5

The Solution to the Communication Multitree Problem

1. Putting the Pieces Together

We now have all the components we need to construct examples of communication multitrees with a provable ability to recover from near-catastrophic vertex failures. We will start with the constructions, and then turn to the proofs of recovery properties.

We begin by choosing the components we will use to build the tree. First, we need a connector, together with the parameters v and b. These parameters cannot be chosen entirely independently; for example, if we choose the connector based on K_v, then b is $\binom{v}{2}$. Given a connector, we choose two permutations α and β that are based on our knowledge of the connector. For example, if v is prime and we are using the connector based on K_v, then α and β can be derived from orthogonal Hamilton cycle decompositions.

The choice of connector determines how many vertices are in each level of our multitree. (Alternately, we can begin with a constraint on these numbers and work

backwards to find a good connector.) Level 0 has only one vertex (the root); level i has n vertices, where

$$n = v\left(\frac{b}{v}\right)^{i-1}.$$

We will define permutations α_i and β_i by:

$$\alpha_i(n) = \alpha(n \bmod b) + b\lfloor n/b \rfloor,$$

and

$$\beta_i(n) = \beta(n \bmod b) + b\lfloor n/b \rfloor.$$

There is nothing deep in these definitions: α and β, as we have defined them, permute b objects; these definitions simply divide the n vertices at each level into blocks of size b and apply our original permutations seperately to each block.

Next, we need to choose a Latin square of order $\frac{b}{v}$. This defines a sequence of permutations π_i. From these, define the sequence π'_i by:

$$\pi'_i(n) = v\pi_i\left(\lfloor n/v \rfloor\right) + (n \bmod v).$$

The permutations σ_i can be written as:

$$\sigma_i(n) = \beta_i(\alpha_i^{-1}(\pi_i(\alpha_i(n)))).$$

Again, this is a simple conversion. We introduced the permutations of Chapter 4 in order to move the blocks created by the α permutation; this definition simply divides the vertices at each level into blocks of size v, and then moves the blocks in the way we originally intended.

Now we place the root and the layers of vertices, and begin putting in edges. We describe the construction recursively. First, connect the root to every vertex in level 1. Second, assume that the multitree has been constructed up to level i. Insert a connector with callers

$$(i, 0), (i, 1), \ldots, (i, v-1)$$

and callees

$$(i+1, 0), (i+1, 1), \ldots, (i+1, b-1),$$

and another connecting the next v vertices in level i to the next b vertices in level $i+1$, and so on. Now permute the vertices at level $i+1$, moving $(i+1, j)$ to $(i+1, \sigma_{i+1}(j))$.

For the proofs, we will look at the σ_i not as monolithic objects, but as compositions of the four base permutations.

2. Recovery (Theorems and Conjectures)

Key Ideas

In order more easily to understand the approach to the proofs of this section, it is helpful to consider the following schematic description of the construction, and, in particular, of the interaction between the various permutations.

Assume that we have constructed a communication multitree. We have chosen some connector with v inputs and b outputs. The permutations α_i break each set of b outputs of a single connector into $\frac{b}{v}$ blocks of v outputs. The permutations π_i move these blocks wholesale: that is, if two vertices are in the same block after α_i, they are still together after the subsequent application of π_i. The application of α_i^{-1} "unwinds" the groups of blocks into what would be (if π_i were the identity) their original connectors. At this point, we apply β_i, and the unwinding allows us to use the orthogonality relationship between α and β.

There are two key properties of the substructures used in the constructions that are crucial to the proofs of the recovery theorems. The first is a property of the connectors we use, and the second is a requirement that is needed to make use of theorem 4.1.

The first requirement is that α and β be connected near-balanced pattern decompositions. This is sufficient to guarantee that every proper subset of size i of the set of inputs is connected to at least $i+1$ outputs in each of the blocks of the partition associated with the α permutation.

The second requirement is the one we stated in §I.3.4: that every input be connected to at least one output in each class of the product partition. Recall that in

the description of theorem 4.1 we discussed "every block that could possibly contain a descendant of a vertex in block x_i." This requirement is sufficient to guarantee that each such block does in fact contain such a descendant.

Communication Multitrees derived from Complete Block Designs

Suppose we build a communication multitree from complete connectors $C_{n,rn}$. We choose orthogonal permutations α and β, and a sequence of permutations π_i based on a Latin square of order r. We finally construct the tree from these pieces as described in §I.5.1.

Now suppose that at some level of that multitree, all the vertices but one fail. We say that the multitree *recovers* if there is a later level in which every vertex is a descendant of the single working vertex at the level where the failures occurred.

We can now state and prove the first of our main results:

Theorem 5.1 Any communication multitree constructed from the complete connector $C_{n,rn}$ with $r \leq n$, using orthogonal permutations α and β, and a sequence of permutations based on a Latin square of order r is able to recover.

Proof We begin by observing that our two basic requirements are both satisfied. First, since every input to a connector is connected to every output, every proper subset of the inputs of size i is trivially connected to at least $i+1$ of the outputs in each block of the α permutation. The second requirement is also trivially satisfied.

Now consider the level where the failures have occurred, and focus on the single working vertex. It is connected to all of the rn outputs of the connector it is in. The α permutation divides these into r blocks, each of size n. The permutation π moves these blocks in such a way that every consecutive group of n blocks contains at most one of them. Thus, when α^{-1} is applied, none of these vertices can be moved into a position where it has a descendant in common with any of the others.

Now we apply β, and consider the second requirement. Since the input is connected to at least one vertex in each class of the product partition, each of the r blocks of descendants of the working vertex is spread among all of the r blocks within the scope of β.

Another way to look at this is to remember our original statement that a communication multitree is built up from layers of disjoint connectors. Suppose we have just one input at a given level working. The question we are addressing is to determine how many connectors at the next level have working inputs, that is, inputs that are descendants of the single vertex.

