Precise coding with noiseless feedback

by

David Lawrence desJardins

S.B. (Massachusetts Institute of Technology), 1983

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Mathematics

in the

GRADUATE DIVISION
of the
UNIVERSITY of CALIFORNIA at BERKELEY

Spring 2002

Precise coding with noiseless feedback

Abstract

Precise coding with noiseless feedback

by

David Lawrence desJardins
Doctor of Philosophy in Mathematics

University of California at Berkeley

Professor Elwyn Berlekamp, Chair

The construction of block codes for communicating over discrete, noisy channels is a central problem of information theory. A block code is a set of codewords such that if the sender chooses one of the codewords to transmit, and no more than a specified number of errors occur, then the recipient can, with certainty, determine which codeword was sent.

If the recipient is also able to transmit information to the sender, instantly and without error, then the theoretical information capacity of the forward channel is not increased, but our ability to utilize that capacity with a coding scheme is enhanced. Assume a binary forward channel. The reverse channel can be used to transmit to the sender, after each bit is sent, the value of the bit as received by the recipient. The sender can then use that information in deciding what bit to send next; this is the idea of an adaptive block code.

We demonstrate that, for many choices of parameters, adaptive block codes may be constructed that are much better than standard block codes without feedback. For a wide range of parameters, we construct precisely optimal adaptive block codes, which contain the maximum possible number of codewords, given the length of the codes and the number of errors they tolerate.

The construction of adaptive block codes is equivalent to the formulation of strategies for a game described by Ulam, a version of "Twenty Questions" with lies. A corollary to our main result resolves Ulam's open question: given 1,000,000 objects, how many binary questions are required to identify a particular object, when the answerer is permitted to lie a given number of times? Our results are, in fact, much more precise: we show not only that one million objects may be distinguished with a certain number of questions, when a certain number of lies are allowed, but we determine the exact number (greater than one million) of objects that can be so distinguished, for each possible number of allowed lies.

Professor Elwyn Berlekamp
Dissertation Committee Chair

Contents

Acknowledgements

Since it's been almost 20 years since I first enrolled in the graduate division at UC Berkeley, I have a lot of people to thank for making it possible for me to finally complete my degree.

My advisor, Elwyn Berlekamp, for taking me on as a student, for getting me through my qualifying exam, for encouraging me to visit the Institute for Defense Analyses (which took time away from my studies in chronological terms, but left me much more prepared to do the research for this dissertation), for encouraging me to re-enroll at Berkeley, for helping me find a new problem that matched my skills, for discussions that helped me find my way to significant results, and for extreme patience with any number of delays and administrative issues in getting me through to completion.

My other committee members, David Eisenbud and Christos Papadimitriou, for promptly reviewing my draft, appreciating my work, and making very helpful suggestions on the exposition of the results.

The administrative staff at UC Berkeley, especially Janet Yonan and Thomas Brown, who were patient and helpful at every opportunity, even in dealing with problems of my own making.

Nick Patterson and David Lieberman, of the Institute for Defense Analyses, for giving me a position and fostering an environment that allowed me to greatly develop my mathematical skills.

George Soules and David Goldschmidt, of the Institute for Defense Analyses, for continuing that environment, and also for encouraging me to return to Berkeley to finish my degree, and especially for arranging for my support by IDA during the 1997–98 academic year.

Most especially, my wife, Nancy Blachman, for always supporting me in finishing this dissertation, for very patiently waiting for it to be done, and for taking on many burdens to give me the time I needed to complete it.

Chapter 1

Introduction

1.1 Introduction

In his autobiography [Ulam], Stanislaw Ulam proposes the following game, "a variation on the game of Twenty Questions":

> Someone thinks of a number between one and one million (which is just less than 2^{20}). Another person is allowed to ask up to twenty questions, to each of which the first person is supposed to answer only yes or no. Obviously the number can be guessed by asking first, "Is the number in the first half million?" then again reduce the reservoir of numbers in the next question by one-half, and so on. Finally the number is obtained in less than $\log_2(1,000,000)$. Now suppose one were allowed to lie once or twice, then how many questions would one need to get the right answer? One clearly needs more than n questions for guessing one of 2^n objects because one does not know when the lie was told. This problem is not solved in general.

Our main result is a general solution to this question, not only when the secret number is chosen from the set $\{1, 2, \ldots, 1000000\}$, but in general for a secret number chosen from $\{1, 2, \ldots, M\}$, for any $M \leq 2^{20} = 1048576$. Our results also strongly suggest a conjectural answer to the question for almost all values of M.

1.2 Background

Ulam's autobiography was first published in 1976. Ulam was apparently unaware that Berlekamp had considered the same problem in the 1960s, in the context of information theory. Berlekamp [Berlekamp], [Berlekamp2] studied the problem of sending data through a binary symmetric channel with noiseless feedback. As we will discuss below, if we consider an adversary who inserts at most a given number of errors into each transmission, then coding schemes for this problem are equivalent to play strategies for Ulam's game.

Berlekamp observed that the "volume bound" (our Theorem 3 of Section 2.5), well-known in the context of communication without feedback, would also apply to communication with feedback. Berlekamp also discovered two key concepts that we use in our work: the "translation bound" (our Theorem 8 of Section 2.10) and the "Fibonacci states" (our Theorem 9 of Section 2.12). Berlekamp applied these discoveries to the asymptotic analysis of coding with noiseless feedback. He showed, for many values of the *rate* (the ratio of the number of information bits sent through the channel to the total number of bits transmitted), that the maximum *error-correction fraction* (the ratio of the number of errors to the total number of bits transmitted) that could be reliably corrected would be asymptotically close to the upper bounds given by the volume and translation bounds, in the sense that the ratio of the maximum error-correction fraction to the upper bound implied by the volume and/or translation bounds approaches 1, as the length of the code goes to infinity with the rate fixed.

While this sort of asymptotic result is common in coding theory, it is far from being a precise solution to Ulam's question. For example, the asymptotic result stated above could be true even if Ulam's game consistently requires 10 more questions than would be implied by the volume bound (because $(N+10)/N \to 1$ as $N \to \infty$), or even if the gap between the number of questions required and the lower bound grows without limit (i.e., it could be that $N_{\text{actual}} - N_{\text{volume}} \to \infty$, even though $N_{\text{actual}}/N_{\text{volume}} \to 1$). Our goal is to *precisely* determine the exact number of questions required, for a broad range of parameters for Ulam's game.

In particular, we determine the number of questions needed in all cases where the number of objects, M, is less than or equal to 2^{20}. So, as a special case, our results answer Ulam's actual question, quoted above. See Table 1.1.

$E=0$	$E=1$	$E=2$	$E=3$	$E=4$	$E=5$	$E=6$	$E=7$	$E=8$	$E=9$	$E \geq 10$
20	25	29	33	36	40	43	46	49	53	$3E+26$

Table 1.1: Questions Needed for Ulam's Game

The most difficult of the cases in Table 1.1 appears to be $E = 8$, for which the result lies closest to the volume bound. Hill and Karim [Hill], [Hill2], [Karim] independently obtained the results in Table 1.1, for all cases except $E = 8$. Karim also obtains results for $M = 2^H$, when $H \leq 20$, and most values of E.

But our results are significantly more precise than just answering Ulam's question. For example, when 8 lies are permitted, we show that 49 questions suffice when the secret number is chosen from $\{1, 2, \ldots, 1017990\}$ (or if 1017990 is replaced by any smaller integer), but not for $\{1, 2, \ldots, 1017991\}$ (or if 1017991 is replaced by any larger integer).

Previously, the only cases for which such a precise result was known were those where E (the number of lies allowed, in Ulam's game) is less than or equal to 3. Complete analyses of winning strategies for Ulam's game, when the number of lies allowed is small, have been obtained by Pelc [Pelc] ($E = 1$), Guzicki [Guzicki] ($E = 2$), and Deppe [Deppe] ($E = 3$). However, the complexity of these analyses increases significantly as E grows.

Finally, Spencer [Spencer] obtained asymptotic results for any fixed value of E. However, his results have two limitations. First, they apply only for a "sufficiently large" number of questions, for a given number of lies. Spencer doesn't explicitly compute how large is "sufficiently large" for particular values of E, but it is clear from the form of his analysis that "sufficiently large" is very large indeed: much larger than any of the cases that we consider. Secondly, Spencer shows that when the number of questions is "sufficiently large", a necessary and sufficient criterion for success in Ulam's game is to be able to meet the volume bound after the first E questions asked. The problem is that it's not clear how difficult it will be to achieve that condition, for large values of E. So, while interesting and significant, Spencer's results don't seem to lead to precise results for any specific cases of Ulam's game.

1.3 Overview

In order to achieve precise results for Ulam's game, we need to obtain matching upper and lower bounds for the number of questions needed, given a specific range from which to choose the secret number, and a specific number of lies allowed.

Our lower bounds are the volume bound and translation bound previously described by Berlekamp, and refinements thereof. Specifically, in some cases we need to use "lookahead" to show that, while the volume or translation bound may be satisfied for the initial state of the game, any questioning strategy for the first one, two, or three questions will necessarily lead to a state that violates one of these bounds. These "lookahead" results are contained in Theorem 15 and Theorem 16 of Section 3.6, and Theorem 17 of Section 3.7.

Our upper bounds are obtained by computer search. That is, we use a computer algorithm to find complete strategies for winning Ulam's game with the number of questions determined by our lower bounds. The program that finds these strategies is given in Appendix A, a complete listing of the strategies it finds is presented in Appendix B, and a table of the results can be found in Table 4.1 of Section 4.1.

A search plan for finding such strategies is by no means obvious, and we need to develop several mathematical tools, which we use in our program. These tools are described in Chapter 3, and the plan for the program that uses these tools to construct the strategies is described in Chapter 4.

1.4 Conclusions and Speculations

In every case that we analyze, we are able to construct explicit strategies for Ulam's game that are very, very close to the limit implied by the volume bound, or which exactly meet the translation bound. The effect of our "lookahead" refinements to the volume bound is never to decrease the maximum number of values that can be distinguished, with a given number of questions and allowing a given number of lies, by more than one from the limit implied by the volume and translation bounds.

At high rates, when the volume bound is stronger than the translation bound, we always find that the maximum number of objects that can be distinguished, with a given number of questions and allowing a given number of lies, is no worse than one less than the maximum value that is consistent with the volume bound. And we conjecture that this may always be the case.

At low rates, when the translation bound is stronger than the volume bound, we always find that the translation bound is tight: i.e., that the maximum number of objects that can be distinguished with N questions when allowing E lies, is the same as the maximum number of objects that can be distinguished with $N-3$ questions when allowing $E-1$ lies. And we conjecture that this may always be the case.

Our data also support the conjecture that Spencer's asymptotic result for fixed E is in fact always valid, above the critical rate. That is, in the case where the volume bound is stronger than the translation bound, if it is possible to devise a questioning strategy such that after E questions all possible states that one might reach are consistent with the volume bound, then there is an actual winning strategy for Ulam's game. In other words, that a "lookahead" of E questions is enough. (In fact, in the cases we consider, we never need a lookahead of more than 3 questions. But we have no evidence that there are not states that require a larger lookahead. It seems very likely, though, that E questions are always enough.)

Chapter 2

Definitions and Previous Results

2.1 Block codes

Consider the problem of transmitting a message through a noisy channel. We will generally restrict our attention to the *binary symmetric channel*: the sender transmits one bit at a time, an element of $\{0, 1\}$, and the receiver receives that bit with probability $1 - g$, or the complementary bit with probability g, where g is in the interval $(0, 1/2)$. When the received bit differs from the transmitted bit, this is considered an *error*.

If $N > 0$ bits are transmitted through the binary symmetric channel, it is possible (albeit unlikely) that all of the bits are received in error. In order to make a deterministic analysis possible, coding theorists frequently consider the case where it is assumed that no more than E errors occur, for some E between 0 and N. If we fix an error-correction fraction $f > g$, then for sufficiently large N, we can say almost surely that the number of errors E will be less than fN. So schemes that correct a fixed number of errors are useful in practice.

A central problem of coding theory is the following: given that N bits are transmitted, and no more than E errors occur, what is the maximum number of distinct messages M that may be sent without any possibility of error on the part of the recipient? A construction that permits this is called a *block code*, the integer N is called the *length* of the code, and the integer E is called the *error-correction capacity.*

Definition 1 (Hamming weight). $H(x_1, \ldots, x_K)$ is the number of the x_i that are nonzero.

Definition 2 (Block code). A block code of length N and error-correcting capacity E is a sequence of functions

$$t_I : \{1, 2, \ldots, M\} \to \{0, 1\},$$

for $I = 1, 2, \ldots, N$, with the following equivalent properties:

- For any $m \neq m'$, there are at least $2E + 1$ values of I such that $t_I(m) \neq t_I(m')$.
- For any $m \neq m'$, and any binary tuples $(e_1, \ldots, e_N)$ and $(e'_1, \ldots, e'_N)$ with $H(e_1, \ldots, e_N) \leq E$ and $H(e'_1, \ldots, e'_N) \leq E$, there must be some I such that $t_I(m) \oplus e_I$ is not equal to $t_I(m') \oplus e'_I$.

To use the block code to transmit a message m, we send the sequence of bits

$$(t_1(m), \ldots, t_N(m)).$$

If an error pattern $(e_1, \ldots, e_N)$ occurs, the receiver receives the tuple

$$(t_1(m) \oplus e_1, \ldots, t_N(m) \oplus e_N).$$

The symbol $\oplus$ represents exclusive or: $0 \oplus 0 = 0$, $0 \oplus 1 = 1$, $1 \oplus 0 = 1$, and $1 \oplus 1 = 0$. So $x \oplus 0$ equals x, while $x \oplus 1$ is the complement of x. If $e_i = 0$, then there was no error in the transmission of the ith bit; the received value $t_i(m) \oplus e_i$ is equal to the transmitted value $t_i(m)$. If $e_i = 1$, there was an error; the received value $t_i(m) \oplus e_i$ is the complement of the transmitted value $t_i(m)$.

Under the hypothesis that $H(e_1, \ldots, e_N) \leq E$, there is no other message m' and error pattern $(e'_1, \ldots, e'_N)$ with $H(e'_1, \ldots, e'_N) \leq E$ that would cause the same tuple to be received. So the receiver can uniquely identify the message m, and the error pattern $(e_1, \ldots, e_N)$.

Definition 3 (Rate). The rate of a block code is

$$R = (\lg M)/N.$$

Definition 4 (Error-correction fraction). The error-correction fraction of a block code is

$$f = E/N.$$

2.2 Adaptive block codes

We are concerned with a variation of the above problem. Assume that after each bit is transmitted, the receiver has the opportunity to transmit an unlimited amount of information, instantly and without the possibility of error, to the sender. We can easily conclude that we cannot gain any advantage by sending through the back channel any more information than a single bit per received bit: the value of the received bit. This is the only information that the receiver has and the sender does not.

Shannon (1956) showed that this feedback cannot increase the theoretical capacity of the forward channel. Nevertheless, as we shall see, the feedback does prove useful at code rates below capacity. It can be used for more explicit encoding and decoding strategies, and at sufficiently low rates, can significantly improve the error-correction capability.

An *adaptive block code* has the same parameters N, E, and M as a standard block code, defined above. The sender agrees in advance with the receiver on a communication scheme. The sender then is given one of M messages to send. Based on that choice, the sender then transmits bits, one at a time. After each bit is sent, the sender learns whether an error occurred in sending that bit, and can use that information in deciding what bit to send next. After N bits are sent, the receiver must be able to determine which one of the M messages was chosen, under the assumption that no more than E errors occurred.

Definition 5 (Adaptive block code). An adaptive block code of length N and error-correcting capacity E is a sequence of functions

$$t_I : \{1, 2, \ldots, M\} \times \{0, 1\}^{I-1} \to \{0, 1\}$$

for $I = 1, 2, \ldots, N$. The functions t_I must have the property:

- For any $m \neq m'$ and any binary tuples $(e_1, \ldots, e_N)$ and $(e'_1, \ldots, e'_N)$ with $H(e_1, \ldots, e_N) \leq E$ and $H(e'_1, \ldots, e'_N) \leq E$, then there must be some I such that $t_I(m, e_1, \ldots, e_{I-1}) \oplus e_I$ is not equal to $t_I(m', e'_1, \ldots, e'_{I-1}) \oplus e'_I$.

The rate and error-correction fraction are defined as for a block code.

To use the adaptive block code to transmit a message m, we send the sequence of bits

$$(t_1(m), t_2(m, e_1), \ldots, t_N(m, e_1, \ldots, e_{N-1})),$$

where e_i is 1 if an error occurred in transmitting the i-th bit and 0 otherwise. Note that $t_I(m)$ depends only upon information available to the sender, from the feedback channel, after transmitting the first $I-1$ bits. If an error pattern $(e_1, \ldots, e_N)$ occurs, the receiver receives the tuple

$$(t_1(m) \oplus e_1, t_2(m, e_1) \oplus e_2, \ldots, t_N(m, e_1, \ldots, e_{N-1}) \oplus e_N).$$

Under the hypothesis that $H(e_1, \ldots, e_N) \leq E$, there is no other message m' and error pattern $(e'_1, \ldots, e'_N)$ with $H(e'_1, \ldots, e'_N) \leq E$ that would cause the same tuple to be received. So the receiver can uniquely identify the message m, and the error pattern $(e_1, \ldots, e_N)$.

2.3 Ulam's game

Ulam's game is described in Section 1.1. We can formalize Ulam's game as follows.

Game 1 (Ulam's game). Alice secretly chooses a number from $\{1, 2, \ldots, M\}$. Bob then asks questions, one at a time, each of which is a subset of $\{1, 2, \ldots, M\}$. Alice answers each question with 1 (the secret number is in the set) or 0 (the secret number is not in the set). Bob is allowed N questions, and Alice is constrained that at most E of her answers may be false. Bob's goal is a questioning strategy such that, after the N questions, he has uniquely identified the secret number.

It turns out that the existence of winning strategies for Ulam's game is exactly equivalent to the existence of adaptive block codes.

Theorem 1. *Bob has a winning strategy for Ulam's game with parameters N, E, M if and only if there exists an adaptive block code with the same parameters.*

Proof. First, suppose that we are given an adaptive block code; we use this to generate a winning strategy for Ulam's game. Bob's first question is the set $\{m|t_1(m) = 1\}$, to which Bob receives the reply r_1. Bob can then compute the function $e_1(m) = r_1 \oplus t_1(m)$, which indicates whether Alice answered the first question falsely under the hypothesis that the secret value is m. Bob's second question is the set $\{m|t_2(m, e_1(m)) = 1\}$, to which Bob receives the reply r_2. Bob can then compute the function $e_2(m) = r_2 \oplus t_2(m, e_1(m))$, which indicates whether Alice answered the second question falsely under the hypothesis that the secret value is m. Bob's third question is the set $\{m|t_3(m, e_1(m), e_2(m)) = 1\}$, and so on.

After N questions, Bob can compute for each m the tuple $(e_1(m), \ldots, e_N(m))$. By the constraint that Alice may answer falsely at most E times, we know that for the actual secret m, $H(e_1(m), \ldots, e_N(m)) \leq E$. Suppose that there is some other m' such that $H(e_1(m'), \ldots, e_N(m')) \leq E$. Then, by the definition of the adaptive block code, there must be some I such that $t_I(m, e_1(m), \ldots, e_{I-1}(m)) \oplus e_I(m) \neq t_I(m', e_1(m'), \ldots, e_{I-1}(m')) \oplus e_I(m')$. But, by the definition of the function e_I, both sides of this inequality are equal to r_I.

Second, suppose that we are given a winning strategy for Ulam's game; we use this to construct an adaptive block code. Let S_1 be Bob's first question. We define $t_1(m)$ to be 1 if $m \in S_1$, 0 otherwise. Let $S_2(r_1)$ be Bob's second question, given that the response to the first question was r_1. Define $t_2(m, e_1)$ to be 1 if $m \in S_2(e_1 \oplus t_1(m))$, 0 otherwise. Let $S_3(r_1, r_2)$ be Bob's third question, given the first two responses. Define $t_3(m, e_1, e_2)$ to be 1 if $m \in S_3(e_1 \oplus t_1(m), e_2 \oplus t_2(m, e_1))$, 0 otherwise. And so on.

Consider any $m \neq m'$, and $(e_1, \ldots, e_N)$ and $(e'_1, \ldots, e'_N)$, with $H(e_1, \ldots, e_N) \leq E$ and $H(e'_1, \ldots, e'_N) \leq E$. Suppose that $e_I \oplus t_I(m, e_1 \oplus t_1(m), \ldots, e_{I-1} \oplus t_{I-1}(m))$ equals $e'_I \oplus t_I(m', e'_1 \oplus t_1(m'), \ldots, e'_{I-1} \oplus t_{I-1}(m'))$ for every $I = 1, 2, \ldots, N$. Then if Alice chooses m as the secret number, and gives false answers for those values of I such that $e_I = 1$, then Bob will ask exactly the same questions and receive exactly the same answers as if Alice chooses m' as the secret number, and gives false answers for those values of I such that $e'_I = 1$. So, by the assumption that Bob's questioning strategy is valid, no such m and m' can exist; this proves that the functions t_I form an adaptive block code. □

Note that if we modify Ulam's game so that Alice does not have to fix her secret choice at the beginning of the game, but only must answer all of the questions in such a way that they are consistent with some possible secret, then the question of existence of a winning strategy for Bob is unchanged. If Bob has a strategy that always wins when Alice chooses her secret in advance, then the same strategy must always win even if Alice is not forced to commit to a particular secret at the start.

2.4 State-space formulation

In this section we present another equivalent formulation of the construction of adaptive block codes, or questioning strategies for Ulam's game. Given integers N, E, and M, we will consider a game played with finite tuples of nonnegative integers. We will delete trailing zeroes from the tuples, so that the $(E+1)$-tuple $(s_0, s_1, \ldots, s_E)$ is equivalent to the E-tuple $(s_0, s_1, \ldots, s_{E-1})$ if $s_E = 0$.

Definition 6 (Translation operator). We define an operator $\mathbf{T}$ on tuples of nonnegative integers. (The letter $\mathbf{T}$ denotes "translation".) Given the $(E+1)$-tuple $\mathbf{s} = (s_0, s_1, \ldots, s_E)$,

$$\mathbf{Ts} = (s_1, s_2, \ldots, s_E, 0) = (s_1, s_2, \ldots, s_E).$$

Game 2 (Tuple-game). The tuple-game has N rounds, and begins with an $(E+1)$-tuple of natural numbers, $\mathbf{s}_0 \in \mathbb{N}^{E+1}$. First, Bob partitions $\mathbf{s}_0$ into $\mathbf{a}_0, \mathbf{b}_0 \in \mathbb{N}^{E+1}$ such that $\mathbf{s}_0 = \mathbf{a}_0 + \mathbf{b}_0$. Then, Alice chooses either $\mathbf{Ta}_0 + \mathbf{b}_0$ or $\mathbf{a}_0 + \mathbf{Tb}_0$, and her choice becomes $\mathbf{s}_1$. Then the process is repeated: Bob chooses $\mathbf{a}_1, \mathbf{b}_1 \in \mathbb{N}^{E+1}$ such that $\mathbf{s}_1 = \mathbf{a}_1 + \mathbf{b}_1$, and Alice chooses either $\mathbf{Ta}_1 + \mathbf{b}_1$ or $\mathbf{a}_1 + \mathbf{Tb}_1$ to be $\mathbf{s}_2$. And so on. At the end of N rounds, Bob wins the game if the sum of the elements of $\mathbf{s}_N$ is 0 or 1; Alice wins otherwise.

Example: Let $N = 6$, $E = 2$, and $\mathbf{s}_0 = (14, 4, 1)$. Bob chooses $\mathbf{a}_0 = (7, 1, 1)$ and $\mathbf{b}_0 = (7, 3)$, so $\mathbf{s}_0 = \mathbf{a}_0 + \mathbf{b}_0$. (As later results will show, this is Bob's only winning move.) Alice then chooses $\mathbf{T}\mathbf{a}_0 + \mathbf{b}_0 = (8, 4)$ or $\mathbf{a}_0 + \mathbf{T}\mathbf{b}_0 = (10, 1, 1)$ to be $\mathbf{s}_1$, and so on.

Theorem 2. *Let $\mathbf{s}_0$ be the $(E+1)$-tuple $(0, 0, \ldots, 0, M)$. Bob has a winning strategy for the tuple-game with N rounds and initial state $\mathbf{s}_0$ if and only if Bob has a winning strategy for Ulam's game with parameters N, E, M.*

Proof. First, suppose that we are given a winning strategy for the tuple-game; we use this to generate a winning strategy for Ulam's game.

Suppose that I questions $S_1, S_2(r_1), \ldots, S_I(r_1, \ldots, r_{I-1})$ have been asked in Ulam's game, with answers $(r_1, \ldots, r_I)$. For any $m \in \{1, 2, \ldots, M\}$, we can compute: $e_1(m) = r_1 \oplus t_1(m)$, where $t_1(m)$ is 1 if $m \in S_1$, 0 otherwise; $e_2(m) = r_2 \oplus t_2(m, e_1(m))$, where $t_2(m, e_1(m))$ is 1 if $m \in S_2(r_1)$, 0 otherwise; and so on to $e_I(m) = r_I \oplus t_I(m, e_1(m), \ldots, e_{I-1}(m))$, where $t_I(m, e_1(m), \ldots, e_{I-1}(m))$ is 1 if $m \in S_I(r_1, \ldots, r_{I-1})$, 0 otherwise. We can then define $E_I(m) = E - H(e_1(m), \ldots, e_I(m))$. A winning strategy for Ulam's game is one which has, for every sequence $(r_1, \ldots, r_N)$, $E_N(m) \geq 0$ for at most one value of m.

Given a partial sequence of received bits $(r_1, \ldots, r_I)$, we can define the state $\mathbf{s}_I \in \mathbb{N}^{E+1}$ to be the tuple where the J-th coordinate $\mathbf{s}_I(J)$ is the size of the set $\{m | E_I(m) = J\}$, for $J = 0, 1, \ldots, E$. Note that $\mathbf{s}_0 = (0, 0, \ldots, 0, M)$, as we specified for the tuple-game.

Given a partition of $\mathbf{s}_I$ into $\mathbf{a}_I + \mathbf{b}_I$, we can choose a partition of $\{1, 2, \ldots, M\}$ into disjoint subsets A_I and B_I, such that the size of $\{m \in A | E_I(m) = J\}$ is exactly the J coordinate of $\mathbf{a}_I$. (For example, we can choose A to contain the smallest $\mathbf{a}_I(J)$ of the total $\mathbf{s}_I(J)$ such messages. The elements of $\{1, 2, \ldots, M\}$ with $E_I(m) < 0$ can go into either A or B.)

Our strategy for the tuple-game gives us such a partition of $\mathbf{s}_I$ into $\mathbf{a}_I + \mathbf{b}_I$. Choose a set A_I, as above; this becomes the question S_{I+1} for Ulam's game. If the answer $r_{I+1} = 1$ is received, then $E_{I+1}(m) = E_I(m)$ for $m \in A_I$ and $E_{I+1}(m) = E_I(m) - 1$ for $m \in B_I$, which implies that $\mathbf{s}_{I+1}$, defined as above, is equal to $\mathbf{a}_I + \mathbf{T}\mathbf{b}_I$. While, if the answer $r_{I+1} = 0$ is received, then $E_{I+1}(m) = E_I(m) - 1$ for $m \in A_I$ and $E_{I+1}(m) = E_I(m)$ for $m \in B_I$, which implies that $\mathbf{s}_{I+1}$, defined as above, is equal to $\mathbf{Ta}_I + \mathbf{b}_I$. Thus, regardless of whether Alice answers 0 or 1 in Ulam's game, the new state $\mathbf{s}_{I+1}$ corresponds to one of Alice's possible choices in the tuple-game. So, after N such questions, if our strategy for the tuple-game is successful, the sum of the elements of $\mathbf{s}_N$ will be zero or one, which means that there is at most one $m \in \{1, 2, \ldots, M\}$ with $E_N(m) \geq 0$, which means that this was a winning strategy for Ulam's game.

Second, suppose that we are given a winning strategy for Ulam's game; we use this to generate a winning strategy for the tuple-game. We can use the $(I+1)$-st question $S_{I+1}(r_1, \ldots, r_I)$ in Ulam's game to define a partition of the state $\mathbf{s}_I$ into $\mathbf{a}_I + \mathbf{b}_I$, by letting $\mathbf{a}_I(J)$ be the number of elements $m \in S_{I+1}$ with $E_I(m) = J$. We can then interpret Alice's choice of $\mathbf{Ta}_1 + \mathbf{b}_1$ or $\mathbf{a}_1 + \mathbf{Tb}_1$ in the tuple-game as a choice of $r_{I+1} = 0$ or 1 respectively in Ulam's game, use that new state of Ulam's game to determine a new question for Ulam's game, and so on. If our strategy for Ulam's game is successful, then after N questions and answers, we will have at most one $m \in \{1, 2, \ldots, M\}$ with $E_N(m) \geq 0$, which implies that the state vector $\mathbf{s}_N$ will sum to zero or one, which is the winning condition for the tuple-game. $\square$

Following Berlekamp, we will call such an $(E+1)$-tuple a *state*.

Definition 7 (Winning N-state). The state $\mathbf{s}$ is a *winning N-state* if Bob has a winning strategy for the tuple-game of N rounds with initial state $\mathbf{s}$.

Corollary 1. *The following are all equivalent:*

- *The $(E+1)$-tuple $(0, 0, \ldots, 0, M)$ is a winning N-state.*
- *Bob has a winning strategy for Ulam's game with parameters N, E, M.*
- *There exists an adaptive block code with parameters N, E, M.*

Proof. Immediate from Theorem 1 and Theorem 2. $\square$

Lemma 1. *If $\mathbf{s}$ is a winning N-state, $\mathbf{s}$ is also a winning N'-state for any $N' > N$.*

Proof. In subsequent rounds of the tuple-game, the sum of the elements of $\mathbf{s}_I$ is nonincreasing, because the sum of the elements of $\mathbf{Ta}$ is at most the sum of the elements of $\mathbf{a}$, and the sum of elements of $\mathbf{Tb}$ is at most the sum of the elements of $\mathbf{b}$. So, if Bob has a strategy that forces the sum of the elements of $\mathbf{s}_N$ to be at most 1, then Bob can force the sum of the elements of $\mathbf{s}_{N'}$ to be at most 1, for any $N' > N$. $\square$

2.5 Volume bound

Definition 8 (Notation). From here on, we will avoid using subscripts to index states, so we can write s_J for $\mathbf{s}(J)$; i.e., the Jth element of the tuple $\mathbf{s}$.

Definition 9 (Partial sum of state). Given a state $\mathbf{s} = (s_0, s_1, \ldots, s_E)$, we define

$$s^{(J)} = \sum_{J'=J}^{E} s_{J'}.$$

We also define $s^{(J)} = 0$ for $J > E$.

Lemma 2 (Partial sum representation). *The mapping*

$$(s_0, s_1, \ldots, s_E) \leftrightarrow (s^{(0)}, s^{(1)}, \ldots, s^{(E)})$$

is a one-to-one correspondence between $(E+1)$-tuples of nonnegative integers, and nonincreasing tuples of nonnegative integers.

Proof. If $(s_0, s_1, \ldots, s_E)$ is nonnegative, then $s^{(J)}$ is certainly nonnegative, and $s^{(J)} = s^{(J+1)} + s_J$, so $s^{(J)} \geq s^{(J+1)}$. Conversely, if $(s^{(0)}, s^{(1)}, \ldots, s^{(E)})$ is nonnegative and nonincreasing, then $s_E = s^{(E)}$ is nonnegative, and $s_J = s^{(J)} - s^{(J+1)}$ is nonnegative for $J < E$. $\square$

Given this lemma, we will use the two representations of a state $\mathbf{s}$ interchangeably. The following terminology will be useful:

Definition 10 (Singlet). A state $\mathbf{s}$ with $s^{(0)} = 1$ is called a *singlet*.

Definition 11 (Doublet). A state $\mathbf{s}$ with $s^{(0)} = 2$ is called a *doublet*.

By definition, every singlet is a winning 0-state, and thus a winning N-state for every N.

Lemma 3. *If* $\mathbf{s}$ *is a winning N-state and* $\mathbf{s}$ *is not a singlet, then $s_J = 0$ for $J \geq N$.*

Proof. Since $\mathbf{s}$ is not a singlet, N is greater than 0. Since $\mathbf{s}$ is a winning N-state, $\mathbf{s} = \mathbf{a}+\mathbf{b}$ where $\mathbf{Ta}+\mathbf{b}$ and $\mathbf{a}+\mathbf{Tb}$ are winning $(N-1)$-states. If $\mathbf{Ta}+\mathbf{b}$ is a singlet, and $\mathbf{a}+\mathbf{Tb}$ is a singlet, then $a_J = b_J = 0$ for all $J > 0$. But if either $\mathbf{Ta}+\mathbf{b}$ or $\mathbf{a}+\mathbf{Tb}$ is not a singlet, then by induction $a_J = b_J = 0$ for $J \geq N$, so $s_J = 0$ for $J \geq N$. $\square$

Following Berlekamp's terminology, we now define the *volume* of an N-state:

Definition 12 (Volume).

$$\mathrm{Vol}_N(\mathbf{s}) = \sum_{J=0}^{E} s_J \sum_{J'=0}^{J} \binom{N}{J'} = \sum_{J=0}^{E} s^{(J)} \binom{N}{J}.$$

When N is not clear from the context, we will refer to Vol_N as the *N-volume*. Note that throughout this text we take $\binom{N}{J} = 0$ when $N < J$ or $J < 0$.

Given this definition, we can easily prove the following lemma and theorem.

Lemma 4 (Conservation of Volume). *If* $\mathbf{s} = \mathbf{a}+\mathbf{b}$*, then* $\mathrm{Vol}_N(\mathbf{s}) = \mathrm{Vol}_{N-1}(\mathbf{Ta}+\mathbf{b}) + \mathrm{Vol}_{N-1}(\mathbf{a}+\mathbf{Tb})$.

Proof. Since $\binom{N}{J} = \binom{N-1}{J} + \binom{N-1}{J-1}$, $\mathrm{Vol}_N(\mathbf{s}) = \mathrm{Vol}_{N-1}(\mathbf{s}) + \mathrm{Vol}_{N-1}(\mathbf{Ts})$ for any state $\mathbf{s}$. So $\mathrm{Vol}_N(\mathbf{s}) = \mathrm{Vol}_N(\mathbf{a}+\mathbf{b}) = \mathrm{Vol}_N(\mathbf{a}) + \mathrm{Vol}_N(\mathbf{b}) = \mathrm{Vol}_{N-1}(\mathbf{a}) + \mathrm{Vol}_{N-1}(\mathbf{Ta}) + \mathrm{Vol}_{N-1}(\mathbf{b}) + \mathrm{Vol}_{N-1}(\mathbf{Tb}) = \mathrm{Vol}_{N-1}(\mathbf{Ta}+\mathbf{b}) + \mathrm{Vol}_{N-1}(\mathbf{a}+\mathbf{Tb})$. $\square$

Theorem 3 (Volume Bound). *If* $\mathbf{s}$ *is a winning N-state, then* $\mathrm{Vol}_N(\mathbf{s}) \leq 2^N$.

Proof. If $N = 0$ then the result is trivial. Otherwise, proceed by induction on N. If $\mathbf{s}$ is a winning N-state, then there exist states $\mathbf{a}$, $\mathbf{b}$ such that $\mathbf{s} = \mathbf{a}+\mathbf{b}$ and $\mathbf{Ta}+\mathbf{b}$, $\mathbf{a}+\mathbf{Tb}$ are winning $(N-1)$-states. So, by the induction hypothesis, $\mathrm{Vol}_{N-1}(\mathbf{Ta}+\mathbf{b}) \leq 2^{N-1}$ and $\mathrm{Vol}_{N-1}(\mathbf{a}+\mathbf{Tb}) \leq 2^{N-1}$. But, using conservation of volume, $\mathrm{Vol}_N(\mathbf{s}) = \mathrm{Vol}_{N-1}(\mathbf{Ta}+\mathbf{b}) + \mathrm{Vol}_{N-1}(\mathbf{a}+\mathbf{Tb}) \leq 2^{N-1} + 2^{N-1} = 2^N$. $\square$

2.6 Borderline states

Definition 13 (Borderline winning N-state). A winning N-state $\mathbf{s}$ is a *borderline* winning N-state if $\mathrm{Vol}_N(\mathbf{s}) = 2^N$; that is, if it satisfies the volume bound with equality.