The answer to that question is r^2. The reasoning is this: α breaks up the working vertices into r blocks. Then π moves them in such a way that they are well distributed; in particular, they are distributed so that β can place one or more vertices from each block as inputs to each of r distinct connectors. The product partition requirement ensures that there are enough working vertices around for β to place. Theorem 4.1 makes precise the notion of "well distributed," and ensures that the subsequent layers will continue to have the same property.

Let us examine our $2 \rightarrow 4$ connector example once again, and consider the message passing behavior of the multitree. We assume that there is some layer where only one vertex is active. That vertex has four descendants, each of which receives the message. The α permutation divides these into two groups of two, and π moves them apart. The permutation α^{-1} "recombines" them, each pair with a pair of vertices that did not receive the message. Finally, β divides each pair. Thus, we have four blocks with one live input. Each of these passes the message to four descendants, for a total of sixteen at the next level. By theorem 4.1, the permutations operate in such a way that we eventually reach a level where exactly one vertex in every input pair has received the message. At the next level, every vertex receives the message, and we have recovered.

Communication Multitrees derived from Incomplete Block Designs

Our second main result shows that we do not need to have complete connectivity in our connectors. Although it seems much more specific than the last theorem, the ideas work for many other designs.

Theorem 5.2 The communication multitree based on the (13,26,6,3,1) balanced incomplete block design and permutations specified in §I.3.6 recovers.

Proof This proof is very similar to the preceding proof. The chief difference is that at the end, we cannot claim to have full recovery in just one additional level.

We need to make two observations. The first is that the design has sufficient connectivity to satisfy the product partition requirement. This means that the descendants of the one working vertex spread through the multitree without interacting until we reach a level where every connector has at least one working input. (This is exactly analogous to the previous theorem.)

Because α and β are connected near-balanced pattern decompositions, at the next level, every connector must have at least two working inputs. At the subsequent level, each connector will have three. This continues until every connector has thirteen working inputs, i.e. the multitree has recovered. This is an upper bound on the number of levels required for recovery, and could perhaps be improved slightly. But as the first phase requires i levels to deal with failure at level i, which grows as the failure level is further down the multitree, we are not particularly concerned with reducing an additive constant.

Communication Multitrees derived from Complete Graphs

Conjecture 5.3 The communication multitree based on K_p and orthogonal Hamilton cycle decompositions recovers.

If we try to analyze this construction in the same way as the two previous constructions, we run into trouble with the product partition requirement. Since each input vertex has six direct descendants and the α partition divides them into three groups of two, the β partition obviously cannot then move the two vertices in a way that puts one in each of three blocks.

This does not mean that the construction does not recover. If more than one input vertex is working, there can be enough working output vertices for β to move.

3. Commentary on Proofs

Although we have demonstrated that several constructions of communication multitrees are able to recover, we have not yet discussed how long it takes.

We first observe that it will require at least $O(i)$ further levels. This is an immediate consequence of the fixed outdegree of the vertices. At best, the set of live vertices is growing (level by level) at a rate of r^i, while the multitree is getting wider at a rate of $(\frac{r}{k})^i$. In the case of the complete designs, this bound is achieved, and a single vertex at level i is an ancestor of every vertex at level $2i$.

In the case of the other constructions that we have proved to recover, we have proved that we reach a level where every input block has at least one live vertex. (In general, there will be more, but we must stick to the worst case since we are interested in complete recovery.) Because that vertex is not connected to every output vertex of the block, it may take a little longer to recover. But since any collection of i live inputs yields at least $i+1$ live outputs, it can only take a constant number of additional levels. (The constant depends, of course, on the connector we have chosen, but not on the level at which the failure occurs.)

Although having at least one live vertex in every block is sufficient for recovery, we have not proved it to be a necessary condition. We will return to this in the next section.

We have entirely ignored second-order effects in the proofs. For example, a single live input vertex in the 13-26-6-3-1 construction passes the message to six descendants. We have used our knowledge of the eventual positions of only four of them. Although this would only affect the constant, and not the dominating $\log i$ term, it might be of importance in the study of a less extreme failure model.

4. Future Directions

There are an enormous number of possible extensions to the work presented here. Not only are there many unanswered questions about communication multitrees, but much of the material developed in Chapter 3 is of independent interest.

In the study of Hamilton cycle decompositions, the foremost question is if it is possible to extend the constructions to K_n for any odd number n. There is one well-known method of decomposing K_n into Hamilton cycles, but it does not provide a route to an orthogonal mate. There are also questions about isomorphisms, as the

decompositions are clearly invariant under several sorts of transformations of the graph.

In design theory, it would be interesting to know if there are any designs that can be divided into balanced patterns in two orthogonal ways.

There is a lot of room for study of second-order effects. An interesting example to consider is a degenerate multitree based on the projective plane of order two. Each input vertex is connected to three output vertices. It is possible to choose a permutation such that every such set includes at least one ancestor of every vertex in the next level, but not all permutations do so. This could become important when considering other failure models, as we would probably want to have every vertex have as large a set of ancestors at each level above it as possible. This number is certainly sensitive to the choice of α and β.

It would be interesting to study the behavior of randomized or partially randomized communication multitrees. We could put edges in from level to level entirely at random, subject only to the constraint that the expected out-degree of each vertex be limited. Or we could use connectors but choose each permutation σ_i at random. These designs would still be constrained by the $\log n$ lower bound, but might have very good properties in other failure models. They would also provide an interesting standard to compare the deterministic constructions to.

Analysis of the oft-mentioned different failure models is a very high priority.

There may be a relation between the sparseness of the connectors used and the recovery properties of the resulting communication multitree. If getting a live vertex into every block is not a necessary condition, what is the weakest condition that is sufficient for recovery?

Finally, it would be interesting to develop some cost metrics that would be able to discriminate between communication multitrees based on complete connectors and those based on sparse connectors.

Part II

Non-Blocking Switches

Chapter 6

The Non-Blocking Switch Problem

1. An Informal Description

The origins of this problem lie in telephony. The problem is simple: the telephone company has a great many subscribers, any one of whom may at any moment want to call any other. The telephone company's job is to make that connection possible, provided, of course, that neither subscriber is already engaged in a call.