Theorem 4. *If $\mathbf{s}$ is a borderline winning N-state, and $\mathbf{s}$ is not a singlet, then $\mathbf{s}$ is not a winning $(N-1)$-state.*

Proof. Since $\mathrm{Vol}_N(\mathbf{s}) = 2^N > 0$, $s_J > 0$ for some J. By Lemma 3, $s_J > 0$ for some $J < N$. Thus we compute:

$$\begin{aligned}
2^N = \mathrm{Vol}_N(\mathbf{s}) &= \sum_{J=0}^{E} s^{(J)} \binom{N}{J} \\
&= \sum_{J=0}^{E} s^{(J)} \binom{N-1}{J} + \sum_{J=0}^{E} s^{(J)} \binom{N-1}{J-1} \\
&= \sum_{J=0}^{E} s^{(J)} \binom{N-1}{J} + \sum_{J=0}^{E-1} s^{(J+1)} \binom{N-1}{J} \\
&= \sum_{J=0}^{E} (2s^{(J)} - s_J) \binom{N-1}{J} \\
&= 2\,\mathrm{Vol}_{N-1}(\mathbf{s}) - \sum_{J=0}^{E} s_J \binom{N-1}{J} \\
&< 2\,\mathrm{Vol}_{N-1}(\mathbf{s})
\end{aligned}$$

since the sum on the next to last line is positive. So $\mathrm{Vol}_{N-1}(\mathbf{s}) > 2^{N-1}$, so $\mathbf{s}$ cannot be a winning $(N-1)$-state. $\square$

Lemma 5. *Given a winning N-state $\mathbf{s} = (s_0, s_1, \ldots, s_E)$, there is a borderline winning N-state $\mathbf{s}' = (s_0', s_1, \ldots, s_E)$ which differs from $\mathbf{s}$ only in the 0th coordinate.*

Proof. When $N = 0$ this is clear, because a winning 0-state is either a singlet, in which case it already meets the volume bound with equality, or is zero, in which case $\mathbf{s}' = (1)$ satisfies the conclusion.

Otherwise, proceed by induction on N. Given a winning N-state $\mathbf{s}$, there is a partition $\mathbf{s} = \mathbf{a} + \mathbf{b}$ such that $\mathbf{c} = \mathbf{Ta} + \mathbf{b}$ and $\mathbf{d} = \mathbf{a} + \mathbf{Tb}$ are winning $(N-1)$-states. By the induction hypothesis, there are borderline winning $(N-1)$-states $\mathbf{c}'$,

$\mathbf{d}'$ which differ from $\mathbf{c}$, $\mathbf{d}$ respectively only in the 0th coordinate. Let $\mathbf{b}' = \mathbf{c}' - \mathbf{Ta}$, $\mathbf{a}' = \mathbf{d}' - \mathbf{Tb}$, and $\mathbf{s}' = \mathbf{a}' + \mathbf{b}'$. Then $\mathbf{s}' - \mathbf{s} = (\mathbf{a}' - \mathbf{a}) + (\mathbf{b}' - \mathbf{b}) = (\mathbf{c}' - \mathbf{c}) + (\mathbf{d}' - \mathbf{d})$, so $\mathbf{s}'$ differs from $\mathbf{s}$ only in the 0th coordinate.

But $\mathbf{c}'$, $\mathbf{d}'$ are borderline winning $(N-1)$-states, so $\mathrm{Vol}_{N-1}(\mathbf{c}') = \mathrm{Vol}_{N-1}(\mathbf{d}') = 2^{N-1}$. By conservation of volume, $\mathrm{Vol}_N(\mathbf{s}') = \mathrm{Vol}_{N-1}(\mathbf{c}') + \mathrm{Vol}_{N-1}(\mathbf{d}') = 2^{N-1} + 2^{N-1} = 2^N$. So $\mathbf{s}'$ is a borderline winning N-state. □

2.7 Doublets

Theorem 5 (Doublet theorem). *If* $\mathbf{s}$ *is a doublet, and*

$$\mathbf{s} = (0, 0, \ldots, 0, 1, 0, \ldots, 0) + (0, 0, \ldots, 0, 1, 0, \ldots, 0),$$

where the 1's occur in positions J and K, then $\mathbf{s}$ *is a borderline winning $(J+K+1)$-state, and not a winning $(J+K)$-state.*

Proof. By induction. Write $\mathbf{s} = \mathbf{a} + \mathbf{b}$, where $\mathbf{a}$ and $\mathbf{b}$ are singlets with $a_J = b_K = 1$. Then each of $\mathbf{Ta} + \mathbf{b}$ and $\mathbf{a} + \mathbf{Tb}$ is either a singlet, or is a doublet that by induction is a winning $(J+K)$-state. So $\mathbf{s}$ is a winning $(J+K+1)$-state.

Let $N = J + K + 1$. Then

$$\begin{aligned}
\mathrm{Vol}_N(\mathbf{s}) &= \sum_{J'=0}^{J} \binom{N}{J'} + \sum_{J'=0}^{K} \binom{N}{J'} \\
&= \sum_{J'=0}^{J} \binom{N}{J'} + \sum_{J'=0}^{K} \binom{N}{N-J'} \\
&= \sum_{J'=0}^{J} \binom{N}{J'} + \sum_{J'=N-K}^{N} \binom{N}{J'} \\
&= \sum_{J'=0}^{N} \binom{N}{J'} \\
&= 2^N,
\end{aligned}$$

so $\mathbf{s}$ is a borderline winning $(J+K+1)$-state. □

2.8 Zero-error states

The following result corresponds to the trivial strategy for the game of "Twenty Questions".

Theorem 6 (Twenty questions without lies). *If $E = 0$, so $\mathbf{s} = (s_0)$, then $\mathbf{s}$ is a winning N-state if and only if $s_0 \leq 2^N$.*

Proof. $\mathrm{Vol}_N(\mathbf{s}) = s_0$, so "only if" follows from the volume bound. For the "if" direction, if $N = 0$, then $\mathbf{s} = (1)$ or $\mathbf{s} = (0)$ which are winning 0-states. If $N > 0$, then we can write $\mathbf{s} = \mathbf{a}+\mathbf{b}$ where $\mathbf{a} = (a_0)$, $\mathbf{b} = (b_0)$, and $a_0, b_0 \leq 2^{N-1}$. So $\mathbf{Ta}+\mathbf{b} = \mathbf{b}$ and $\mathbf{a} + \mathbf{Tb} = \mathbf{a}$, and by induction both $\mathbf{a}$ and $\mathbf{b}$ are winning $(N-1)$-states, so $\mathbf{s}$ is a winning N-state. $\square$

2.9 Dominated states

Definition 14 (Domination). We say that $\mathbf{t}$ *dominates* $\mathbf{s}$ if (and only if)

$$s^{(J)} \leq t^{(J)} \text{ for all } J = 0, 1, \dots, E.$$

We will use the notation $\mathbf{t} \geq \mathbf{s}$ for this relation.

Theorem 7 (Domination theorem). *If $\mathbf{t}$ dominates $\mathbf{s}$, and $\mathbf{t}$ is a winning N-state, then $\mathbf{s}$ is a winning N-state.*

Proof. By induction on N. The statement is trivial when $N = 0$. Otherwise, since $\mathbf{t}$ is a winning N-state, $\mathbf{t} = \mathbf{c}+\mathbf{d}$, where $\mathbf{Tc}+\mathbf{d}$ and $\mathbf{c}+\mathbf{Td}$ are winning $(N-1)$-states. And $s^{(J)} \leq t^{(J)} = c^{(J)} + d^{(J)}$ for each J. If we find $\mathbf{a}$, $\mathbf{b}$ such that $\mathbf{s} = \mathbf{a} + \mathbf{b}$, $\mathbf{a} \leq \mathbf{c}$, and $\mathbf{b} \leq \mathbf{d}$, then we are done, because $\mathbf{Ta} + \mathbf{b} \leq \mathbf{Tc} + \mathbf{d}$ and $\mathbf{a} + \mathbf{Tb} \leq \mathbf{c} + \mathbf{Td}$, so $\mathbf{Ta} + \mathbf{b}$ and $\mathbf{a} + \mathbf{Tb}$ are winning $(N-1)$-states by induction.

The tuples $(a^{(0)}, \dots, a^{(E)})$ and $(b^{(0)}, \dots, b^{(E)})$ must satisfy $s^{(J)} = a^{(J)} + b^{(J)}$ for each J, and the constraint that $a^{(J)} \geq a^{(J+1)}$ and $b^{(J)} \geq b^{(J+1)}$ for each J. We will construct such a state $\mathbf{a}$, and the corresponding state $\mathbf{b}$, by choosing $a^{(E)}$, then $a^{(E-1)}$, and so on.

For each $J = 0, 1, \ldots, E$, we need to choose $a^{(J)}$, and $b^{(J)} = s^{(J)} - a^{(J)}$, such that all of the following inequalities are simultaneously satisfied:

$$\begin{aligned} a^{(J+1)} \leq a^{(J)} &\quad\Longleftrightarrow\quad a^{(J)} \geq a^{(J+1)} \\ b^{(J+1)} \leq b^{(J)} &\quad\Longleftrightarrow\quad a^{(J)} \leq s^{(J)} - b^{(J+1)} \\ a^{(J)} \leq c^{(J)} &\quad\Longleftrightarrow\quad a^{(J)} \leq c^{(J)} \\ b^{(J)} \leq d^{(J)} &\quad\Longleftrightarrow\quad a^{(J)} \geq s^{(J)} - d^{(J)}. \end{aligned}$$

(Recall that $a^{(E+1)} = b^{(E+1)} = 0$ by definition.)

Once a valid choice of $a^{(J+1)}$ has been made, there exists at least one value for $a^{(J)}$ that simultaneously satisfies all of these inequalities, because:

$$\begin{aligned} a^{(J+1)} + b^{(J+1)} = s^{(J+1)} \leq s^{(J)} &\quad\Longrightarrow\quad a^{(J+1)} \leq s^{(J)} - b^{(J+1)} \\ a^{(J+1)} \leq c^{(J+1)} \leq c^{(J)} &\quad\Longrightarrow\quad a^{(J+1)} \leq c^{(J)} \\ b^{(J+1)} \leq d^{(J+1)} \leq d^{(J)} &\quad\Longrightarrow\quad s^{(J)} - d^{(J)} \leq s^{(J)} - b^{(J+1)} \\ s^{(J)} \leq t^{(J)} = c^{(J)} + d^{(J)} &\quad\Longrightarrow\quad s^{(J)} - d^{(J)} \leq c^{(J)}. \end{aligned}$$

So the sequence of choices is always possible, so such an $\mathbf{a}$, $\mathbf{b}$ exist. □

2.10 Translation bound

Berlekamp discovered the following important result.

Theorem 8 (Translation bound). *If* $\mathbf{s}$ *is a winning* N*-state with* $N \geq 3$*, and* $s^{(0)} \geq 3$*, then* $\mathbf{Ts}$ *is a winning* $(N-3)$*-state.*

Proof. When $N = 3$, then any counterexample must have $s^{(1)} \geq 2$; otherwise $\mathbf{Ts}$ would be a winning 0-state. But, together with the hypothesis, this implies that $\mathrm{Vol}_3(\mathbf{s}) \geq 3\binom{3}{0} + 2\binom{3}{1} = 3 \cdot 1 + 2 \cdot 3 = 9$, which contradicts the volume bound, so $\mathbf{s}$ cannot be a winning 3-state.

If $N > 3$, proceed by induction. Write $\mathbf{s} = \mathbf{a} + \mathbf{b}$, where $\mathbf{Ta} + \mathbf{b}$ and $\mathbf{a} + \mathbf{Tb}$ are winning $(N-1)$-states. If $a^{(1)} + b^{(0)} \geq 3$ and $a^{(0)} + b^{(1)} \geq 3$, then by the induction hypothesis $\mathbf{TTa} + \mathbf{Tb}$ and $\mathbf{Ta} + \mathbf{TTb}$ are winning $(N-4)$-states, so the partition $\mathbf{Ts} = \mathbf{Ta} + \mathbf{Tb}$ implies that $\mathbf{Ts}$ is a winning $(N-3)$-state.

Otherwise, without loss of generality assume that $a^{(0)} + b^{(1)} \leq 2$. Then $s^{(1)} = a^{(1)} + b^{(1)} \leq a^{(0)} + b^{(1)} \leq 2$. If $s^{(1)} = 1$, then $\mathbf{Ts}$ is a winning 0-state, so trivially a winning $(N-3)$-state. If $s^{(1)} = 2$, let $\mathbf{s}'$ be the state equal to $\mathbf{s}$ except that $s'_0 = 0$. Then $\mathbf{s}'$ is a winning N-state, since $\mathbf{s}' \leq \mathbf{s}$. But $\mathbf{s}'$ is a doublet, so it is either a borderline winning N-state, or it is a winning $(N-1)$-state. If $\mathbf{s}'$ is a borderline winning N-state, then $2^N = \mathrm{Vol}_N(\mathbf{s}') \leq \mathrm{Vol}_N(\mathbf{s}) \leq 2^N$, so $\mathrm{Vol}_N(s) = \mathrm{Vol}_N(s')$, so $s_0 = 0$, so $\mathbf{s}$ is a doublet, contradicting the hypothesis. On the other hand, if $\mathbf{s}'$ is a winning $(N-1)$-state, then $\mathbf{Ts}'$ is a winning $(N-3)$-state, by the formula for doublets, and $\mathbf{Ts} = \mathbf{Ts}'$, proving the desired result. $\square$

2.11 Infinite state sequences

Definition 15 (State sequence). A *state sequence* is an infinite sequence of states $\mathbf{s}_0, \mathbf{s}_1, \mathbf{s}_2, \ldots$, such that $\mathbf{Ts}_{i+1} = \mathbf{s}_i$ for each $i \geq 0$.

If the translation bound of the previous section applies to $\mathbf{s}_1$ (i.e., $\mathbf{s}_1$ is not a singlet or doublet), then it also applies to all of the subsequent states of the sequence. Then, if $\mathbf{s}_0$ is a winning N-state but not a winning $(N-1)$-state, the translation bound implies that $\mathbf{s}_1$ may be a winning $(N+3)$-state but cannot be a winning $(N+2)$-state. And, in general, $\mathbf{s}_n$ may be a winning $(N+3n)$-state but cannot be a winning $(N+3n-1)$-state.

We will be particularly interested in state sequences which achieve this bound:

Definition 16 (Winning N-state sequence). A state sequence is a *winning N-state sequence* if $\mathbf{s}_n$ is a winning $(N+3n)$-state, for every n.

Definition 17 (Borderline winning N-state sequence). A winning N-state sequence is a *borderline winning N-state sequence* if $\mathbf{s}_n$ is a borderline winning $(N+3n)$-state, for every n.

We can write a winning N-state sequence in the following compact form. Let

$$\begin{aligned}\mathbf{s}_0 &= (s_0, s_1, \ldots, s_E)\\ \mathbf{s}_1 &= (s_{-1}, s_0, s_1, \ldots, s_E)\\ \mathbf{s}_2 &= (s_{-2}, s_{-1}, s_0, s_1, \ldots, s_E)\end{aligned}$$

and so on. Then we can write the entire state sequence as the infinite tuple

$$\mathbf{s} = (\ldots, s_{-n}, s_{1-n}, \ldots, s_{-1}; s_0, s_1, \ldots, s_E).$$

(The semicolon indicates the first state in the state sequence.)

Definition 18 (Partial sums for state sequences). Given an infinite state sequence

$$\mathbf{s} = (\ldots, s_{-n}, s_{1-n}, \ldots, s_{-1}; s_0, s_1, \ldots, s_E),$$

we define

$$s^{(J)} = \sum_{J'=J}^{E} s_{J'}.$$

Definition 19 (Domination for state sequences). If $\mathbf{s}$ and $\mathbf{t}$ are infinite state sequences, we say that $\mathbf{t}$ dominates $\mathbf{s}$ if (and only if)

$$\mathbf{s}^{(J)} \leq \mathbf{t}^{(J)} \text{ for all } J.$$

We will use the notation $\mathbf{t} \geq \mathbf{s}$ for this relation.

Corollary 2 (Domination theorem for state sequences). *If* $\mathbf{t}$ *dominates* $\mathbf{s}$*, and* $\mathbf{t}$ *is a winning* N*-state sequence, then* $\mathbf{s}$ *is a winning* N*-state sequence.*

Proof. Apply the domination theorem for states to $\mathbf{s}_J$ and $\mathbf{t}_J$, for every J. □

2.12 Fibonacci states

To use Corollary 2 to prove that a given state sequence is a winning N-state sequence, we need to have some winning N-state sequences to compare the given sequence to. Fortunately, Berlekamp discovered such sequences: in fact, the sequences he discovered are borderline winning N-state sequences.

A borderline winning N-state sequence is obviously highly constrained. If $\mathbf{s}_0$ is given, and is a borderline winning N-state, and we want $\mathbf{s}_1$ to be a borderline winning $(N+3)$-state, then the value of s_{-1} is determined by the volume formula. We want to have $\mathrm{Vol}_{N+3}(\mathbf{s}_1) = 2^{N+3}$, so we must have

$$s_{-1} = 2^{N+3} - \sum_{J=0}^{E} s_J \sum_{J'=0}^{J+1} \binom{N+3}{J'}.$$

If this formula gives a negative value for s_{-1}, then there is no borderline winning N-state sequence (indeed, no winning N-state sequence at all) beginning with $\mathbf{s}_0$. If the formula gives a nonnegative value for s_{-1}, then we can determine

$$s_{-2} = 2^{N+6} - \sum_{J=-1}^{E} s_J \sum_{J'=0}^{J+2} \binom{N+6}{J'}$$
$$s_{-3} = 2^{N+9} - \sum_{J=-2}^{E} s_J \sum_{J'=0}^{J+3} \binom{N+9}{J'}$$

and so on. Only if s_J turns out to be nonnegative for every J is there a unique borderline winning N-state sequence beginning with $\mathbf{s}_0$.

Berlekamp noticed that for certain starting states, the entire state sequence can be stated in closed form in terms of Fibonacci numbers. This makes it easy to see that the values are all nonnegative, so the state sequence exists. For example, the starting state $(4,1)$, which is a borderline winning 3-state, leads to the borderline winning 3-state sequence $(\ldots, 4414, 1042, 246, 58, 14; 4, 1)$.

We will call such a sequence a *Fibonacci state sequence*, and any state in such a sequence a *Fibonacci state*. Here are some more examples of Fibonacci state sequences:

$$\begin{array}{ll}
(\ldots, 7072, 1744, 352, 128, 0; 16) & (N=4) \\
(\ldots, 4408, 1048, 240, 64, 8; 8) & (N=3) \\
(\ldots, 2728, 644, 152, 36, 8; 4) & (N=2) \\
(\ldots, 1686, 398, 94, 22, 6; 2) & (N=1) \\
(\ldots, 4414, 1042, 246, 58, 14; 4, 1) & (N=3) \\
(\ldots, 2728, 644, 152, 36, 10; 1, 1) & (N=2) \\
(\ldots, 7142, 1686, 398, 94, 24; 5, 0, 1) & (N=4) \\
(\ldots, 4414, 1042, 246, 60, 15; 1, 0, 1) & (N=3) \\
(\ldots, 11556, 2728, 644, 154, 39; 6, 0, 0, 1) & (N=5) \\
(\ldots, 7142, 1686, 400, 99, 21; 1, 0, 0, 1) & (N=4) \\
(\ldots, 18698, 4414, 1044, 253, 60; 7, 0, 0, 0, 1) & (N=6) \\
(\ldots, 11556, 2730, 653, 159, 28; 1, 0, 0, 0, 1) & (N=5) \\
(\ldots, 30254, 7144, 1697, 412, 88; 8, 0, 0, 0, 0, 1) & (N=7) \\
(\ldots, 18700, 4427, 1065, 247, 36; 1, 0, 0, 0, 0, 1) & (N=6) \\
(\ldots, 48954, 11571, 2762, 659, 124; 9, 0, 0, 0, 0, 0, 1) & (N=8) \\
(\ldots, 30271, 7189, 1724, 371, 45; 1, 0, 0, 0, 0, 0, 1). & (N=7)
\end{array}$$

Theorem 9 (Fibonacci state sequences). *There exist (unique) borderline winning N-state sequences, as follows:*

- *With $E = 0$ and $1 \leq N \leq 4$, where the borderline winning N-state is just (2^N).*
- *With $E \geq 1$ and $N = E + 1$, where the borderline winning N-state has the form $(s_0 = 1, 0, 0, \ldots, 0, s_E = 1)$.*
- *With $E \geq 1$ and $N = E + 2$, where the borderline winning N-state has the form $(s_0 = N + 1, 0, 0, \ldots, 0, s_E = 1)$.*

Proof. (1) When computed according to the above definition, all of the entries in the above state sequences are nonnegative integers. (2) Each N-state sequence $\mathbf{s}$ of this class can be written as $\mathbf{a} + \mathbf{b}$, where $\mathbf{Ta} + \mathbf{b}$ and $\mathbf{a} + \mathbf{Tb}$ are winning $(N-1)$-state sequences of this class. See Berlekamp [1968] for details. $\square$

Chapter 3

Computational methods

3.1 Summation rule for states

Suppose that $\mathbf{c}$ and $\mathbf{d}$ are winning $(N-1)$-states. For which states $\mathbf{s}$ does this information suffice to show that $\mathbf{s}$ is a winning N-state?

If we can construct $\mathbf{a}$, $\mathbf{b}$ such that $\mathbf{s} = \mathbf{a} + \mathbf{b}$, $\mathbf{Ta} + \mathbf{b} \leq \mathbf{c}$, and $\mathbf{a} + \mathbf{Tb} \leq \mathbf{d}$, then we can conclude that $\mathbf{s}$ is a winning N-state.

Theorem 10 (Summation theorem for states). *Given states* $\mathbf{s}$*,* $\mathbf{c}$*,* $\mathbf{d}$*, the following are necessary and sufficient conditions for the existence of states* $\mathbf{a}$*,* $\mathbf{b}$ *such that* $\mathbf{s} = \mathbf{a} + \mathbf{b}$*,* $\mathbf{Ta} + \mathbf{b} \leq \mathbf{c}$*, and* $\mathbf{a} + \mathbf{Tb} \leq \mathbf{d}$*:*

$$\begin{aligned} s^{(J+1)} &\leq c^{(J)} \\ s^{(J+1)} &\leq d^{(J)} \\ s^{(J)} + s^{(J+1)} &\leq c^{(J)} + d^{(J)}, \end{aligned}$$

for all $J = 0, 1, \ldots, E$*. (Note that* $s^{(E+1)}$ *is zero by definition.)*

Proof. The conditions are clearly necessary, since:

$$\begin{aligned}
\mathbf{Ta}+\mathbf{b}\le\mathbf{c} &\implies s^{(J+1)}=a^{(J+1)}+b^{(J+1)}\le a^{(J+1)}+b^{(J)}\le c^{(J)}\\
\mathbf{a}+\mathbf{Tb}\le\mathbf{d} &\implies s^{(J+1)}=a^{(J+1)}+b^{(J+1)}\le a^{(J)}+b^{(J+1)}\le d^{(J)}\\
\mathbf{Ta}+\mathbf{b}+\mathbf{a}+\mathbf{Tb}\le\mathbf{c}+\mathbf{d} &\implies s^{(J)}+s^{(J+1)}=a^{(J)}+b^{(J)}+a^{(J+1)}+b^{(J+1)}\\
&\qquad\qquad\le c^{(J)}+d^{(J)}.
\end{aligned}$$

To show the conditions are sufficient, for each J, we need to find $a^{(J)}$, and $b^{(J)}=s^{(J)}-a^{(J)}$, such that all of the following inequalities are simultaneously satisfied:

$$\begin{aligned}
a^{(J+1)}\le a^{(J)} &\iff a^{(J)}\ge a^{(J+1)}\\
b^{(J+1)}\le b^{(J)} &\iff a^{(J)}\le s^{(J)}-b^{(J+1)}\\
a^{(J+1)}+b^{(J)}\le c^{(J)} &\iff a^{(J)}\ge a^{(J+1)}+s^{(J)}-c^{(J)}\\
a^{(J)}+b^{(J+1)}\le d^{(J)} &\iff a^{(J)}\le d^{(J)}-b^{(J+1)}.
\end{aligned}$$

(Recall that $a^{(E+1)}=b^{(E+1)}=0$ by definition.)

Once a valid choice of $a^{(J+1)}$ has been made, there exists at least one value for $a^{(J)}$ that simultaneously satisfies all of these inequalities, because:

$$\begin{aligned}
a^{(J+1)}\le s^{(J)}-b^{(J+1)} &\iff s^{(J+1)}\le s^{(J)}\\
a^{(J+1)}\le d^{(J)}-b^{(J+1)} &\iff s^{(J+1)}\le d^{(J)}\\
a^{(J+1)}+s^{(J)}-c^{(J)}\le s^{(J)}-b^{(J+1)} &\iff s^{(J+1)}\le c^{(J)}\\
a^{(J+1)}+s^{(J)}-c^{(J)}\le d^{(J)}-b^{(J+1)} &\iff s^{(J)}+s^{(J+1)}\le c^{(J)}+d^{(J)}.
\end{aligned}$$

The first follows from the definition of $s^{(J)}$, and the next three are the hypotheses of the theorem. □

Definition 20 (Summation for states). If $\mathbf{c}$, $\mathbf{d}$ are states, we will write $\mathbf{c}\oplus\mathbf{d}$ for the set of states $\mathbf{s}$ that satisfy the inequalities:

$$\begin{aligned}
s^{(J+1)}&\le c^{(J)}\\
s^{(J+1)}&\le d^{(J)}\\
s^{(J)}+s^{(J+1)}&\le c^{(J)}+d^{(J)},
\end{aligned}$$

for $J=0,1,\ldots,E$.

Note that if $\mathbf{s} = \mathbf{a} + \mathbf{b}$, then $\mathbf{s} \in (\mathbf{Ta} + \mathbf{b}) \oplus (\mathbf{a} + \mathbf{Tb})$. But $\mathbf{c} \oplus \mathbf{d}$ is well-defined even if there is no pair of states $\mathbf{a}$, $\mathbf{b}$ such that $\mathbf{c} = \mathbf{Ta} + \mathbf{b}$ and $\mathbf{d} = \mathbf{a} + \mathbf{Tb}$, as will often be the case.

Corollary 3. *If* $\mathbf{c}$, $\mathbf{d}$ *are winning* $(N-1)$*-states, and* $\mathbf{s} \in \mathbf{c} \oplus \mathbf{d}$*, then* $\mathbf{s}$ *is a winning* N*-state.*

Corollary 4. *If* $\mathbf{s} = \mathbf{a} + \mathbf{b}$*,* $\mathbf{Ta} + \mathbf{b} \leq \mathbf{c}$*, and* $\mathbf{a} + \mathbf{Tb} \leq \mathbf{d}$*, then* $\mathbf{s} \in \mathbf{c} \oplus \mathbf{d}$*.*

3.2 Summation rule for state sequences

Theorem 11 (Summation theorem for state sequences). *Given state sequences* $\mathbf{s}$*,* $\mathbf{c}$*,* $\mathbf{d}$*, the following are necessary and sufficient conditions for the existence of state sequences* $\mathbf{a}$*,* $\mathbf{b}$ *such that* $\mathbf{s} = \mathbf{a} + \mathbf{b}$*,* $\mathbf{Ta} + \mathbf{b} \leq \mathbf{c}$*, and* $\mathbf{a} + \mathbf{Tb} \leq \mathbf{d}$*:*

$$\begin{aligned} s^{(J+1)} &\leq c^{(J)} \\ s^{(J+1)} &\leq d^{(J)} \\ s^{(J)} + s^{(J+1)} &\leq c^{(J)} + d^{(J)}, \end{aligned}$$

for all $J \leq E$*. (Note that* $s^{(E+1)}$ *is zero by definition.)*

Proof. The proof is exactly the same as for Theorem 10, but now we consider all values of $J \leq E$, instead of just $J = 0, 1, \ldots, E$. □

Definition 21 (Summation for state sequences). If $\mathbf{c}$, $\mathbf{d}$ are state sequences, we will write $\mathbf{c} \oplus \mathbf{d}$ for the set of state sequences $\mathbf{s}$ that satisfy the inequalities:

$$\begin{aligned} s^{(J+1)} &\leq c^{(J)} \\ s^{(J+1)} &\leq d^{(J)} \\ s^{(J)} + s^{(J+1)} &\leq c^{(J)} + d^{(J)}, \end{aligned}$$

for all $J \leq E$.

Corollary 5. *If* $\mathbf{c}$, $\mathbf{d}$ *are winning* $(N-1)$*-state sequences, and* $\mathbf{s} \in \mathbf{c} \oplus \mathbf{d}$*, then* $\mathbf{s}$ *is a winning* N*-state sequence.*

3.3 Game trees for states

One of our goals is to prove that certain states are winning N-states. Each such state will demonstrate the existence of an optimal adaptive block code for a particular choice of code parameters.

To do this, we will construct *game trees.* We know that we can prove that a state $\mathbf{s}$ is a winning N-state by writing $\mathbf{s} = \mathbf{a} + \mathbf{b}$ and proving that $\mathbf{Ta} + \mathbf{b}$ and $\mathbf{a} + \mathbf{Tb}$ are winning $(N-1)$-states. Or, more generally, we can use Corollary 3. In this case, we need to exhibit $\mathbf{c}$, $\mathbf{d}$ which are winning $(N-1)$-states, such that $\mathbf{s} \in \mathbf{c} \oplus \mathbf{d}$.

In order to show that any state is a winning state this way, we have to have some winning states to start with. We have four sorts of states that we have previously shown to be winning N-states for appropriate choices of N: singlets (Section 2.5), doublets (Theorem 5), zero-error states (Theorem 6), and Fibonacci states (Section 2.12). Given two such winning N-states, we can then show that some other states are winning $(N+1)$-states by using Corollary 3. Given two winning $(N+1)$-states, we can show that another state is a winning $(N+2)$-state, and so on.

We will represent this process of inference by a binary tree. If we check that each leaf is provably a winning N-state for the value of N at that leaf, and that each non-leaf satisfies Definition 20 with respect to its children, then we have a proof that the root of the tree is a winning N-state.

Definition 22 (Game tree of states). A *game tree of states* is a finite binary tree in which each node contains a state $\mathbf{s}$ and an integer $N \geq 0$. At each leaf, $(\mathbf{s}, N)$, $\mathbf{s}$ is provably a winning N-state: either a singlet, or a doublet (Theorem 5), or a zero-error state (Theorem 6), or a Fibonacci state (Theorem 9). Each non-leaf node, $(\mathbf{s}, N)$, has precisely two children, $(\mathbf{c}, N-1)$ and $(\mathbf{d}, N-1)$, such that $\mathbf{s} \in \mathbf{c} \oplus \mathbf{d}$.

Theorem 12 (Game tree theorem for states). *For every node* $(\mathbf{s}, N)$ *in a valid game tree of states,* $\mathbf{s}$ *is a winning* N*-state.*

Proof. At each leaf, $\mathbf{s}$ is provably a winning N-state, by the definition of the game tree. At each other node, Corollary 3 shows that the theorem holds at that node given that it holds at each child node. By induction on the tree, the theorem holds at every node. □

In practice, many of the nodes with the same value N may contain the same state. In that case, we can collapse the tree so that we have only a single node with that state and value, and it is the child of several different parent nodes. It may also be the case that both children of a given node are equal, and again we can make them the same node. In this manner, we can greatly reduce the number of distinct nodes required to represent a proof that a given state is a winning N-state.

3.4 Game trees for state sequences

Another goal is to prove that certain infinite state sequences are winning N-state sequences. Each such state sequence will demonstrate the existence of optimal adaptive block codes for an infinite collection of parameter choices.

In order to show that any state sequence is a winning state sequence, we have to have some winning state sequences to start with. For this purpose we will rely on the Fibonacci state sequences of Theorem 9. Given two such winning N-state sequences, we can then show that some other state sequences are winning $(N+1)$-state sequences by using Corollary 5. Given two winning $(N+1)$-state sequences, we can show that another state is a winning $(N+2)$-state sequence, and so on.

We can also represent this process of inference by a binary tree. If we check that each leaf is provably a winning N-state sequence for the value of N at that leaf, and that each non-leaf satisfies Definition 21 with respect to its children, then we have a proof that the root of the tree is a winning N-state sequence.

Definition 23 (Game tree of state sequences). A *game tree of state sequences* is a finite binary tree in which each node contains a state sequence $\mathbf{s}$ and an integer N. At each leaf, $(\mathbf{s}, N)$, $\mathbf{s}$ satisfies the criteria of Section 2.12. Each non-leaf node, $(\mathbf{s}, N)$, has precisely two children, $(\mathbf{c}, N-1)$ and $(\mathbf{d}, N-1)$, such that $\mathbf{s} \in \mathbf{c} \oplus \mathbf{d}$.

Theorem 13 (Game tree theorem for state sequences). *For every node* $(\mathbf{s}, N)$ *in a valid game tree of state sequences,* $\mathbf{s}$ *is a winning* N*-state sequence.*

Proof. At each leaf, $\mathbf{s}$ is provably a winning N-state sequence, by the definition of the game tree. At each other node, Corollary 5 shows that the theorem holds at that node given that it holds at each child node. By induction on the tree, the theorem holds at every node. □

In practice, many of the nodes with the same value N may contain the same state sequence. In that case, we can collapse the tree so that we have only a single node with that state sequence and value, and it is the child of several different parent nodes. It may also be the case that both children of a given node are equal, and again we can make them the same node. In this manner, we can greatly reduce the number of distinct nodes required to represent a proof that a given state sequence is a winning N-state sequence.

3.5 Truncated state sequences

There is a practical problem with constructing game trees for state sequences: each node is supposed to contain an infinite state sequence, which would require infinite storage.

Fortunately, in some cases we can truncate the state sequences in a given game tree and still prove that the root node is a winning N-state sequence. Suppose that we have a state sequence with only finitely many nonzero coefficients: $\mathbf{s} = (\dots, 0, 0, 0; s_0, s_1, \dots, s_E)$. Suppose further that we have a game tree in the sense of Section 3.3 which proves that the state $\mathbf{s}_0 = (s_0, s_1, \dots, s_E)$ is a winning N-state. Then, under certain additional conditions, we can conclude from that tree that $\mathbf{s}$ is a winning N-state sequence.

Theorem 14 (Truncated game tree theorem). *Consider a game tree of states, with root $(\mathbf{s}_0, N)$, where $\mathbf{s}_0 = (s_0, s_1, \dots, s_E)$ and $M = s^{(0)}$. Suppose that each node, $(\mathbf{t}_0, N')$ contains a winning N'-state $\mathbf{t}_0$. And suppose that each leaf, $(\mathbf{t}_0, N')$, contains a winning N'-state $\mathbf{t}_0$ that can be extended to a winning N'-state sequence $\mathbf{t}$ such that $t^{(-1)} \geq M$. Then $\mathbf{s} = (\dots, 0, 0, 0; s_0, s_1, \dots, s_E)$ is a winning N-state sequence.*

Proof. From the given game tree of states, construct a game tree of state sequences as follows. Replace each node $(\mathbf{t}_0, N')$, containing a state $\mathbf{t}_0$, with a node $(\mathbf{t}, N)$, containing a state sequence $\mathbf{t}$, such that:

$$t^{(J)} = \begin{cases} \min({t_0}^{(J)}, M) & \text{if } J \geq 0, \\ M & \text{if } J < 0, \end{cases}$$

The root of the new tree is $(\mathbf{s}, N)$, by construction. At each leaf node $(\mathbf{t}, N')$, the constructed state sequence $\mathbf{t}$ is dominated by the state sequence provided in the hypothesis of the theorem, so is a winning N'-state sequence.

It remains to show that Definition 21 holds at each node that is not a leaf. Consider such a node, $(\mathbf{t}, N')$, with children $(\mathbf{c}, N'-1)$, $(\mathbf{d}, N'-1)$. The given game tree of states had corresponding nodes $(\mathbf{t}_0, N')$, $(\mathbf{c}_0, N'-1)$, $(\mathbf{d}_0, N'-1)$, with $\mathbf{t}_0 \in \mathbf{c}_0 \oplus \mathbf{d}_0$. From Definition 20, we know that:

$$\begin{aligned} {t_0}^{(J+1)} &\leq {c_0}^{(J)} \\ {t_0}^{(J+1)} &\leq {d_0}^{(J)} \\ {t_0}^{(J)} + {t_0}^{(J+1)} &\leq {c_0}^{(J)} + {d_0}^{(J)}, \end{aligned}$$

for $J = 0, 1, \ldots, E$.

For $J \leq -1$:

$$\begin{aligned} t^{(J+1)} &\leq M = c^{(J)} \\ t^{(J+1)} &\leq M = {d_0}^{(J)} \\ t^{(J)} + t^{(J+1)} &\leq 2M = {c_0}^{(J)} + {d_0}^{(J)}. \end{aligned}$$

For $J \geq 0$:

$$\begin{aligned} t^{(J+1)} &= \min({t_0}^{(J+1)}, M) \leq \min({c_0}^{(J)}, M) = c^{(J)} \\ t^{(J+1)} &= \min({t_0}^{(J+1)}, M) \leq \min({d_0}^{(J)}, M) = d^{(J)} \\ t^{(J)} + t^{(J+1)} &= \min({t_0}^{(J)}, M) + \min({t_0}^{(J+1)}, M) \\ &\leq \min({c_0}^{(J)}, M) + \min({d_0}^{(J)}, M) = c^{(J)} + d^{(J)}. \end{aligned}$$

Thus all the conditions of Definition 21 are satisfied, so $\mathbf{t} \in \mathbf{c} \oplus \mathbf{d}$.

Since this holds at every non-leaf node, and every leaf node contains a winning N'-state sequence, the constructed game tree is a valid game tree of state sequences, so $\mathbf{s}$ is a winning N-state sequence. □

3.6 Losing states

We have previously stated two criteria by which to show that an N-state is a losing N-state (i.e., not a winning N-state). One is the volume bound of Section 2.5, and the other is the translation bound of Section 2.10. For our characterization of optimal adaptive block codes, we need some additional results to show that certain N-states are losing N-states. Because we cannot always divide a state exactly in two, there are N-states which satisfy the volume bound, and yet any potential game tree starting with such a state must violate the volume bound within the first few levels of the tree. The requirement that the states at the first level of the tree must be winning $(N-1)$-states, the states at the second level of the tree must be winning $(N-2)$-states, and so on, translates into a requirement that those states must meet the corresponding volume bounds, if the N-state at the root is to be a winning N-state.