2. The Formal, Mathematical Formulation

A switch is a device for creating electrical connections between elements of two sets, the inputs and outputs of the switch. In most actual constructions, the chief component of the cost is the elementary switching elements, called *crosspoints*, rather than the wires that interconnect them. In the type of analysis that we will be doing, most of the physical switch is abstracted away, and the focus is on counting the crosspoints.

Switches are frequently drawn like this:

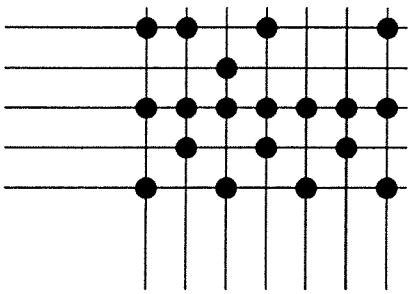

Figure 6.1

The inputs are on the left, the outputs are on the bottom, and the heavy spots mark the positions where a crosspoint is placed to allow the connection of two wires.

Switches are sometimes drawn as bipartite graphs, with the inputs and outputs on either side of the bipartition, and an edge wherever a crosspoint is to be placed. For example, the following picture shows the same switch as in the first example.

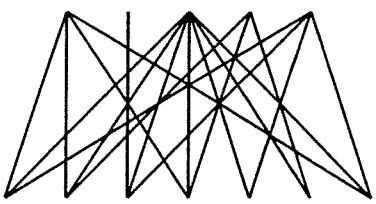

Figure 6.2

The above example is a *single-stage* switch. There is every reason, however, to consider multi-stage switches. These are simply cascaded stages of single switches, with the requirement that any two paths that are to be used simultaneously must be vertex disjoint. Note that this immediately implies that they are also edge disjoint. If we visualize our switches as bipartite graphs, we immediately note some resemblance to the communication multitrees of Part I.

The problem that we will be examining here is the construction of an $N \times N$ *non-blocking* switch. This is a switch that connects N inputs to N outputs, with the property that any idle input may be connected to any idle output regardless of the state of the rest of the network. We wish to do so while using as few crosspoints as possible. The switch may be single stage or multi-stage.

3. Bounds from the Literature

There are three bounds on switch size that are of interest. The first is the trivial N^2 bound. Clearly, if every input is directly connected to every output through a dedicated link, the switch is non-blocking. But this upper bound is extremely expensive, prompting the search for smaller switches.

The lower bound on the crosspoint count of an $N \times N$ non-blocking switch is $N \log N$. The crux of the argument is that in order for the network to have $N!$ possible states, it must have $\log(N!)$ crosspoints, or $N \log N$. A more complete argument is given in [31], where the result is attributed to Shannon.

There are two proofs of the existence of $O(N \log N)$ non-blocking switches. The first is based on properties of random graphs. Due to Pinsker and Bassalygo, it suffers from the deficiency that it provides no certain method for constructing a switch. Even if we were to construct a switch with random interconnections in a manner that assured us that it was, with extremely high probability, non-blocking, we would still face a very difficult control problem. After all, it is not enough to know that a connection path exists: if we are to use a switch, we must know how to find the connection.

More recently, Gabber and Galil [14] have devised an explicit construction of a class of graphs that can be used to build $O(N \log N)$ non-blocking switches. They do not carry out the full construction, however, but say "... N. Pippenger has told us that [these graphs] can be used to give an explicit construction of $O(N \log N)$ non-blocking networks." Pippenger remarks elsewhere [18] that "... the constants involved are so large as to render the result completely impractical."

Practical examples are drawn from the third class of $N \times N$ non-blocking switches, which have crosspoint count $O(N(\log N)^2)$. Prior to the work of Pinsker and Bassalygo, this was the best known bound. There is a $4N(\log_2 N)^2$ switch due to Cantor [6], and this switch has been refined by Pippenger [32] to $16N(\log_5 N)^2$ crosspoints.

The constructions and proofs given by Cantor and Pippenger are lengthy and fairly difficult. In the next chapter we will present a much easier construction of

Cantor's switch, a construction which not only yields Pippenger's improved bound as a quick corollary, but also suggests a possible route to further improvement.

Chapter 7

An Easy, Better Approach to Switches

1. Digression: Layered Bipartite Conjunction

Conjunction

There are, in the literature [7], definitions of several different kinds of graph multiplication. In this section, we will define a variant of one of them—conjunction—and then, later, will use this product in combination with the material in the first three chapters, to produce non–blocking switches.

We begin with the definition of *conjunction.* Given two graphs, $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, the conjunction is defined by:

$$V = V_1 \times V_2,$$

and

$$((v_{i_1}, v_{j_1}), (v_{i_2}, v_{j_2})) \in E \iff (v_{i_1}, v_{i_2}) \in E_1 \text{ and} (v_{j_1}, v_{j_2}) \in E_2.$$

Bipartite Conjunction

The graph product we will be using is very similar. Suppose we have two bipartite graphs, G and H, with bipartitions (A, B) and (C, D), respectively. Then the *bipartite conjunction* is a bipartite graph with bipartition $(A \times C, B \times D)$ and an edge from (a, c) to (b, d) if and only if G contains the edge (a, b) and H contains the edge (c, d). Note that the number of edges in the bipartite conjunction of two graphs is the product of the number of edges in each graph.

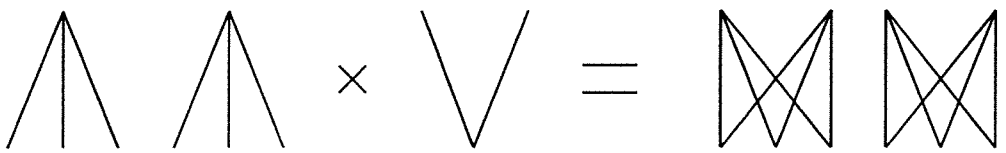

Figure 7.1

Finally, we define *layered bipartite conjunction*. Suppose we have two graphs, like the structures in Chapter 3, with the same number of levels. Then we multiply them by multiplying (bipartite conjunction) the bipartite graphs between corresponding pairs of levels.