Theorem 15 (First step). *If* $\mathbf{s} = (0, 0, \ldots, 0, 0, s_E)$ *is a winning* N*-state, with* $E > 0$*, then* $\mathbf{t} = (0, 0, \ldots, 0, \lfloor s_E/2 \rfloor, \lceil s_E/2 \rceil)$ *is a winning* $(N-1)$*-state. If* s_E *is odd, then* $\mathrm{Vol}_N(\mathbf{s}) \leq 2^N - \binom{N-1}{E}$*.*

Proof. Suppose that $\mathbf{s} = \mathbf{a} + \mathbf{b}$ where $\mathbf{c} = \mathbf{Ta} + \mathbf{b}$ and $\mathbf{d} = \mathbf{a} + \mathbf{Tb}$ are winning $(N-1)$-states. Then $s_E = a_E + b_E$, so either $a_E \geq \lceil s_E/2 \rceil$ or $b_E \geq \lceil s_E/2 \rceil$. Since $c^{(E)} = b_E$, $d^{(E)} = a_E$, and $c^{(J)} = d^{(J)} = s_E$ for $J < E$, either $\mathbf{t} \leq \mathbf{c}$ or $\mathbf{t} \leq \mathbf{d}$. In either case, $\mathbf{t}$ is a winning $(N-1)$-state.

If E is odd, then

$$\mathbf{t} = (0, 0, \ldots, 0, (s_E - 1)/2, (s_E + 1)/2)$$

is a winning $(N-1)$-state, so:

$$\begin{aligned}
\mathrm{Vol}_{N-1}(\mathbf{t}) &= \frac{s_E - 1}{2}\sum_{J=0}^{E-1}\binom{N-1}{E} + \frac{s_E+1}{2}\sum_{J=0}^{E-1}\binom{N-1}{J} \\
&= s_E\sum_{J=0}^{E-1}\binom{N-1}{J} + \frac{s_E+1}{2}\binom{N-1}{E} \\
&\leq 2^{N-1}.
\end{aligned}$$

Then:

$$\begin{aligned}
\mathrm{Vol}_N(\mathbf{s}) &= s_E\sum_{J=0}^{E}\binom{N}{J} \\
&= s_E\sum_{J=0}^{E}\binom{N-1}{J} + s_E\sum_{J=0}^{E}\binom{N-1}{J-1} \\
&= 2s_E\sum_{J=0}^{E-1}\binom{N-1}{J} + s_E\binom{N-1}{E} \\
&= 2\,\mathrm{Vol}_{N-1}(\mathbf{t}) - \binom{N-1}{E} \\
&\leq 2\cdot 2^{N-1} - \binom{N-1}{E}.
\end{aligned}$$

□

Theorem 16 (Second step). *Suppose that* $\mathbf{s} = (0,0,\ldots,0,0,s_{E-1},s_E)$ *is a winning* N*-state, with* $E > 1$*. Let* $g = \gcd\left(\binom{N-1}{E},\binom{N-1}{E-1}\right)$*. Write* $\binom{N-1}{E} = kg$ *and* $\binom{N-1}{E-1} = \ell g$*. Suppose that* $s_E k + s_{E-1}\ell$ *is odd. Then* $\mathrm{Vol}_N(\mathbf{s}) \leq 2^N - g$*.*

Proof. Write $\mathbf{s} = \mathbf{a} + \mathbf{b}$ such that $\mathbf{Ta} + \mathbf{b}$ and $\mathbf{a} + \mathbf{Tb}$ are winning $(N-1)$-states. By conservation of volume, $\mathrm{Vol}_{N-1}(\mathbf{Ta}+\mathbf{b}) + \mathrm{Vol}_{N-1}(\mathbf{a}+\mathbf{Tb}) = \mathrm{Vol}_N(\mathbf{s})$. Without loss of generality, assume $\mathrm{Vol}_{N-1}(\mathbf{a}+\mathbf{Tb}) \geq \mathrm{Vol}_N(\mathbf{s})/2$. Let $\delta_E = 2a_E - s_E$ and $\delta_{E-1} = 2a_{E-1} - s_{E-1}$. Then:

$$\mathbf{a}+\mathbf{Tb} = (0,0,\ldots,0,(s_{E-1}-\delta_{E-1})/2,(s_E+s_{E-1}-\delta_E+\delta_{E-1})/2,(s_E+\delta_E)/2),$$

so

$$\begin{aligned}\mathrm{Vol}_{N-1}(\mathbf{a}+\mathbf{Tb}) = s_{E-1}\sum_{J=0}^{E-2}\binom{N-1}{J} + \frac{s_{E-1}+\delta_{E-1}}{2}\binom{N-1}{E-1}\\ + s_E\sum_{J=0}^{E-1}\binom{N-1}{J} + \frac{s_E+\delta_E}{2}\binom{N-1}{E}.\end{aligned}$$

Then:

$$\begin{aligned}\mathrm{Vol}_N(\mathbf{s}) &= s_{E-1}\sum_{J=0}^{E-1}\binom{N}{J} + s_E\sum_{J=0}^{E}\binom{N}{J}\\ &= 2s_{E-1}\sum_{J=0}^{E-2}\binom{N-1}{J} + s_{E-1}\binom{N-1}{E-1} + 2s_E\sum_{J=0}^{E-1}\binom{N-1}{J} + s_E\binom{N-1}{E}\\ &= 2\,\mathrm{Vol}_{N-1}(\mathbf{a}+\mathbf{Tb}) - \delta_{E-1}\binom{N-1}{E-1} - \delta_E\binom{N-1}{E}\\ &= 2\,\mathrm{Vol}_{N-1}(\mathbf{a}+\mathbf{Tb}) - g(\delta_E k + \delta_{E-1}\ell).\end{aligned}$$

But $\delta_E \equiv s_E \pmod 2$ and $\delta_{E-1} \equiv s_{E-1} \pmod 2$, so

$$\delta_E k + \delta_{E-1}\ell \equiv s_E k + s_{E-1}\ell \equiv 1 \pmod 2,$$

so $\delta_E k + \delta_{E-1}\ell$ is not zero. Since $\mathrm{Vol}_{N-1}(\mathbf{a}+\mathbf{Tb}) \geq \mathrm{Vol}_N(\mathbf{s})/2$, $\delta_E k + \delta_{E-1}\ell$ must be a positive integer, so $\mathrm{Vol}_N(\mathbf{s}) \leq 2\,\mathrm{Vol}_{N-1}(\mathbf{a}+\mathbf{Tb}) - g \leq 2^N - g$. □

3.7 A three-step losing state

We want to demonstrate one more losing state, not covered by Theorem 15 or Theorem 16 of the previous section. Rather than generalize to other states, we will consider only the specific result we need. Of course, a generalization of this calculation would be possible.

Theorem 17 (Three-step theorem). *The 37-state*

$$\mathbf{s} = (0, 0, 0, 0, 0, 0, 48475)$$

is a losing 37-state.

Proof. Assume that $\mathbf{s}$ is a winning 37-state. Then, by Theorem 15,

$$\mathbf{t} = (0,0,0,0,0,24237,24238)$$
$$\mathrm{Vol}_{36}(\mathbf{t}) = 68719133896 < 68719476736 = 2^{36}$$

is a winning 36-state. So there exist $\mathbf{a}$, $\mathbf{b}$ such that $\mathbf{t} = \mathbf{a}+\mathbf{b}$ and $\mathbf{Ta}+\mathbf{b}$, $\mathbf{a}+\mathbf{Tb}$ are winning 35-states. Choose δ_5, δ_6 such that:

$$\mathbf{a} = (0,0,0,0,0,(24237-\delta_5)/2,(24238-\delta_6)/2)$$
$$\mathbf{b} = (0,0,0,0,0,(24237+\delta_5)/2,(24238+\delta_6)/2).$$

Since $\mathrm{Vol}_{36}(\mathbf{t}) = \mathrm{Vol}_{35}(\mathbf{Ta}+\mathbf{b}) + \mathrm{Vol}_{35}(\mathbf{a}+\mathbf{Tb})$, assume without loss of generality that $\mathrm{Vol}_{35}(\mathbf{Ta}+\mathbf{b}) \geq \mathrm{Vol}_{36}(\mathbf{t})/2$. So

$$\mathbf{u} = \mathbf{Ta}+\mathbf{b} = (0,0,0,0,(24237-\delta_5)/2,(48475-\delta_6+\delta_5)/2,(24238+\delta_6)/2)$$

$$\begin{aligned}\mathrm{Vol}_{35}(\mathbf{u}) &= 34359566948 + 162316\delta_5 + 811580\delta_6\\ &= 34359566948 + 162316(\delta_5+5\delta_6)\end{aligned}$$

is a winning 35-state. But in order for the elements of $\mathbf{a}$ to be integers, δ_5 must be odd and δ_6 must be even, so $\delta_5+5\delta_6$ is odd, and positive since $\mathrm{Vol}_{35}(\mathbf{u}) \geq \mathrm{Vol}_{36}(\mathbf{t})/2 = 34359566948$.

But if $\delta_5+5\delta_6 \geq 3$, then

$$\mathrm{Vol}_{35}(\mathbf{u}) \geq 34359566948 + 3\cdot 162316 = 34360053896 > 34359738368 = 2^{35}.$$

So $\delta_5+5\delta_6 = 1$, so $\delta_5 = 1-5\delta_6$, so

$$\mathbf{u} = (0,0,0,0,(24236+5\delta_6)/2,(48476-6\delta_6)/2,(24238+\delta_6)/2)$$
$$\mathrm{Vol}_{35}(\mathbf{u}) = 34359566948 + 162316 = 34359729264$$

is a winning 35-state. So there exist $\mathbf{c}$, $\mathbf{d}$ such that $\mathbf{u} = \mathbf{c}+\mathbf{d}$ and $\mathbf{Tc}+\mathbf{d}$, $\mathbf{c}+\mathbf{Td}$ are winning 34-states. Choose $\epsilon_4, \epsilon_5, \epsilon_6$ such that:

$$\mathbf{c} = (0,0,0,0,(24236+5\delta_6-\epsilon_4)/4,(48476-6\delta_6-\epsilon_5)/4,(24238+\delta_6-\epsilon_6)/4)$$
$$\mathbf{d} = (0,0,0,0,(24236+5\delta_6+\epsilon_4)/4,(48476-6\delta_6+\epsilon_5)/4,(24238+\delta_6+\epsilon_6)/4).$$

Since $\mathrm{Vol}_{35}(\mathbf{u}) = \mathrm{Vol}_{34}(\mathbf{Tc}+\mathbf{d}) + \mathrm{Vol}_{34}(\mathbf{c}+\mathbf{Td})$, assume without loss of generality that $\mathrm{Vol}_{34}(\mathbf{Tc}+\mathbf{d}) \geq \mathrm{Vol}_{35}(\mathbf{u})/2$. So

$$\begin{aligned}\mathbf{v} = \mathbf{Tc}+\mathbf{d} = (0,0,0,&(24236+5\delta_6-\epsilon_4)/4,(72712-\delta_6-\epsilon_5+\epsilon_4)/4,\\ &(72714-5\delta_6-\epsilon_6+\epsilon_5)/4,(24238+\delta_6+\epsilon_6)/4)\end{aligned}$$

$$\begin{aligned}\mathrm{Vol}_{34}(\mathbf{v}) &= 17179864632 + 11594\epsilon_4 + 69564\epsilon_5 + 336226\epsilon_6\\ &= 17179864632 + 11594(\epsilon_4+6\epsilon_5+29\epsilon_6).\end{aligned}$$

is a winning 34-state. But in order for the elements of $\mathbf{c}$ to be integers,

$$\begin{aligned}\epsilon_4 &\equiv 24236+5\delta_6 \equiv \delta_6 \pmod 4\\ \epsilon_5 &\equiv 48476-6\delta_6 \equiv 2\delta_6 \pmod 4\\ \epsilon_6 &\equiv 24238+\delta_6 \equiv 2+\delta_6 \pmod 4\\ \epsilon_4+6\epsilon_5+29\epsilon_6 &\equiv \epsilon_4+2\epsilon_5+\epsilon_6 \equiv 2+2\delta_6 \equiv 2 \pmod 4.\end{aligned}$$

But $\epsilon_4+6\epsilon_5+29\epsilon_6$ cannot be negative, since $\mathrm{Vol}_{34}(\mathbf{v}) \geq \mathrm{Vol}_{35}(\mathbf{u})/2 = 17179864632$, so $\epsilon_4+6\epsilon_5+29\epsilon_6$ is at least 2, so

$$\mathrm{Vol}_{34}(\mathbf{v}) \geq 17179864632 + 2\cdot 11594 = 17179887820 > 17179869184 = 2^{34}.$$

So $\mathbf{v}$ cannot be a winning 34-state. $\square$

Corollary 6. *There is no adaptive block code with $N = 37$, $E = 6$, and $M = 48475$.*

Chapter 4

Optimal codes

4.1 Main Result

When $M = 1$ or $M = 2$, determining when an adaptive error-correcting code exists is trivial; see Sections 2.5 and 2.7. (When $M = 1$, the code always exists. When $M = 2$, the code exists iff $N \geq 2E+1$.) Our main result is to completely determine necessary and sufficient conditions for the existence of adaptive error-correcting codes, for all values of M up to 2^{20}.

Definition 24 (Optimal code). An *optimal adaptive error-correcting code* has parameters N, E, M such that there exists no adaptive error-correcting code with the same value of N and E and a larger value of M.

Theorem 18 (Optimal codes). *The optimal adaptive error-correcting codes with* $3 \leq M \leq 2^{20}$ *are precisely those shown in Table 4.1.*

Corollary 7 (Main result). *If* $3 \leq M \leq 2^{20}$*, there exists an adaptive block code with parameters* N*,* E*,* M *if and only if* M *is less than or equal to the value indexed by* N *and* E *in Table 4.1, or if* N *is greater than the largest value indexed for* E *in Table 4.1.*

The proof combines the techniques of Chapter 3 with a large computation, to construct search trees demonstrating the existence of the codes in Table 4.1. The fact

N	$E=0$	$E=1$	$E=2$	$E=3$	$E=4$	$E=5$	$E=6$	$E=7$	$E=8$	$E=9$	$E\geq 10$
$3E+2$	4	4	4	4	4	4	4	4	4	4	4
$3E+3$	8	8	8	8	8	8	8	8	8	8	8
$3E+4$	16	16	16	16	16	16	16	16	16	16	16
$3E+5$	32	28	28	28	28	28	28	28	28	28	28
$3E+6$	64	50	50	50	50	50	50	50	50	50	50
$3E+7$	128	92	88	88	88	88	88	88	88	88	88
$3E+8$	256	170	154	154	154	154	154	154	154	154	154
$3E+9$	512	314	270	264	264	264	264	264	264	264	264
$3E+10$	1024	584	478	451	451	451	451	451	451	451	451
$3E+11$	2048	1092	850	776	768	768	768	768	768	768	768
$3E+12$	4096	2048	1524	1342	1294	1294	1294	1294	1294	1294	1294
$3E+13$	8192	3854	2744	2337	2196	2192	2192	2192	2192	2192	2192
$3E+14$	16384	7280	4968	4096	3748	3662	3662	3662	3662	3662	3662
$3E+15$	32768	13796	9038	7216	6436	6154	6154	6154	6154	6154	6154
$3E+16$	65536	26214	16512	12776	11110	10406	10249	10249	10249	10249	10249
$3E+17$	131072	49932	30282	22732	19282	17686	17116	17116	17116	17116	17116
$3E+18$	262144	95324	55738	40622	33626	30216	28734	28467	28467	28467	28467
$3E+19$	524288	182360	102926	72884	58904	51868	48474	47306	47306	47306	47306
$3E+20$	1048576	349524	190650	131264	103620	89438	82160	78986	78589	78589	78589
$3E+21$		671088	354136	237238	183005	154876	139866	132474	130088	130088	130088
$3E+22$			659546	430184	324416	269268	239102	223138	216269	215900	215900
$3E+23$				782468	577124	469928	410374	377388	361034	356245	356245
$3E+24$					1030090	823076	706994	640756	605094	590174	590174
$3E+25$									1017990	981464	972164

Table 4.1: All Optimal Adaptive Codes with $M \leq 2^{20}$

that each code is optimal (i.e., the nonexistence of adaptive codes with larger values of M) in each case follows from the results in Chapter 3.

4.2 Upper bounds on M

In order to show that each value in Table 4.1 is optimal, we need to show that increasing M by 1 makes it impossible to construct an adaptive error-correcting code. In each case, the result follows from results in Chapter 3. The symbols in Table 4.2 show which theorem applies in each case:

V: Theorem 3 (Volume bound)

1: Theorem 15 (First step)

2: Theorem 16 (Second step)

3: Theorem 17 (Third step)

T: Theorem 8 (Translation bound)

N	$E=0$	$E=1$	$E=2$	$E=3$	$E=4$	$E=5$	$E=6$	$E=7$	$E=8$	$E=9$	$E\geq 10$
$3E+2$	V	T	T	T	T	T	T	T	T	T	T
$3E+3$	V	T	T	T	T	T	T	T	T	T	T
$3E+4$	V	T	T	T	T	T	T	T	T	T	T
$3E+5$	V	V	T	T	T	T	T	T	T	T	T
$3E+6$	V	1	T	T	T	T	T	T	T	T	T
$3E+7$	V	1	1	T	T	T	T	T	T	T	T
$3E+8$	V	V	V	T	T	T	T	T	T	T	T
$3E+9$	V	1	V	1	T	T	T	T	T	T	T
$3E+10$	V	1	V	V	T	T	T	T	T	T	T
$3E+11$	V	V	1	V	1	T	T	T	T	T	T
$3E+12$	V	V	V	V	1	T	T	T	T	T	T
$3E+13$	V	1	V	V	V	V	T	T	T	T	T
$3E+14$	V	1	1	V	V	V	T	T	T	T	T
$3E+15$	V	1	1	V	V	1	T	T	T	T	T
$3E+16$	V	V	1	2	1	1	V	T	T	T	T
$3E+17$	V	V	1	1	1	1	1	T	T	T	T
$3E+18$	V	1	V	V	V	1	V	V	T	T	T
$3E+19$	V	1	1	1	V	1	3	1	T	T	T
$3E+20$	V	1	V	V	V	1	V	V	V	T	T
$3E+21$		V	V	V	V	V	1	V	V	T	T
$3E+22$			V	1	2	V	1	V	V	V	T
$3E+23$				V	1	V	V	V	1	V	T
$3E+24$					1	V	V	V	V	V	T
$3E+25$									1	1	VT

Table 4.2: Derivation of Upper Bounds

Note that when $E = 10$ and $N = 55$, the result follows from the Volume bound; when $E > 10$ and $N = 3E + 25$, the result follows from the Translation bound.

When $N < 3E + 2$, no adaptive error-correcting code can have $M > 2$. If $E = 0$ and $N = 1$, $M = 3$ would violate the translation bound. If $E > 0$ and $N = 3E + 1$, then $M = 3$ would violate the volume bound. If $N < 3E + 1$, then $M = 3$ would violate Lemma 1.

To prove Theorem 18, it remains to show that: (1) for each entry in Table 4.1, an adaptive error-correcting code exists with the specified values of N, E, M; (2) for values of N greater than those in Table 4.1, an adaptive error-correcting code exists with $M > 2^{20}$.

4.3 Search algorithm for states

The results of Chapter 3 let us use search trees to demonstrate the existence of adaptive block codes. In this chapter we will describe the algorithms that allow us to construct search trees, and thus establish our main results.

Our basic approach is a recursive depth-first search for constructing search trees, with caching of intermediate results. The goal of the algorithm is to prove that a given N-state is a winning N-state. (In some cases, we will subsequently extend the analysis to show that a given N-state sequence is a winning N-state sequence.)

We use a cache to record winning N'-states as we find them, so that later in the depth-first search, if we visit a state that is dominated by one of these cached winning states, we know that the new state is a winning state without further search.

Algorithm 1 (Depth-first search).

Input: An N-state $\mathbf{s}$.

Output: A proof that $\mathbf{s}$ is a winning N-state, or failure.

Start: Initialize an empty table of winning N'-states, for each $N' \leq N$.

Procedure:

1. If $\mathbf{s}$ is provably a winning N-state (using Theorem 9), find $\mathbf{t}$, a winning N-state which dominates $\mathbf{s}$; go to step 9.

2. If $\mathbf{s}$ is provably a losing state (using Theorem 3, the volume bound, or Theorem 8, the translation bound), return failure.

3. Search the table of winning N-states for a winning N-state $\mathbf{t}$ which dominates $\mathbf{s}$; if found, return $\mathbf{t}$.

4. Choose states $\mathbf{a}$ and $\mathbf{b}$ such that $\mathbf{s} = \mathbf{a} + \mathbf{b}$ (using Algorithm 2).

5. Apply Algorithm 1 to the $(N-1)$-state $\mathbf{c} = \mathbf{Ta} + \mathbf{b}$. If the algorithm fails, go to step 8.

6. Apply Algorithm 1 to the $(N-1)$-state $\mathbf{d} = \mathbf{a} + \mathbf{T}\mathbf{b}$. If the algorithm fails, go to step 8.

7. If Algorithm 1 succeeds for $\mathbf{c}$ and $\mathbf{d}$, returning $\mathbf{c}'$ and $\mathbf{d}'$ respectively, find $\mathbf{t} \in \mathbf{c}' \oplus \mathbf{d}'$ which dominates $\mathbf{s}$ (using Algorithm 4); go to step 9.

8. If Algorithm 1 fails for $\mathbf{c}$ or $\mathbf{d}$, repeat step 5 with a different choice of $\mathbf{a}$ and $\mathbf{b}$ (using Algorithm 3). If too many choices of $\mathbf{a}$ and $\mathbf{b}$ have already been tried, return failure.

9. Store the winning N-state $\mathbf{t}$ which dominates $\mathbf{s}$ in the table of winning N-states; return $\mathbf{t}$.

Theorem 19 (Search theorem for states). *If Algorithm 1 with input* $(\mathbf{s}, N)$ *returns with success, then* $\mathbf{s}$ *is a winning* N*-state.*

Proof. The winning states found by Algorithm 1 form the game tree in the hypothesis of Theorem 12. The root of the tree contains $(\mathbf{t}, N)$, where $\mathbf{t}$ dominates $\mathbf{s}$. The algorithm does not return with success unless the game tree is complete. □

If Algorithm 1 returns failure, that does not imply that the state $\mathbf{s}$ cannot be a winning N-state, because not all possibilities for $\mathbf{a}, \mathbf{b}$ are tried in step 4. However, in the cases we shall consider for the proof of Theorem 18, when Algorithm 1 fails, we shall be able to show that $\mathbf{s}$ is not a winning N-state.

4.4 Partitioning algorithm

One key to Algorithm 1 is making a good choice of $\mathbf{a}$ and $\mathbf{b}$ in step 4. Of course, many different choices may work, and any choice that works leads to a correct proof. Because sometimes our first choice of $\mathbf{a}$ and $\mathbf{b}$ fails, we also need a method to generate additional candidates.

These algorithms are implemented by the `depthsearch` function in Appendix A.

Algorithm 2 (Partitioning).

Input: An N-state $\mathbf{s}$.

Output: A pair of states $\mathbf{a}$, $\mathbf{b}$ such that $\mathbf{s} = \mathbf{a} + \mathbf{b}$, or failure.

Start: Let $\mathbf{s} = (s_0, s_1, \dots, s_E)$.

Procedure:

In turn for $J = E$, $J = E - 1$, $\dots$, $J = 0$, choose a_J, b_J so that $s_J = a_J + b_J$, as follows.

1. If $s_J = 0$, let $a_J = 0$ and $b_J = 0$.
2. If $s_J = 1$ and $s^{(J+1)} = 0$, let $a_J = 1$ and $b_J = 0$.
3. If $s_J = 1$ and $s^{(J+1)} = 1$, let $a_J = 0$ and $b_J = 1$.
4. If $s_J = 2$ and $s^{(J+1)} = 0$, let $a_J = 1$ and $b_J = 1$.
5. Otherwise, $s^{(J)} \geq 3$, so $\mathbf{T}^I\mathbf{s}$ is a winning $(N - 3I)$-state, for $0 \leq I \leq J$ (by Theorem 8). We want $\mathbf{T}^{I+1}\mathbf{a}+\mathbf{T}^I\mathbf{b}$ and $\mathbf{T}^I\mathbf{a}+\mathbf{T}^{I+1}\mathbf{b}$ to be winning $(N-3I-1)$-states; this places an upper and lower bound on $\mathrm{Vol}_{N-3I-1}(\mathbf{T}^{I+1}\mathbf{a} + \mathbf{T}^I\mathbf{b})$, for each I. Given the previous choices of $a_{J+1}, \dots, a_E$, and given that $0 \leq a_K \leq s_K$ for $0 \leq K \leq J - 1$, for each I this determines an interval of feasible values for a_J. Take the intersection of those intervals, and if it contains no integer value, return failure. Otherwise, let a_J be the integer closest to the midpoint of the interval, except that if it is less than 0 let $a_J = 0$, and if it is greater than s_J let $a_J = s_J$. Let $b_J = s_J - a_J$.

The idea of Algorithm 2 is that we want to divide the lower levels of $\mathbf{s}$ approximately evenly between $\mathbf{a}$ and $\mathbf{b}$. The choice of a_J will constrain our ability to do that while satisfying the volume bounds. So we make a choice that corresponds as closely as possible to satisfying the volume bounds when a_K and b_K are approximately $s_K/2$ for $0 \leq K \leq J - 1$. Intersecting the intervals for the various values of I means that the calculation is dominated by the smallest interval, the bound which will be most difficult to satisfy.

Algorithm 3 (Biased partitioning).

Input: An N-state $\mathbf{s}$, an integer *bias level* $L \geq 0$, and a small nonzero integer *bias* B.

Output: A pair of states $\mathbf{a}$, $\mathbf{b}$ such that $\mathbf{s} = \mathbf{a} + \mathbf{b}$, or failure.

Start: Let $\mathbf{s} = (s_0, s_1, \ldots, s_E)$.

Procedure:

In turn for $J = E$, $J = E - 1$, …, $J = 0$, choose a_J, b_J so that $s_J = a_J + b_J$, as follows. Use the method of Algorithm 2, except that when $J = L$, add B to the value of a_J computed by Algorithm 2. If then $a_J < 0$ or $a_J > s_J$, return failure.

The idea of Algorithm 3 is that the greatest shortcoming of Algorithm 2 is that we must choose an integer value at each level. (If we could choose a real number at each level, satisfying all of the volume bounds would be easy.) So, if we fail, it is natural to try changing one of our choices by $+1$ or -1, since we might have rounded to the "wrong" integer value. Once we have introduced this "bias", we can continue with the method of Algorithm 2 on the lower levels.

4.5 Summation algorithm

A second key to Algorithm 1 is choosing the state $\mathbf{t} \in \mathbf{c} \oplus \mathbf{d}$ in step 7. Again, many different choices could work, and any choice that works leads to a correct proof. In general, we want to choose $\mathbf{t}$ as "large" as possible, so that when we store $\mathbf{t}$ in our table of winning N-states, we will be likely to later find other N-states in our search that are dominated by $\mathbf{t}$.

The method we use, which works well in practice, is to make the lexicographically largest choice of $\mathbf{t}$. That is, of all of the elements of $\mathbf{c} \oplus \mathbf{d}$ that dominate $\mathbf{s}$, we choose $\mathbf{t}$ with t_E as large as possible. Among such states, we choose $\mathbf{t}$ with t_{E-1} as large as possible, and so on.

When applying Algorithm 1 to a root state $\mathbf{s}$ with $s^{(0)} = M$, all of the intermediate states in the game tree for $\mathbf{s}$ can be chosen to have $t^{(0)} \le M$. So, for convenience, we will also require that the state produced by Algorithm 4 has $t^{(0)} \le M$. This constraint has no effect on the success of the program; it simply lets us avoid computing and storing very large numbers that would otherwise occur.

This algorithm is implemented by the `addstate` function in Appendix A.

Algorithm 4 (Summation).

Input: An N-state $\mathbf{s}$, $(N-1)$-states $\mathbf{c}$, $\mathbf{d}$ such that $\mathbf{s} \in \mathbf{c} \oplus \mathbf{d}$, and an integer $M \ge s^{(0)}$.

Output: An N-state $\mathbf{t}$ such that $\mathbf{s} \le \mathbf{t}$, $\mathbf{t} \in \mathbf{c} \oplus \mathbf{d}$, and $t^{(0)} \le M$, and which is lexicographically largest among all such $\mathbf{t}$.

Start: Let $\mathbf{s} = (s_0, s_1, \ldots, s_E)$.

Procedure:

1. Let $t^{(E+1)} = 0$.
2. In turn for $J = E$, $J = E - 1$, ..., $J = 1$, let
$$t^{(J)} = \min(M, c^{(J-1)}, d^{(J-1)}, c^{(J)} + d^{(J)} - t^{(J+1)}, \min_{0 \le I \le J-1}(c^{(I)} + d^{(I)} - s^{(I)})).$$
3. Let
$$t^{(0)} = \min(M, c^{(0)} + d^{(0)} - t^{(1)}).$$

Proof. From Definition 20, if $\mathbf{t} \in \mathbf{c} \oplus \mathbf{d}$, $J > 0$, and $0 \le I \le J - 1$, then

$$\begin{aligned} t^{(J)} &\le c^{(J-1)} \\ t^{(J)} &\le d^{(J-1)} \\ t^{(J)} &\le c^{(J)} + d^{(J)} - t^{(J+1)} \\ t^{(J)} \le t^{(I+1)} &\le c^{(I)} + d^{(I)} - t^{(I)}. \end{aligned}$$

If also $\mathbf{s} \le \mathbf{t}$, then $s^{(I)} \le t^{(I)}$, so

$$t^{(J)} \le c^{(I)} + d^{(I)} - s^{(I)}.$$

And if $t^{(0)} \leq M$, then

$$t^{(J)} \leq t^{(0)} \leq M.$$

So

$$t^{(J)} \leq \min(M, c^{(J-1)}, d^{(J-1)}, c^{(J)} + d^{(J)} - t^{(J+1)}, \min_{0 \leq I \leq J-1}(c^{(I)} + d^{(I)} - s^{(I)}))$$

is a necessary condition on $t^{(J)}$ if $\mathbf{t}$ is to be the result of Algorithm 4. Similarly,

$$t^{(0)} \leq \min(M, c^{(0)} + d^{(0)} - t^{(1)})$$

is necessary.

Next, we need to show that Algorithm 4 computes $t^{(J)} \geq t^{(J+1)}$ for each J, so that $\mathbf{t}$ is a valid state. This follows because:

$$\begin{aligned} c^{(J-1)} &\geq c^{(J)} \\ d^{(J-1)} &\geq d^{(J)} \\ c^{(J)} + d^{(J)} - t^{(J+1)} &\geq t^{(J+1)} + t^{(J+1)} - t^{(J+1)} = t^{(J+1)} \end{aligned}$$

So each term in the formula for $t^{(J)}$ is at least as large as $t^{(J+1)}$.

Now, $\mathbf{t} \in \mathbf{c} \oplus \mathbf{d}$ and $\mathbf{t}^{(0)} \leq M$ follow directly from the formula for $t^{(J)}$ and $t^{(0)}$. To show that $t^{(J)} \geq s^{(J)}$, we observe that if $0 \leq I \leq J-1$, then

$$\begin{aligned} M &\geq s^{(0)} \geq s^{(J)} \\ c^{(J-1)} &\geq s^{(J)} \\ d^{(J-1)} &\geq s^{(J)} \\ c^{(J)} + d^{(J)} - t^{(J+1)} &\geq c^{(J)} + d^{(J)} - (c^{(J)} + d^{(J)} - s^{(J)}) = s^{(J)} \\ c^{(I)} + d^{(I)} - s^{(I)} &\geq s^{(I+1)} \geq s^{(J)} \end{aligned}$$

So the formula for $t^{(J)}$ is at least $s^{(J)}$. And similarly,

$$c^{(0)} + d^{(0)} - t^{(1)} \geq c^{(0)} + d^{(0)} - (c^{(0)} + d^{(0)} - s^{(0)}) = s^{(0)}.$$

Finally, at each step, the choice of $t^{(J)}$ or $t^{(0)}$ is the largest consistent with the goal of Algorithm 4. So $\mathbf{t}$ is the lexicographically largest such state.

$\square$

4.6 Lifting game trees

If the game tree produced by Algorithm 1 with input $(\mathbf{s}_0, N)$ satisfies certain additional conditions, we can conclude not only that $\mathbf{s}_0 = (s_0, s_1, \ldots, s_E)$ is a winning N-state, but that $\mathbf{s} = (\ldots, 0, 0, 0; s_0, s_1, \ldots, s_E)$ is a winning N-state sequence. The conditions are given by Theorem 14: for each leaf of the game tree produced by Algorithm 1, $(\mathbf{t}_0, N')$, we will say that the state is *extendable* if we can extend $\mathbf{t}_0$ to a winning N'-state sequence $\mathbf{t}$ such that $t^{(-1)} \geq s^{(0)}$.

This is a condition that we can check after the execution of Algorithm 1. Each leaf node contains a Fibonacci state, which can be extended to a state sequence (Section 2.12). So, if each of the leaves reached in the execution of Algorithm 1 is extendable, then we can assert that $\mathbf{s}$ is a winning N-state sequence.

If not all of the leaves are extendable, then we can attempt to *lift* the game tree. Lifting replaces each node $(\mathbf{t}, N')$ with a node $(\mathbf{t}', N' + 3)$, where $\mathbf{t}'$ is a winning $(N' + 3)$-state. If the lifted game tree is still a valid game tree, then we can check whether its leaves are extendable. Lifting the Fibonacci states at the leaves several times will eventually ensure that all of the leaves are extendable. So, if we can lift the game tree enough times, we can eventually obtain a tree with the desired property.

Lifting starts at the leaves of the tree and proceeds toward the root. Lifting of each node assumes that the children of that node have been lifted. If the children $(\mathbf{a}, N' - 1)$ and $(\mathbf{b}, N' - 1)$ have been lifted to $(\mathbf{a}', N' + 2)$ and $(\mathbf{b}', N' + 2)$, then we must lift the node $(\mathbf{t}, N')$ to a node $(\mathbf{t}', N' + 3)$ such that $\mathbf{t}' \in \mathbf{a}' \oplus \mathbf{b}'$.

Algorithm 5 (Lifting).

Input: A game tree with the N-state $\mathbf{s} = (s_0, s_1, \ldots, s_E)$ at the root, and Fibonacci states at the leaves.

Output: A game tree with the $(N+3)$-state $(0, s_0, s_1, \ldots, s_E)$ at the root, or failure.

Procedure:

1. Each leaf node $(\mathbf{t}, N')$ contains a Fibonacci state $\mathbf{t}$. That Fibonacci state belongs to a state sequence; lift the node to $(\mathbf{t}', N' + 3)$ where $\mathbf{t} = \mathbf{T}\mathbf{t}'$ and $\mathbf{t}'$ is taken

from the Fibonacci state sequence containing $\mathbf{t}$.

2. For each non-leaf node $(\mathbf{t}, N')$, after its children have been lifted to $(\mathbf{a}, N'+2)$ and $(\mathbf{b}, N'+2)$, use Algorithm 4 to find the lexicographically largest $\mathbf{t}' \in \mathbf{a}' \oplus \mathbf{b}'$ such that $\mathbf{T}\mathbf{t}' \geq \mathbf{t}$, if such an $\mathbf{t}'$ exists. Lift to $(\mathbf{t}', N'+3)$.

3. Otherwise, use Algorithm 4 to find the lexicographically largest $\mathbf{t}' \in \mathbf{a}' \oplus \mathbf{b}'$, without regard to $\mathbf{t}$. Lift to $(\mathbf{t}', N'+3)$.

4. When all nodes have been lifted, if the root node $(\mathbf{s}', N+3)$ has $\mathbf{s}' \geq (0, s_0, s_1, \ldots, s_E)$ return success, otherwise failure.

Lifting may fail when it would have been possible to lift the game tree with a more sophisticated algorithm, but it succeeds often enough for our purpose. Step (3) is important: lifting will often succeed with step (3) as an option, and fail without it.

4.7 Computational results

Appendix A contains the program which implements Algorithms 1, 2, 3, 4, and 5. Appendix B contains the output of this program, which takes about 240 seconds to run to completion on a 600 MHz Pentium III system.

Each line of the output displays the success or failure of Algorithm 1 to find an optimal adaptive block code, or the success or failure of Algorithm 5 to "lift" a game tree. When Algorithm 1 succeeds, then there exists an adaptive block code with parameters N, E, M. When Algorithm 5 succeeds, and when all of the leaves of the lifted game tree are "safe", then Theorem 14 implies the existence of adaptive block codes with parameters $N+3k$, $E+k$, M, for all $k \geq 0$. Together, these results from Appendix B imply all of the numeric values in Table 4.1. Note that Algorithm 1 fails, in the cases in Appendix B, only when one of the results in Table 4.2 demonstrates that it must fail. So we can conclude that the results in Table 4.1 are optimal.