An example: the FFT

We turn now to an elegant example of layered bipartite conjunction, first brought to our attention by Sandeep Bhatt.

The well-known network that is often, as in [39], called the FFT is simply the layered bipartite conjunction of two binary trees, one pointing up and the other pointing down.

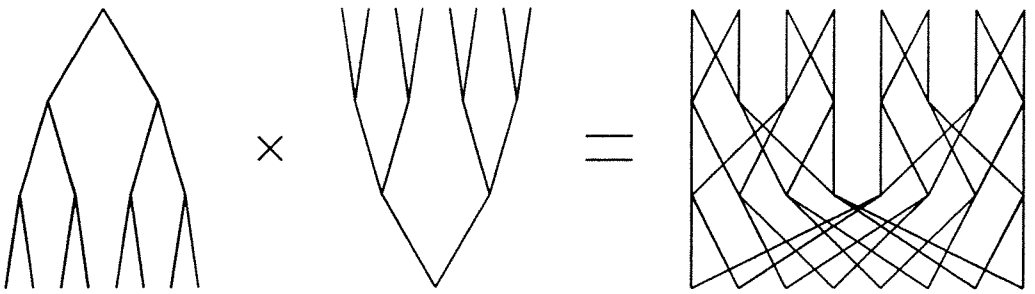

Figure 7.2

It is worth noting the similarity between each layer in this product and Figure 7.1.

2. A New Construction of Old Switches

We will now build a $(2^n - 1) \times (2^n - 1)$ non-blocking switch using $4n^2 2^n + 2n2^n$ crosspoints. This switch is called the Cantor network, and was first described in [6].

The switch is built from two copies of a single structure that looks like this:

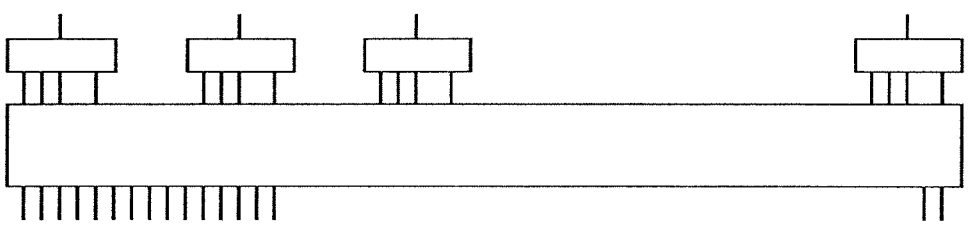

Figure 7.3

This structure has the property that any unused input may be connected to at least one more than half of its outputs, regardless of the state of the rest of the network. If we connect such a structure to a mirror reflection of itself, then any unused input and any unused output in the resulting switch will—by the pigeonhole principle—see at least one common point in the center, and thus we have a non-blocking switch.

The heart of our construction is the layered bipartite conjunction of n binary trees of depth n with a single upside down binary tree of the same depth:

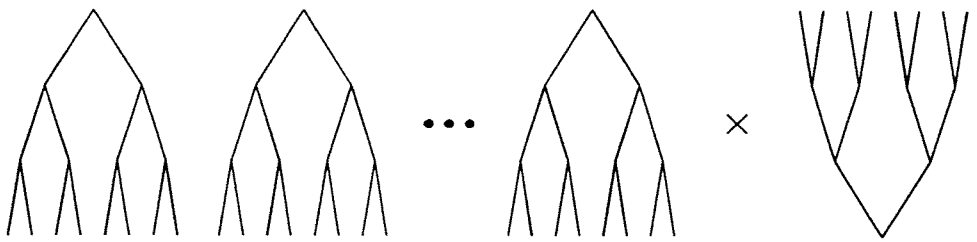

Figure 7.4

This product can be viewed as n disjoint FFTs, each with 2^n inputs. Each input is connected to each of the FFTs, and the entire structure can be drawn as:

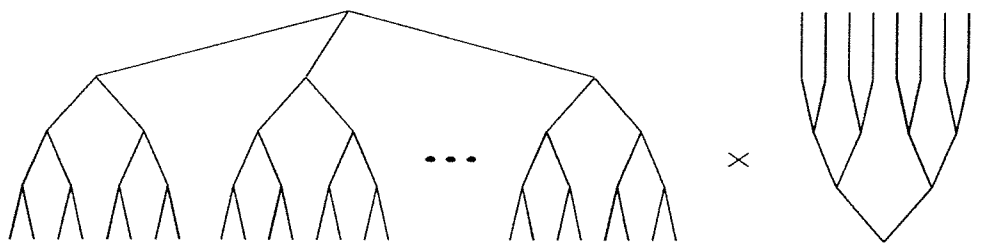

Figure 7.5

Consider a single input. It sees (that is, the induced subgraph below it is) n edges, each leading to the root of a binary tree with 2^n leaves. In the absence of any other activity in the switch, it would be free to choose any of the $n2^n$ outputs of the structure. Other calls may, however, block access to some of those outputs, and we will now upper bound that blocking. The first layer of bipartite conjunction cannot cause any interference. The second layer of bipartite conjunction can introduce only a single call from another input. As we are now at the third level of vertices, this call can block off access to exactly one-half of one of our n binary trees. The next layer of bipartite conjunction introduces two calls, but as they are one level further down, they can together block off at most half of a tree. (Each call can block access to a quarter of a tree, and so—if they are both disjoint the half tree we lost at the second layer—can block off at most another half tree. This is repeated n times, and in the end we lose exactly n half trees, or access to $n2^{n-1}$ outputs. We need to

provide access to $(\frac{1}{2}+\epsilon)n2^n$ outputs, so we will designate one input as permanently unavailable. In the worst case, this input is one of the 2^{n-1} that we consider only at the last layer of merging, but even then it is sufficient to guarantee access to one additional output. We now have a structure like that in Figure 7.3, and can, by connecting it to its mirror reflection, construct a non-blocking switch.

The next question we face is to determine the crosspoint count of this switch. The first layer of bipartite conjunction creates 2^n copies of the n-ary tree structure that connects each input to the binary trees, for $n2^n$ edges. Each subsequent layer of bipartite conjunction creates $2 \times n \times 2^n$ edges, and there are n such layers. The total for the switch is twice (because of the reflection) the sum of these two terms, or $2n2^n + 4n^2 2^n$, as required.

The requirement that one input (and, correspondingly, one output) be left unused explains the $2^n - 1$ in the statement of the claim, but does not affect the order of the result, $N \log_2^2 N$.

Let us now consider a switch made in exactly this way from r-ary trees rather than binary trees. We again begin with a half-switch which we will eventually connect to its mirror reflection. Suppose that each input is connected to k r-ary trees of n. We are going to solve the equations for k in terms of n and r.

In this construction, the second layer of bipartite conjunction introduces not one, but $r-1$ interfering calls. Each of these can block access to $\frac{1}{r}$ of a tree. At the next layer, $r(r-1)$ new calls can each block access to $\frac{1}{r^2}$ of a tree. In the end, we can lose access to at most

$$n\left(\frac{r-1}{r}\right)r^n$$

outputs. If we equate this to half of the original total and solve, we get:

$$k = 2n\left(\frac{r-1}{r}\right).$$

This now allows us to build an $(r^n-1) \times (r^n-1)$ non-blocking switch using

$$4n^2\,(r-1)\,r^n + 4n\left(\frac{r-1}{r}\right)r^n$$

crosspoints. We are interested in the dominant term. If we let $N = r^n$, then we have constructions of non-blocking switches with

$$4(r-1)N\log_r^2 N$$

crosspoints. For $r = 5$, this is a $16N \log_5^2 N$ crosspoint non-blocking switch—Pippenger's result—and elemenatry manipulation shows that this is the best we can do with this particular construction.

3. A New Switch

If we turn back to Figure 7.4, we notice that the pieces used in the construction in the last section were very simple. In fact, of the two graphs that we combine with layered bipartite conjunction, one is not even connected! And if we now look at the construction, we notice that the problem that each input faces—that it may lose access to a fraction of the outputs due to a single call being routed through a vertex—bears a striking similarity to the problem that we attacked in Part I.

Suppose we attempt to apply the theory of communication multitrees to the problem of non-blocking switches. We will begin the same general approach to the construction, but this time we will consider the layered bipartite conjunction of m communication multitrees based on the 2–4 complete connector with a single upside down binary tree:

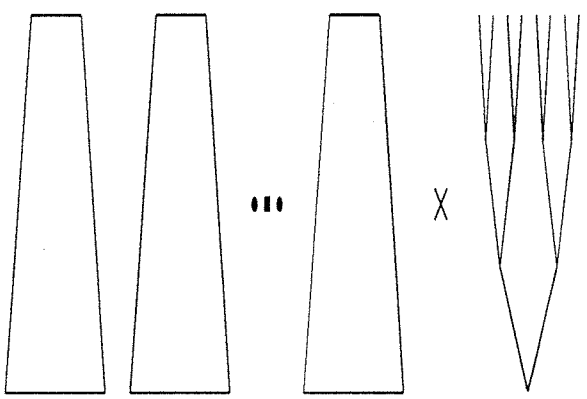

Figure 7.6

This time we will consider the blocking from the bottom up. At the last stage of merging, there are 2^{n-1} calls we have not dealt with further up. Each of these can block access to a single output. At the next layer up, every pair of calls that

are routed through the two inputs to a single connector can block access to four outputs. (This comes from the fact that our communication multitrees are built from disjoint identical connectors.) Thus, these 2^{n-2} calls can—provided that there are enough of them to be grouped in pairs—block access to two outputs each. At the next level, the calls must come in blocks of four to block access to four outputs each. In general, a block of calls merging i layers up from the outputs must be of size 2^i to block access to 2^{n-1} outputs. If it is any smaller, the recovery property of the communication multitree ensures that the calls can block only one output each (instead of 2^i each). But the blocks of merging calls can continue to get larger at this rate only for half the depth of the structure. Thus, we can express the total number of potentially inaccessible outputs as:

$$\underbrace{\cdots + \left(1 \times 2^{\frac{j-5}{2}}\right) + \left(1 \times 2^{\frac{j-3}{2}}\right)}_{\left(\frac{j+1}{2}\right)} + \underbrace{\left(2^{\frac{j-1}{2}} \times 2^{\frac{j-1}{2}}\right) + \cdots + \left(4 \times 2^{j-3}\right) + \left(2 \times 2^{j-2}\right) + 2^{j-1}}_{\left(\frac{j+1}{2}\right)}$$

If we set this equal to half of the total number of outputs, we get:

$$\left(\frac{j+1}{2}\right) 2^{j-1} + 2^{\frac{j-1}{2}} = \frac{m}{2} 2^{j+1},$$

This yields:

$$m \approx \frac{j}{4}.$$

Unfortunately, the total number of crosspoints in the structure is $8mj2^j$, so we once again get a $4N \log^2 N$ non-blocking switch. It appears, however, that this switch has some fault-tolerance that is lacking in the Cantor switch.

4. Commentary, Speculation, and Future Directions

The construction in the preceding section did not answer the questions raised at the beginning of that section. The next step is to consider using connected graphs on both sides of the layered bipartite conjunction.

The first case to be considered is to determine for what values of n the layered bipartite conjunction of two oppositely oriented (i.e. one inverted relative to the other) n-layer 2–4 complete connector based communication multitrees is usable as half of a non-blocking switch. The next cases would be those built from identical oppositely oriented multitrees based on other connectors, and eventually one would want to consider arbitrary layered bipartite conjunctions of communication multitrees.