Appendix B also contains results that imply that for all values of N larger than those shown in Table 4.1, there exists an adaptive block code with $M > 2^{20} = 1048576$. This completes the proof of Theorem 18 and Corollary 7.

Bibliography

[Balakirsky] Vladimir B. Balakirsky, *A Direct Approach to Searching with Lies*, submitted to *Discrete Mathematics.* Author is currently with Fakultät für Mathematik, Universität Bielefeld, Postfach 100131, D-33501 Bielefeld 1, Germany, submitted

[Berlekamp] Elwyn R. Berlekamp, *Block Coding with Noiseless Feedback*, doctoral dissertation, Massachusetts Institute of Technology, 1964.

[Berlekamp2] Elwyn R. Berlekamp, *Block Coding for the Binary Symmetric Channel with Noiseless, Delayless Feedback* in *Error Correcting Codes*, Proceedings of a Symposium Conducted by the Mathematics Research Center, United States Army at the University of Wisconsin, Madison, edited by Henry B. Mann, John Wiley & Sons, Inc. (May 6–8, 1968), pp. 61–88.

[Cesa-Bianchi] Nicoló Cesa-Bianchi, Yoav Freund, David P. Helmbold, and Manfred K. Warmuth, *On-line Prediction and Conversion Strategies* in *Machine Learning* **25**, Kluwer Academic Publishers, Boston (1996), pp. 71–110.

[Czyzowicz] Jurek Czyzowicz, Daniele Mundici, and Andrezej Pelc, *Solution of Ulam's Problem on Binary Search with Two Lies* in *Journal of Combinatorial Theory*, Series A **49** (1988), pp. 384–388.

[Czyzowicz2] Jurek Czyzowicz, Daniele Mundici, and Andrezej Pelc, *Ulam's Search-*

ing Game with Lies in *Journal of Combinatorial Theory*, Series A **52** (1989), pp. 62–76.

[Deppe] Christian Deppe, *Solution of Ulam's Searching Game with Three Lies or an Optimal Adaptive Strategy for Binary Three-Error-Correcting-Codes*, doctoral dissertation, Universität Bielefeld, 1998.

[Guzicki] Wojciech Guzicki, *Ulam's Searching Game with Two Lies* in *Journal of Combinatorial Theory*, Series A **54** (1990), pp. 1–19.

[Hill] Ray Hill, *Searching with Lies*, Department of Mathematics & Computer Science, University of Salford, Salford M5 4WT, England.

[Hill2] Ray Hill, J. P. Karim, and Elwyn R. Berlekamp, *The Solution of a problem of Ulam on searching with lies* in *Proc of IEEE International Symposium on Information Theory* (1998), p. 244.

[Karp] Richard M. Karp, *ISIT'98 Plenary Lecture Report: Variations on the Theme of 'Twenty Questions'* in *IEEE Information Theory Society Newsletter* **49** No. 1, edited by Kimberly Wasserman, March 1999, pp. 1–5, 21–22.

[Karim] Jehangir Pervaiz Karim, *Searching with Lies: The Ulam Problem*, doctoral dissertation, Unversity of Salford, 1999.

[Lawler] Eugene L. Lawler and Sergei Sarkissian, *An algorithm for "Ulam's Game" and its application to error correcting codes* in *Information Processing Letters* **56**, Elsevier (1995), pp. 89-93.

[Pelc] Andrzej Pelc, *Solution of Ulam's Problem on Searching with a Lie* in *Journal of Combinatorial Theory*, Series A **44** (1987), pp. 129–140.

[Piccione] Michele Piccione and Ariel Rubinstein, *On the Interpretation of Decision Problems with Imperfect Recall*, in *Games and Economic Behavior* **20**, Article No. GA970536, (1997), pp. 3–24.

[Spencer] Joel Spencer, *Ulam's searching game with a fixed number of lies* in *Theoretical Computer Science* **95**, Elsevier (1992), pp. 307–321.

[Spencer2] Joel Spencer, *Randomization, derandomization and antirandomization: three games* in *Theoretical Computer Science* **131**, Elsevier (1994), pp. 415–429.

[Ulam] *Adventures of a Mathematician*, University of California Press, Berkeley, CA, 1992.

Appendix A

Program Listing

```
#include <stdio.h>
#include <malloc.h>
#include <assert.h>
#include <values.h>
#include <math.h>

#define NMAX 126
#define EMAX 32

#define FEXP(n) (pow (2.0, (double) (n)))

typedef struct winstate {
  struct winstate *next;
  int ns;
  struct winstate *left, *right;
  int state[EMAX+1];
} winstate;

#define MAX(a,b) ((a) > (b) ? (a) : (b))
#define MIN(a,b) ((a) < (b) ? (a) : (b))

static double bico[128][128], wts[128][128];
static winstate *tree[NMAX+1];
static int mcurrent;
static int leafnodes, safeleaves;
static int treenodes, searchnodes;
```

```
int treesearch (int e, int n, int m);
int lifttree (int e, int n, int m);
winstate *depthsearch (int state[], int ns, int n);
void inittree (void);
winstate *addtree (int state[], int ns, int n,
   winstate *left, winstate *right);
winstate *cachetest (int state[], int ns, int n);
void cleartree (void);
int igcd (int m, int n);
void initwts (void);
double weight (int state[], int ns, int n);
int stateleq (int state1[], int ns1, int state2[], int ns2);
int mbound (int m, int e);
void copystate (int copy[], int *nc, int state[], int ns);
int leaftest (int state[], int *nsp, int n, int *safe);
int sumtest (int sum[], int ns,
     int state1[], int ns1, int state2[], int ns2);
void addstate (int sum[], int *nsp,
       int state1[], int ns1, int state2[], int ns2);

int main (int argc, char **argv) {
  int win;
  int e, k, m, n;

  setvbuf (stdout, NULL, _IOLBF, BUFSIZ);
  setvbuf (stderr, NULL, _IOLBF, BUFSIZ);

  initwts ();
  inittree ();

  for (k=2; k<=25; k++) {
    for (e=1; e<=EMAX; e++) {
      n = 3 * e + k;
      printf ("\n");
      for (m = mbound(n,e); m > 0; m--) {
win = treesearch (e, n, m);
if (win) break;
      }
      if (m <= mbound(n+3, e+1)) {
win = lifttree(e, n, m);
if (win) break;
      }
    }
  }

  printf ("\n");
  (void) treesearch (10, 56, 1048577);
  return 0;
}


int treesearch (int e, int n, int m) {
  int state[EMAX+1], ns;
  winstate *wp;
  int j;

  cleartree();

  ns = e + 1;
  for (j=0; j<e; j++)
    state[j] = 0;
  state[e] = m;
  printf ("E=%d N=%d M=%d: ", e, n, m);

  mcurrent = m;
  leafnodes = safeleaves = 0;
  wp = depthsearch (state, ns, n);

  if (wp != 0) {
    printf ("OK, %d tree nodes, %d leaf nodes (%d safe).\n",
    treenodes, leafnodes, safeleaves);
    return 1;
  } else {
    printf ("Failed, %d search nodes.\n", searchnodes);
    return 0;
  }
}


int lifttree (int e, int n, int m) {
  int state[EMAX+1];
  winstate *wp;
  int safe;
  int ns, nn;
  while (leafnodes > safeleaves) {
    e += 1;
    n += 3;

    printf ("E=%d N=%d M=%d: ", e, n, m);

    leafnodes = safeleaves = 0;

    for (nn=n; nn>=3; nn--) {
      tree[nn] = tree[nn-3];
      tree[nn-3] = 0;
    }
    for (nn=3; nn<=n; nn++) {
      for (wp=tree[nn]; wp!=0; wp=wp->next) {
state[0] = 0;
copystate (state+1, &ns, wp->state, wp->ns);
ns++;
if (wp->left == 0) {
  if (! leaftest (state, &ns, nn, &safe))
    goto FAIL;
  leafnodes++;
  if (safe) safeleaves++;
} else {
  if (! sumtest (state, ns, wp->left->state, wp->left->ns,
 wp->right->state, wp->right->ns)) {
    state[0] = 1;
    ns = 1;
  }
  addstate (state, &ns, wp->left->state, wp->left->ns,
    wp->right->state, wp->right->ns);
}
copystate (wp->state, &wp->ns, state, ns);
      }
    }
    if (tree[n]->state[e] != m) {
    FAIL:
      printf ("lift failed.\n");
      return 0;
    }
    printf ("lift succeeded: %d leaf nodes (%d safe).\n",
    leafnodes, safeleaves);
  }

  return (leafnodes == safeleaves);
}


winstate *depthsearch (int state[], int ns, int n) {
  int cut[EMAX+1], left[EMAX+1], right[EMAX+1], bias[EMAX+1];
  int temp[EMAX+1], nr, nt;
  double z, v, w, cmin, cmax;
  winstate *wp, *wpr, *wpl;
  int j, jj, k, nn, biaslvl, safe;

  searchnodes++;

  /* Cache check */
  wp = cachetest (state, ns, n);
  if (wp != 0) return wp;

  /* Check for leaf nodes */
  copystate (temp, &nt, state, ns);
  if (leaftest (temp, &nt, n, &safe)) {
    leafnodes++;
    if (safe) safeleaves++;
    return addtree (temp, nt, n, 0, 0);
  }

  for (j=0; j<ns; j++)
    bias[j] = 0;
  biaslvl = ns-1;
  goto TRY;

 RETRY:
  if (bias[biaslvl] < 0)
    bias[biaslvl] = -bias[biaslvl];
  else
    bias[biaslvl] = -bias[biaslvl] - 1;
  if (bias[biaslvl] <= -3) {
    bias[biaslvl] = 0;
    do {
      biaslvl--;
      if (biaslvl == 0) return 0;
    } while (state[biaslvl] == 0);
    bias[biaslvl] = -1;
  }

TRY:
  for (j=0; j<ns; j++)
    left[j] = right[j] = 0;
  k = 0;

  for (j=ns-1; j>=0; j--) {
    if (state[j] == 0) {
      cut[j] = 0;
      continue;
    }

    k += state[j];
    if (k == 1) {
      if (bias[j] != 0) goto RETRY;
      cut[j] = 0;
      goto CUT;
    }
    if (k == 2) {
      cut[j] = 1 + bias[j];
      if (cut[j] < 0 || cut[j] > state[j]) goto RETRY;
      goto CUT;
    }

    cmin = -MAXINT;
    cmax = MAXINT;
    for (jj=j; jj>=0; jj--) {
      nn = n - 3*jj - 1;
      /* Projected left gap if cut[0..j] == state[0..j] */
      v = FEXP (nn) - weight (left+jj, ns-jj, nn) -
        weight (state+jj+1, j-jj, nn);
      /* Projected right gap if cut[0..j] == 0 */
      w = FEXP (nn) - weight (right+jj, ns-jj, nn) -
        weight (state+jj+1, j-jj, nn);
      if (v < 0.0 || w < 0.0) goto RETRY;
      cmin = MAX (cmin, (state[j] - v / bico[nn][j-jj]));
      cmax = MIN (cmax, w / bico[nn][j-jj]);
    }
    if (cmin > cmax)
      goto RETRY;
    z = floor (0.5 * (cmin + cmax + 1));
    z = MAX (z, 0.0);
    z = MIN (z, state[j]);
    cut[j] = z + bias[j];
    if (cut[j] < cmin || cut[j] > cmax) goto RETRY;

  CUT:
    if (cut[j] < 0 || cut[j] > state[j]) goto RETRY;
    left[j] += state[j] - cut[j];
    right[j] += cut[j];
    if (j > 0) {
      left[j-1] += cut[j];
      right[j-1] += state[j] - cut[j];
    }
  }

  nr = ns;
  while (nr > 0 && right[nr-1] == 0)
    nr--;

  wpl = depthsearch (left, ns, n-1);
  if (wpl == 0) goto RETRY;

  wpr = depthsearch (right, nr, n-1);
  if (wpr == 0) goto RETRY;

  copystate (temp, &nt, state, ns);
  addstate (temp, &nt, wpl->state, wpl->ns, wpr->state, wpr->ns);
  return addtree (temp, nt, n, wpl, wpr);
}


void inittree () {
  int n;

  for (n=0; n<=NMAX; n++)
    tree[n] = 0;
  treenodes = 0;
}
```

```
winstate *addtree
(int state[], int ns, int n, winstate *left, winstate *right) {
  winstate *wp;
  int j;

  for (wp=tree[n]; wp!=0; wp=wp->next)
    if (stateleq (state, ns, wp->state, wp->ns))
      return wp;

  treenodes++;
  wp = tree[n];
  tree[n] = (winstate *) malloc (sizeof (winstate));
  tree[n]->next = wp;
  tree[n]->ns = ns;
  for (j=0; j<ns; j++)
    tree[n]->state[j] = state[j];
  tree[n]->left = left;
  tree[n]->right = right;

  return tree[n];
}


winstate *cachetest (int state[], int ns, int n) {
  winstate *wp;

  wp = tree[n];
  while (wp != 0) {
    if (stateleq (state, ns, wp->state, wp->ns))
      return wp;
    wp = wp->next;
  }
  return 0;
}


void cleartree () {
  int n;
  winstate *wp, *wpnext;

  for (n=0; n<=NMAX; n++) {
    wp = tree[n];
    while (wp != 0) {
      wpnext = wp->next;
      free (wp);
      wp = wpnext;
    }
    tree[n] = 0;
  }
  treenodes = 0;
  searchnodes = 0;
}


int igcd (int m, int n) {
  if (m < 0) m = -m;
  if (n < 0) n = -n;
  if (m > n) return igcd (n, m);
  if (m == 0) return n;
  return igcd (n % m, m);
}


void initwts (void) {
  int g, j, n;

  for (n=0; n<=NMAX; n++) {
    bico[n][0] = 1.0;
    wts[n][0] = 1.0;
    for (j=1; j<=n; j++) {
      g = igcd (n-j+1, j);
      bico[n][j] = (bico[n][j-1] / (j/g)) * ((n-j+1) / g);
      wts[n][j] = wts[n][j-1] + bico[n][j];
    }
  }
}


double weight (int state[], int ns, int n) {
  double w;
  int j;

  w = 0.0;
  for (j=0; j<ns; j++) {
    w += state[j] * wts[n][j];
  }
  return w;
}


int stateleq (int state1[], int ns1, int state2[], int ns2) {
  int j, k1, k2;

  if (ns1 > ns2)
    return 0;
  k1 = k2 = 0;
  for (j=ns2-1; j>ns1-1; j--)
    k2 += state2[j];
  for (j=ns1-1; j>=0; j--) {
    k1 += state1[j];
    k2 += state2[j];
    if (k2 >= mcurrent) return 1;
    if (k1 > k2) return 0;
  }
  return 1;
}


int mbound (int n, int e) {
  int j, m;
  double z, zz;

  if (n < 2*e+1) {
    m = 1;
  } else if (n <= 3*e) {
    m = 2;
  } else {
    z = FEXP(n) / wts[n][e];
    for (j=1; j<=e; j++) {
      zz = FEXP(n-3*j) / wts[n-3*j][e-j];
      z = MIN (z, zz);
    }
    m = floor(z);
  }
  return m;
}


void copystate (int copy[], int *nc, int state[], int ns) {
  int i;

  *nc = ns;
  for (i=0; i<ns; i++)
    copy[i] = state[i];
}


int leaftest (int state[], int *nsp, int n, int *safe) {
  int leafstate[EMAX+2];
  double w;
  int i, k, nn, ns;

  ns = *nsp;
  nn = n - 3 * (ns-1);
  if (nn > 4)
    return 0;

  leafstate[ns] = (nn >= 0) ? (1 << nn) : 1;
  for (i=0; i<ns; i++)
    leafstate[i] = 0;

  k = leafstate[ns];
  for (i=1; i<ns; i++) {
    nn += 3;
    if (nn < i+1) continue;
    w = FEXP(nn) - weight (leafstate+(ns-i), i+1, nn);
    if (w >= mcurrent-k) {
      leafstate[ns-i] = mcurrent - k;
      k = mcurrent;
      break;
    }
    leafstate[ns-i] = w;
    k += leafstate[ns-i];
  }

  if (! stateleq (state, ns, leafstate+1, ns))
```

```
    return 0;

  copystate (state, nsp, leafstate+1, ns);
  *safe = (k >= mcurrent);
  return 1;
}


int sumtest
(int sum[], int ns, int state1[], int ns1, int state2[], int ns2) {
  int accum, accum1, accum2;
  int i;

  if (ns1 < ns-1 || ns2 < ns-1) return 0;

  accum = sum[ns-1];
  accum1 = 0;
  for (i=ns-1; i<ns1; i++)
    accum1 += state1[i];
  accum2 = 0;
  for (i=ns-1; i<ns2; i++)
    accum2 += state2[i];
  if (accum > accum1 + accum2) return 0;

  for (i=ns-2; i>=0; i--) {
    accum1 += state1[i];
    accum2 += state2[i];
    if (accum > accum1) return 0;
    if (accum > accum2) return 0;
    if (sum[i] + 2*accum > accum1 + accum2) return 0;
    accum += sum[i];
  }
  return 1;
}


void addstate
(int sum[], int *nsp, int state1[], int ns1, int state2[], int ns2) {
  int accum[EMAX+1], accum1[EMAX+1], accum2[EMAX+1];
  int ns, i, j, k;

  /* Ensure that output(sum[]) .ge. input(sum[]). */
  ns = *nsp;
  ns = MAX (ns, ns1);
  ns = MAX (ns, ns2);
  accum[ns] = accum1[ns] = accum2[ns] = 0;
  for (i=ns-1; i>=0; i--) {
    accum[i] = accum[i+1] + (i < *nsp ? sum[i] : 0);
    accum1[i] = accum1[i+1] + (i < ns1 ? state1[i] : 0);
    accum2[i] = accum2[i+1] + (i < ns2 ? state2[i] : 0);
  }
  for (j=ns-1; j>=0; j--) {
    k = mcurrent;
    k = MIN (k, accum1[j] + accum2[j] - accum[j+1]);
    if (j > 0) k = MIN (k, accum1[j-1]);
    if (j > 0) k = MIN (k, accum2[j-1]);
    for (i=0; i<j; i++)
      k = MIN (k, accum1[i] + accum2[i] - accum[i]);
    accum[j] = k;
  }

  while (ns > 0 && accum[ns-1] == 0)
    ns--;
  *nsp = ns;
  sum[ns-1] = accum[ns-1];
  for (j=ns-2; j>=0; j--) {
    accum[j] = MIN (accum[j], mcurrent);
    sum[j] = accum[j] - accum[j+1];
  }
  assert (sumtest (sum, ns, state1, ns1, state2, ns2));
}
```