Part III

Kolmogorov–Chaitin Metric Spaces

Chapter 8

An Algorithmic Information Theoretic Metric

1. Existence and Construction

Introduction

In this chapter we begin the study of a metric on finite binary strings based on the algorithmic theory of information. We demonstrate that it is possible to construct a metric in which the distance between two strings is a measure of the amount of specification required for a universal computer to interconvert the strings. We prove two topological theorems relating this metric, called the Kolmogorov metric because of its relation to Kolmogorov complexity, to the more familiar Euclidean and Hamming metrics.

This work will provide new mathematical foundations for pattern recognition and inductive inference. In pattern recognition, we have noticed that Hamming distance fails to capture many important types of similarity. For example, a photographic print and the negative from which it was produced are enormously widely separated in Hamming space, but visually almost identical.

Our thinking can be summed up (in pidgin mathematics) thus:

$$\text{Kolmogorov distance} \supset \text{visual distance} \supset \text{Hamming distance}.$$

The interpretation of these *intuitive* realtions is the following. If two images are close in the Hamming sense, then they are necessarily close in a visual sense, and, as we need only list which pixels differ to transform one into the other, must be close in the sense of Kolmogorov. If two images appear close to a human, they must be close in an algorithmic sense: specify the program that verifies their visual similarity. But as the positive/negative examples show, their Hamming distance can be great. Finally, we can take a very short program that produces a pseudo-random string and xors it with an image; such an algorithm can make two visually disparate images extremely close in Kolmogorov space.

In inductive inference, the process of learning from examples can be described in terms of minimization of the Kolmogorov diameter of a set of strings. Systems that learn, we suggest, attempt to find sets that encompass all (or most) of the examples presented and that have all of their members fairly similar. This notion is especially appropriate in the domain of learning for image recognition.

Combinatorial Observations

We need a couple of simple combinatorial observations. First, consider the binary trees (not necessarily balanced) with i leaves (terminal nodes). Let $f(i)$ denote the number of such trees. We obtain the following recurrence:

$$f(1) = 1,$$
$$f(n+1) = \sum_{j=1}^{n} f(j)f(n+1-j).$$

Elementary methods (see, e.g. [5]) yield the closed form solution:

$$f(i) = \frac{1}{n}\binom{2n-2}{n-1}.$$

Note that $f(i) < 4^i$. Thus, on information theoretic grounds, $2i$ bits suffice to specify any such tree. There is obviously an effective procedure for tabulating the trees with i leaves and assigning to each a codeword of length exactly $2i$.

Note also that, as there are $i-1$ non-terminal nodes in such a tree, we can specify a traversal with $i-1$ bits. At each non-terminal node we specify whether to traverse the right or left subtree first. Observe that a traversal induces an ordering on the leaves.

Construction

We are now ready to define a Kolmogorov–Chaitin Metric Space. The notation used in this section is the same as that in Zvonkin and Levin and it is very similar to that used in Liu. Let **U** denote a fixed Universal Turing Machine. We define the following specific model of computation, $\mathcal{U}$. It includes Universal computation, but has some special syntactic features that make the proofs easier. First,

$$(q_0, \lambda, \#, 0\mathbf{w}) \longrightarrow_{\mathcal{U}}^{*} (q_1, \lambda, \#, \mathbf{w}).$$

(Where q_0 is the starting state, q_1 the halting state, λ designates the empty string, and $\#$ is the blank symbol.) This says that $\mathcal{U}$ takes any string prefixed with a 0 and strips the 0 off, leaving as output the given string. Another way to interpret this is that the encoding of the Everhalting machine over $\mathcal{U}$ is "0."

Second,

$$(q_0, \lambda, \#, 10^i 1 m n \rho(\mathbf{M}_1)\rho(\mathbf{M}_2)\ldots\rho(\mathbf{M}_i)\mathbf{w}) \longrightarrow_{\mathcal{U}}^{*} (q_1, \lambda, \#, \mathbf{v}),$$

where:

- $|m| = i$, $|n| = 2i$,
- n specifies a binary tree with i leaves,
- the first $i-1$ bits of m specify a traversal (the last bit of m is ignored),
- and $\mathbf{v}$ is the result of simulating the action of **U** on the first encoding in the ordering specified by m and n to $\mathbf{w}$, the second encoding to the output of the first computation, and so on. Note that, since we are given the structure of the tree by n, we can use a recursive specification for the traversal. That is, if the first bit of m is a "0" we traverse the left subtree first, and read the second bit of m to decide what to do in that subtree. But if the first bit of m is a "1" we traverse the right subtree first, and skip over as many bits of m as there are

non-terminal nodes in the left subtree before reading the bit that tells us what to do at the top of the right subtree. The reason for this method of specification is that it permits the merging of two trees in a very simple manner.

We now define the distance between two binary strings by:

$$
\begin{aligned}
d(\mathbf{s},\mathbf{t}) = \min \{ |\mathbf{p}| : \quad & (q_0,\lambda,\#,\mathbf{ps}) \longrightarrow_{\mathcal{U}}^{*} (q_1,\lambda,\#,\mathbf{t}) \\
& \text{and } (q_0,\lambda,\#,\mathbf{p}'\mathbf{t}) \longrightarrow_{\mathcal{U}}^{*} (q_1,\lambda,\#,\mathbf{s}) \\
& \text{where} \\
& \quad \text{either } \mathbf{p} = \mathbf{p}' = 0 \\
& \quad \text{or} \quad \mathbf{p} = 10^i 10k\rho(\mathbf{M}_1)\rho(\mathbf{M}_2)\ldots\rho(\mathbf{M}_i) \\
& \quad\quad \text{and } \mathbf{p}' = 10^i 11k\rho(\mathbf{M}_1)\rho(\mathbf{M}_2)\ldots\rho(\mathbf{M}_i) \\
\} - 1.
\end{aligned}
$$

That $d(\mathbf{x},\mathbf{x}) = 0$ and $d(\mathbf{x},\mathbf{y}) = d(\mathbf{y},\mathbf{x})$ are obvious.

We prove that the definition satisfies the triangle inequality by direct construction. Assume that the minimal length input to $\mathcal{U}$ that interconverts two strings $\mathbf{x}$ and $\mathbf{y}$ is:

$$10^i 1mn\rho(\mathbf{M}_1)\rho(\mathbf{M}_2)\ldots\rho(\mathbf{M}_i),$$

and that the minimal length input that interconverts $\mathbf{y}$ and $\mathbf{z}$ is:

$$10^j 1m'n'\rho(\mathbf{M'}_1)\rho(\mathbf{M'}_2)\ldots\rho(\mathbf{M'}_j).$$

Then the following input interconverts $\mathbf{x}$ and $\mathbf{z}$:

$$10^{i+j}1\left\{\begin{matrix}0\\1\end{matrix}\right\}m''1n''\rho(\mathbf{M}_1)\ldots\rho(\mathbf{M}_i)\rho(\mathbf{M'}_1)\ldots\rho(\mathbf{M'}_j).$$

From the original definitions, $|m| = i, |n| = 2i, |m'| = j$, and $|n'| = 2j$. Let n'' be the code for the tree that has the tree specified by n as its left subtree, and the tree specified by n' as its right subtree. Then $|n''| = 2(i+j)$. Let m'' be the binary string formed by concatenating the first $i-1$ bits of m onto the first $j-1$ bits of m'. The "1" between m'' and n'' will be ignored. The notation $\left\{\begin{matrix}0\\1\end{matrix}\right\}$ means either "0" or "1." Let $\alpha = |\rho(\mathbf{M}_1)| + |\rho(\mathbf{M}_2)| + \ldots + |\rho(\mathbf{M}_i)|$ and $\beta = |\rho(\mathbf{M'}_1)| + |\rho(\mathbf{M'}_2)| + \ldots + |\rho(\mathbf{M'}_j)|$. Then $d(\mathbf{x},\mathbf{y}) = \alpha + 4i + 1, d(\mathbf{y},\mathbf{z}) = \beta + 4j + 1$, and $d(\mathbf{x},\mathbf{z}) = \alpha + \beta + 4i + 4j + 1$ and we are done.

2. Topological Results

We will identify finite and infinite binary strings with points in the unit interval. Let $\mathbf{x}$ be an infinite binary string, and assume that $|\mathbf{y}| = \mathrm{N}$. Define:

$$\begin{aligned} \mathrm{B}_{\mathrm{K}}^{N,\epsilon}(\mathbf{x}) &= \{\mathbf{y} \mid d(\mathbf{x}_N, \mathbf{y}) < \epsilon N\}, \\ \mathrm{B}_{\mathrm{H}}^{N,\epsilon}(\mathbf{x}) &= \{\mathbf{y} \mid d_{\mathrm{H}}(\mathbf{x}_N, \mathbf{y}) < \epsilon N\}, \\ \mathrm{B}_{\mathrm{E}}^{\epsilon}(x) &= \{y \mid |x - y| < \epsilon\}. \end{aligned}$$

The first is the Kolmogorov distance, the second the Hamming distance, and the third the usual Euclidean distance. We prove the following:

$$\forall x, y \in [0,1]\ \forall \epsilon, \delta > 0\ \exists N_0 \forall N \geq N_0\ \exists \mathbf{z} \in \mathrm{B}_{\mathrm{K}}^{N,\epsilon}(\mathbf{x}) \text{ such that } \mathbf{z} \in \mathrm{B}_{\mathrm{E}}^{\delta}(y).$$

Intuitively, this says that the points in any small Kolmogorov ball are dense in the unit interval. Or, in other words, given any two points in the unit interval, we can find another point that has a numerical value very close to one point but a binary numeral very similar to the other. The proof is easy: given x and y, choose m so that $2^{-m} < \delta$. Let the first m digits of $\mathbf{z}$ match the first m digits of $\mathbf{y}$, and let the rest of $\mathbf{z}$ match the rest of $\mathbf{x}$. This works because the Euclidean distance weights the early bits very heavily. The same result holds between Hamming and Euclidean metrics.

However, the Hamming and Kolmogorov metrics do not share such a density property. Consider two unrelated maximally random infinite binary strings, i.e. $\forall N\ \mathrm{K}(\mathbf{x}_N \mid \mathbf{y}_N) \approx N$. Points in $\mathrm{B}_{\mathrm{K}}^{N,\epsilon}(\mathbf{x})$ are at Kolmogorov distance no greater than ϵN from $\mathbf{x}_N$, and points in $\mathrm{B}_{\mathrm{H}}^{N,\delta}(\mathbf{y})$ are at Kolmogorov distance no greater than $\mathcal{H}(\delta)N$ from $\mathbf{y}_N$. These will not in general add up to the N bits required by the assumption.

Next, we observe the differences in the number of points (strings of length N) in a ball in each of the metrics.

$$\begin{aligned} |\mathrm{B}_{\mathrm{K}}^{N,\epsilon}(\mathbf{x})| &\approx 2^{\epsilon N}, \\ |\mathrm{B}_{\mathrm{H}}^{N,\epsilon}(\mathbf{x})| &\approx 2^{\mathcal{H}(\epsilon)N}, \\ |\mathrm{B}_{\mathrm{E}}^{\epsilon}(x)| &\approx \epsilon 2^{N}. \end{aligned}$$

Finally, we note that standard methods (see, e.g. [2, p. 122]) allow the extension of bounded metrics on infinitely many finite sets to a new metric on the cartesian product of the original sets. Under a suitable choice of sets of finite strings, this induces an interesting topology (based on Kolmogorov complexity) on infinite binary strings.

3. Future Work

This novel metric is interesting for two reasons. First, it captures the algorithmic aspects of image similarity. Second, it may present a way to connect parts of computability theory to analysis and topology. There is a good chance that the well-developed tools of those mature areas will help find results in the younger and less well explored field of Kolmogorov complexity.

References

[1] Noga Alon. Expanders, sorting in rounds, and superconcentrators of limited depth. In *Proceedings of the 17th ACM STOC*, pages 98–102, 1985.

[2] J.-C. Bermond. Hamiltonian decompositions of graphs, directed graphs and hypergraphs. In Béla Bollobás, editor, *Advances in Graph Theory*, number 3 in Annals of Discrete Mathematics, pages 21–28. North-Holland, 1978.

[3] Thomas Beth, Dieter Jungnickel, and Hanfried Lenz. *Design Theory.* Cambridge University Press, 1986.

[4] Daniel Bienstock. Broadcasting with random faults. *Discrete Applied Mathematics*, 20(1):1–7, 1988.

[5] George Broomell and J. Robert Heath. Classification categories and historical development of circuit switching topologies. *Computing Surveys*, 15(2):95–133, June 1983.

[6] D. G. Cantor. On non-blocking switching networks. *Networks*, 1:367–377, 1972.

[7] K. W. Cattermole. Graph theory and communications networks. In Robin J. Wilson and Lowell W. Beineke, editors, *Applications of Graph Theory*, chapter 2, pages 17–57. Academic Press, 1979.

[8] Gregory J. Chaitin. A theory of program size formally identical to information theory. *Journal of the ACM*, 22:329–340, 1975.

[9] Gary Chartrand and Linda Lesniak. *Graphs and Digraphs.* Wadsworth & Brooks/Cole, second edition, 1986.

[10] F. R. K. Chung. On concentrators, superconcentrators, generalizers, and nonblocking networks. *Bell System Technical Journal*, 58(8):1765–1777, October 1978.

[11] Charles Clos. A study of non-blocking switching networks. *Bell System Technical Journal*, 32:406–424, March 1953.

[12] Jozsef Denes and A. D. Keedwell. *Latin Squares and Their Applications.* Academic Press, 1974.

[13] Howard Frank and Ivan T. Frisch. Analysis and design of survivable networks. *IEEE Transactions on Communication Technology*, COM-18(5):501–519, October 1970.

[14] Ofer Gabber and Zvi Galil. Explicit constructions of linear-sized superconcentrators. *Journal of Computer and System Sciences*, 22:407–420, 1981.

[15] Michael R. Garey, Frank K. Hwang, and Gaylord W. Richards. Asymptotic results for partial concentrators. *IEEE Transactions on Communications*, 36(2):214–217, February 1988.

[16] Sandra M. Hedetniemi, Stephen T. Hedetniemi, and Arthur L. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18:319–349, 1988.

[17] Joseph Y. Hui. *Switching and Traffic Theory for Integrated Broadband Networks.* Kluwer Academic Publishers, 1990.

[18] D. G. Kirkpatrick, M. Klawe, and N. Pippenger. Some graph-coloring theorems with applications to generalized connection networks. *SIAM Journal on Algebraic and Discrete Methods*, 6(4):576–582, 1985.

[19] Donald E. Knuth. *The TEXbook.* Addson-Wesley, 1984.

[20] Andrei N. Kolmogorov. Three approaches to the quantitative definition of information. *Problemy Peredachi Informatsii*, 1:3–11, 1965.

[21] Harry R. Lewis and Christos H. Papadimitriou. *Elements of the Theory of Computation.* Prentice-Hall, 1981.

[22] Arthur L. Liestman. Fault-tolerant broadcast graphs. *Networks*, 15:159–171, 1985.

[23] C. L. Liu. *Introduction to Combinatorial Mathematics.* McGraw-Hill, 1968.

[24] A. Lubotzky, R. Phillips, and P. Sarnak. Explicit expanders and the Ramanujan conjectures. In *Proceedings of the 18th STOC*, pages 240–246, 1986.

[25] A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.

[26] Gerald M. Masson. Binomial switching networks for concentration and distribution. *IEEE Transactions on Communications*, COM-25(9):873–883, September 1977.

[27] Gerald M. Masson, George C. Gingher, and Shinji Nakamura. A sampler of circuit switching networks. *Computer*, 12(6):32–47, June 1979.

[28] Gerald M. Masson and S. Brent Morris. Expected capacity of $\binom{m}{2}$-networks. *IEEE Transactions on Computers*, C-32(7):649–657, July 1983.

[29] Shinji Nakamura and Gerald M. Masson. Lower bounds on crosspoints in concentrators. *IEEE Transactions on Computers*, C-31(12):1173–1179, December 1982.

[30] Nicholas Pippenger. On the complexity of strictly nonblocking concentration networks. *IEEE Transactions on Communications*, COM-22:1890–1892, November 1974.

[31] Nicholas Pippenger. Complexity theory. *Scientific American*, pages 114–124, June 1978.

[32] Nicholas Pippenger. On rearrangeable and non-blocking switching networks. *Journal of Computer and System Sciences*, 17:145–162, 1978.

[33] Edward C. Posner. Crosspoint minimization. Lecture Notes for EE 183, Integrated Digital Communication; rev. F, April 1989.

[34] Damaraju Raghavarao. *Constructions and Combinatorial Problems in Design of Experiments.* Dover, 1971.

[35] Gaylord W. Richards and Frank K. Hwang. A two-stage rearrangeable broadcast switching network. *IEEE Transactions on Communications*, COM-33(10):1025–1035, October 1985.

[36] Herbert John Ryser. *Combinatorial Mathematics.* Number 14 in Carus Mathematical Monographs. Mathematical Association of America, 1963.

[37] David L. Schweizer. Some results on Kolmogorov–Chaitin complexity. Technical report 5233:TR:86, California Institute of Technology, 1986.

[38] R. Michael Tanner. Explicit concentrators from generalized n-gons. *SIAM Journal on Algebraic and Discrete Methods*, 5(3):287–293, September 1984.

[39] Clark D. Thompson. Fourier transforms in VLSI. *IEEE Transactions on Computers*, C-32(11):1047–1057, November 1983.

[40] A. K. Zvonkin and L. A. Levin. The complexity of finite objects and the development of the concepts of information and randomness by means of the theory of algorithms. *Russian Mathematical Surveys*, 25(6):83–124, 1970.