Appendix B

Program Output

```
E=1 N=5 M=4: OK, 1 tree nodes, 1 leaf nodes (1 safe).

E=1 N=6 M=8: OK, 1 tree nodes, 1 leaf nodes (1 safe).

E=1 N=7 M=16: OK, 1 tree nodes, 1 leaf nodes (1 safe).

E=1 N=8 M=28: OK, 9 tree nodes, 4 leaf nodes (0 safe).
E=2 N=11 M=28: lift succeeded: 4 leaf nodes (2 safe).
E=3 N=14 M=28: lift succeeded: 4 leaf nodes (4 safe).

E=1 N=9 M=51: Failed, 1 search nodes.
E=1 N=9 M=50: OK, 12 tree nodes, 4 leaf nodes (0 safe).
E=2 N=12 M=50: lift failed.

E=2 N=12 M=51: Failed, 1 search nodes.
E=2 N=12 M=50: OK, 22 tree nodes, 8 leaf nodes (0 safe).
E=3 N=15 M=50: lift succeeded: 8 leaf nodes (4 safe).
E=4 N=18 M=50: lift succeeded: 8 leaf nodes (8 safe).

E=1 N=10 M=93: Failed, 1 search nodes.
E=1 N=10 M=92: OK, 13 tree nodes, 3 leaf nodes (0 safe).

E=2 N=13 M=89: Failed, 1 search nodes.
E=2 N=13 M=88: OK, 25 tree nodes, 8 leaf nodes (0 safe).
E=3 N=16 M=88: lift succeeded: 8 leaf nodes (1 safe).
E=4 N=19 M=88: lift succeeded: 8 leaf nodes (6 safe).
E=5 N=22 M=88: lift succeeded: 8 leaf nodes (8 safe).
```

```
E=1 N=11 M=170: OK, 16 tree nodes, 3 leaf nodes (0 safe).

E=2 N=14 M=154: OK, 31 tree nodes, 7 leaf nodes (0 safe).
E=3 N=17 M=154: lift succeeded: 7 leaf nodes (0 safe).
E=4 N=20 M=154: lift succeeded: 7 leaf nodes (3 safe).
E=5 N=23 M=154: lift succeeded: 7 leaf nodes (7 safe).

E=1 N=12 M=315: Failed, 1 search nodes.
E=1 N=12 M=314: OK, 18 tree nodes, 3 leaf nodes (0 safe).

E=2 N=15 M=270: OK, 34 tree nodes, 6 leaf nodes (0 safe).

E=3 N=18 M=265: Failed, 1 search nodes.
E=3 N=18 M=264: OK, 54 tree nodes, 9 leaf nodes (0 safe).
E=4 N=21 M=264: lift succeeded: 9 leaf nodes (0 safe).
E=5 N=24 M=264: lift succeeded: 9 leaf nodes (4 safe).
E=6 N=27 M=264: lift succeeded: 9 leaf nodes (9 safe).

E=1 N=13 M=585: Failed, 1 search nodes.
E=1 N=13 M=584: OK, 18 tree nodes, 3 leaf nodes (0 safe).

E=2 N=16 M=478: OK, 39 tree nodes, 5 leaf nodes (0 safe).

E=3 N=19 M=451: OK, 116 tree nodes, 9 leaf nodes (0 safe).
E=4 N=22 M=451: lift succeeded: 9 leaf nodes (0 safe).
E=5 N=25 M=451: lift succeeded: 9 leaf nodes (1 safe).
E=6 N=28 M=451: lift succeeded: 9 leaf nodes (7 safe).
E=7 N=31 M=451: lift succeeded: 9 leaf nodes (9 safe).

E=1 N=14 M=1092: OK, 21 tree nodes, 3 leaf nodes (0 safe).

E=2 N=17 M=851: Failed, 1 search nodes.
E=2 N=17 M=850: OK, 40 tree nodes, 5 leaf nodes (0 safe).

E=3 N=20 M=776: OK, 67 tree nodes, 9 leaf nodes (0 safe).

E=4 N=23 M=769: Failed, 1 search nodes.
E=4 N=23 M=768: OK, 74 tree nodes, 13 leaf nodes (0 safe).
E=5 N=26 M=768: lift succeeded: 13 leaf nodes (0 safe).
E=6 N=29 M=768: lift succeeded: 13 leaf nodes (3 safe).
E=7 N=32 M=768: lift succeeded: 13 leaf nodes (10 safe).
```

```
E=8 N=35 M=768: lift succeeded: 13 leaf nodes (13 safe).

E=1 N=15 M=2048: OK, 14 tree nodes, 2 leaf nodes (0 safe).

E=2 N=18 M=1524: OK, 45 tree nodes, 5 leaf nodes (0 safe).

E=3 N=21 M=1342: OK, 86 tree nodes, 8 leaf nodes (0 safe).

E=4 N=24 M=1295: Failed, 1 search nodes.
E=4 N=24 M=1294: OK, 131 tree nodes, 11 leaf nodes (0 safe).
E=5 N=27 M=1294: lift succeeded: 11 leaf nodes (0 safe).
E=6 N=30 M=1294: lift succeeded: 11 leaf nodes (0 safe).
E=7 N=33 M=1294: lift succeeded: 11 leaf nodes (6 safe).
E=8 N=36 M=1294: lift succeeded: 11 leaf nodes (11 safe).

E=1 N=16 M=3855: Failed, 1 search nodes.
E=1 N=16 M=3854: OK, 26 tree nodes, 3 leaf nodes (0 safe).

E=2 N=19 M=2744: OK, 47 tree nodes, 4 leaf nodes (0 safe).

E=3 N=22 M=2337: OK, 122 tree nodes, 7 leaf nodes (0 safe).

E=4 N=25 M=2196: OK, 151 tree nodes, 10 leaf nodes (0 safe).

E=5 N=28 M=2192: OK, 331 tree nodes, 15 leaf nodes (0 safe).
E=6 N=31 M=2192: lift succeeded: 15 leaf nodes (0 safe).
E=7 N=34 M=2192: lift succeeded: 15 leaf nodes (1 safe).
E=8 N=37 M=2192: lift succeeded: 15 leaf nodes (8 safe).
E=9 N=40 M=2192: lift succeeded: 15 leaf nodes (13 safe).
E=10 N=43 M=2192: lift succeeded: 15 leaf nodes (15 safe).

E=1 N=17 M=7281: Failed, 1 search nodes.
E=1 N=17 M=7280: OK, 25 tree nodes, 3 leaf nodes (0 safe).

E=2 N=20 M=4969: Failed, 1 search nodes.
E=2 N=20 M=4968: OK, 46 tree nodes, 4 leaf nodes (0 safe).

E=3 N=23 M=4096: OK, 43 tree nodes, 6 leaf nodes (0 safe).

E=4 N=26 M=3748: OK, 161 tree nodes, 9 leaf nodes (0 safe).

E=5 N=29 M=3662: OK, 276 tree nodes, 13 leaf nodes (0 safe).
```

```
E=6 N=32 M=3662: lift succeeded: 13 leaf nodes (0 safe).
E=7 N=35 M=3662: lift succeeded: 13 leaf nodes (0 safe).
E=8 N=38 M=3662: lift succeeded: 13 leaf nodes (1 safe).
E=9 N=41 M=3662: lift succeeded: 13 leaf nodes (8 safe).
E=10 N=44 M=3662: lift succeeded: 13 leaf nodes (13 safe).

E=1 N=18 M=13797: Failed, 1 search nodes.
E=1 N=18 M=13796: OK, 29 tree nodes, 3 leaf nodes (0 safe).

E=2 N=21 M=9039: Failed, 1 search nodes.
E=2 N=21 M=9038: OK, 57 tree nodes, 4 leaf nodes (0 safe).

E=3 N=24 M=7216: OK, 86 tree nodes, 6 leaf nodes (0 safe).

E=4 N=27 M=6436: OK, 171 tree nodes, 9 leaf nodes (0 safe).

E=5 N=30 M=6155: Failed, 1 search nodes.
E=5 N=30 M=6154: OK, 276 tree nodes, 12 leaf nodes (0 safe).
E=6 N=33 M=6154: lift succeeded: 12 leaf nodes (0 safe).
E=7 N=36 M=6154: lift succeeded: 12 leaf nodes (0 safe).
E=8 N=39 M=6154: lift succeeded: 12 leaf nodes (0 safe).
E=9 N=42 M=6154: lift succeeded: 12 leaf nodes (3 safe).
E=10 N=45 M=6154: lift succeeded: 12 leaf nodes (10 safe).
E=11 N=48 M=6154: lift succeeded: 12 leaf nodes (12 safe).

E=1 N=19 M=26214: OK, 32 tree nodes, 3 leaf nodes (0 safe).

E=2 N=22 M=16513: Failed, 1 search nodes.
E=2 N=22 M=16512: OK, 40 tree nodes, 4 leaf nodes (0 safe).

E=3 N=25 M=12777: Failed, 224 search nodes.
E=3 N=25 M=12776: OK, 100 tree nodes, 6 leaf nodes (0 safe).

E=4 N=28 M=11111: Failed, 1 search nodes.
E=4 N=28 M=11110: OK, 180 tree nodes, 9 leaf nodes (0 safe).

E=5 N=31 M=10406: OK, 297 tree nodes, 11 leaf nodes (0 safe).

E=6 N=34 M=10249: OK, 1369 tree nodes, 16 leaf nodes (0 safe).
E=7 N=37 M=10249: lift succeeded: 16 leaf nodes (0 safe).
E=8 N=40 M=10249: lift succeeded: 16 leaf nodes (0 safe).
E=9 N=43 M=10249: lift succeeded: 16 leaf nodes (0 safe).
```

```
E=10 N=46 M=10249: lift succeeded: 16 leaf nodes (5 safe).
E=11 N=49 M=10249: lift succeeded: 16 leaf nodes (13 safe).
E=12 N=52 M=10249: lift succeeded: 16 leaf nodes (16 safe).

E=1 N=20 M=49932: OK, 33 tree nodes, 3 leaf nodes (0 safe).

E=2 N=23 M=30283: Failed, 1 search nodes.
E=2 N=23 M=30282: OK, 67 tree nodes, 5 leaf nodes (0 safe).

E=3 N=26 M=22733: Failed, 1 search nodes.
E=3 N=26 M=22732: OK, 101 tree nodes, 6 leaf nodes (0 safe).

E=4 N=29 M=19283: Failed, 1 search nodes.
E=4 N=29 M=19282: OK, 219 tree nodes, 9 leaf nodes (0 safe).

E=5 N=32 M=17687: Failed, 1 search nodes.
E=5 N=32 M=17686: OK, 359 tree nodes, 10 leaf nodes (0 safe).

E=6 N=35 M=17117: Failed, 1 search nodes.
E=6 N=35 M=17116: OK, 557 tree nodes, 14 leaf nodes (0 safe).
E=7 N=38 M=17116: lift succeeded: 14 leaf nodes (0 safe).
E=8 N=41 M=17116: lift succeeded: 14 leaf nodes (0 safe).
E=9 N=44 M=17116: lift failed.

E=7 N=38 M=17117: Failed, 1 search nodes.
E=7 N=38 M=17116: OK, 572 tree nodes, 21 leaf nodes (0 safe).
E=8 N=41 M=17116: lift succeeded: 21 leaf nodes (0 safe).
E=9 N=44 M=17116: lift failed.

E=8 N=41 M=17117: Failed, 1 search nodes.
E=8 N=41 M=17116: OK, 572 tree nodes, 21 leaf nodes (0 safe).
E=9 N=44 M=17116: lift failed.

E=9 N=44 M=17117: Failed, 1 search nodes.
E=9 N=44 M=17116: OK, 676 tree nodes, 21 leaf nodes (0 safe).
E=10 N=47 M=17116: lift succeeded: 21 leaf nodes (0 safe).
E=11 N=50 M=17116: lift succeeded: 21 leaf nodes (8 safe).
E=12 N=53 M=17116: lift succeeded: 21 leaf nodes (17 safe).
E=13 N=56 M=17116: lift succeeded: 21 leaf nodes (21 safe).

E=1 N=21 M=95325: Failed, 1 search nodes.
E=1 N=21 M=95324: OK, 35 tree nodes, 3 leaf nodes (0 safe).
```

```
E=2 N=24 M=55738: OK, 68 tree nodes, 5 leaf nodes (0 safe).

E=3 N=27 M=40622: OK, 131 tree nodes, 6 leaf nodes (0 safe).

E=4 N=30 M=33626: OK, 206 tree nodes, 7 leaf nodes (0 safe).

E=5 N=33 M=30217: Failed, 1 search nodes.
E=5 N=33 M=30216: OK, 345 tree nodes, 10 leaf nodes (0 safe).

E=6 N=36 M=28734: OK, 706 tree nodes, 13 leaf nodes (0 safe).

E=7 N=39 M=28467: OK, 4410 tree nodes, 19 leaf nodes (0 safe).
E=8 N=42 M=28467: lift succeeded: 19 leaf nodes (0 safe).
E=9 N=45 M=28467: lift succeeded: 19 leaf nodes (0 safe).
E=10 N=48 M=28467: lift succeeded: 19 leaf nodes (0 safe).
E=11 N=51 M=28467: lift succeeded: 19 leaf nodes (1 safe).
E=12 N=54 M=28467: lift succeeded: 19 leaf nodes (11 safe).
E=13 N=57 M=28467: lift succeeded: 19 leaf nodes (17 safe).
E=14 N=60 M=28467: lift succeeded: 19 leaf nodes (19 safe).

E=1 N=22 M=182361: Failed, 1 search nodes.
E=1 N=22 M=182360: OK, 36 tree nodes, 3 leaf nodes (0 safe).

E=2 N=25 M=102927: Failed, 1 search nodes.
E=2 N=25 M=102926: OK, 70 tree nodes, 5 leaf nodes (0 safe).

E=3 N=28 M=72885: Failed, 1 search nodes.
E=3 N=28 M=72884: OK, 113 tree nodes, 5 leaf nodes (0 safe).

E=4 N=31 M=58904: OK, 206 tree nodes, 7 leaf nodes (0 safe).

E=5 N=34 M=51869: Failed, 1 search nodes.
E=5 N=34 M=51868: OK, 404 tree nodes, 10 leaf nodes (0 safe).

E=6 N=37 M=48475: Failed, 2212 search nodes.
E=6 N=37 M=48474: OK, 584 tree nodes, 13 leaf nodes (0 safe).

E=7 N=40 M=47307: Failed, 1 search nodes.
E=7 N=40 M=47306: OK, 1071 tree nodes, 16 leaf nodes (0 safe).
E=8 N=43 M=47306: lift succeeded: 16 leaf nodes (0 safe).
E=9 N=46 M=47306: lift succeeded: 16 leaf nodes (0 safe).
```

```
E=10 N=49 M=47306: lift failed.

E=8 N=43 M=47307: Failed, 1 search nodes.
E=8 N=43 M=47306: OK, 1071 tree nodes, 16 leaf nodes (0 safe).
E=9 N=46 M=47306: lift succeeded: 16 leaf nodes (0 safe).
E=10 N=49 M=47306: lift failed.

E=9 N=46 M=47307: Failed, 1 search nodes.
E=9 N=46 M=47306: OK, 1071 tree nodes, 16 leaf nodes (0 safe).
E=10 N=49 M=47306: lift failed.

E=10 N=49 M=47307: Failed, 1 search nodes.
E=10 N=49 M=47306: OK, 1574 tree nodes, 16 leaf nodes (0 safe).
E=11 N=52 M=47306: lift succeeded: 16 leaf nodes (0 safe).
E=12 N=55 M=47306: lift succeeded: 16 leaf nodes (4 safe).
E=13 N=58 M=47306: lift succeeded: 16 leaf nodes (12 safe).
E=14 N=61 M=47306: lift succeeded: 16 leaf nodes (16 safe).

E=1 N=23 M=349525: Failed, 1 search nodes.
E=1 N=23 M=349524: OK, 39 tree nodes, 3 leaf nodes (0 safe).

E=2 N=26 M=190650: OK, 77 tree nodes, 5 leaf nodes (0 safe).

E=3 N=29 M=131264: OK, 105 tree nodes, 6 leaf nodes (0 safe).

E=4 N=32 M=103620: OK, 214 tree nodes, 7 leaf nodes (0 safe).

E=5 N=35 M=89439: Failed, 1 search nodes.
E=5 N=35 M=89438: OK, 494 tree nodes, 10 leaf nodes (0 safe).

E=6 N=38 M=82160: OK, 605 tree nodes, 12 leaf nodes (0 safe).

E=7 N=41 M=78986: OK, 1336 tree nodes, 15 leaf nodes (0 safe).

E=8 N=44 M=78589: OK, 10920 tree nodes, 23 leaf nodes (0 safe).
E=9 N=47 M=78589: lift succeeded: 23 leaf nodes (0 safe).
E=10 N=50 M=78589: lift failed.

E=9 N=47 M=78589: OK, 10920 tree nodes, 23 leaf nodes (0 safe).
E=10 N=50 M=78589: lift failed.

E=10 N=50 M=78589: OK, 17324 tree nodes, 23 leaf nodes (0 safe).
```

```
E=11 N=53 M=78589: lift succeeded: 23 leaf nodes (0 safe).
E=12 N=56 M=78589: lift succeeded: 23 leaf nodes (0 safe).
E=13 N=59 M=78589: lift succeeded: 23 leaf nodes (8 safe).
E=14 N=62 M=78589: lift succeeded: 23 leaf nodes (18 safe).
E=15 N=65 M=78589: lift succeeded: 23 leaf nodes (23 safe).

E=1 N=24 M=671088: OK, 39 tree nodes, 3 leaf nodes (0 safe).

E=2 N=27 M=354136: OK, 76 tree nodes, 5 leaf nodes (0 safe).

E=3 N=30 M=237238: OK, 124 tree nodes, 6 leaf nodes (0 safe).

E=4 N=33 M=183005: OK, 434 tree nodes, 8 leaf nodes (0 safe).

E=5 N=36 M=154876: OK, 361 tree nodes, 10 leaf nodes (0 safe).

E=6 N=39 M=139867: Failed, 1 search nodes.
E=6 N=39 M=139866: OK, 812 tree nodes, 11 leaf nodes (0 safe).

E=7 N=42 M=132474: OK, 1135 tree nodes, 14 leaf nodes (0 safe).

E=8 N=45 M=130088: OK, 2050 tree nodes, 20 leaf nodes (0 safe).
E=9 N=48 M=130088: lift succeeded: 20 leaf nodes (0 safe).
E=10 N=51 M=130088: lift succeeded: 20 leaf nodes (0 safe).
E=11 N=54 M=130088: lift failed.

E=9 N=48 M=130088: OK, 2060 tree nodes, 28 leaf nodes (0 safe).
E=10 N=51 M=130088: lift succeeded: 28 leaf nodes (0 safe).
E=11 N=54 M=130088: lift failed.

E=10 N=51 M=130088: OK, 2060 tree nodes, 28 leaf nodes (0 safe).
E=11 N=54 M=130088: lift failed.

E=11 N=54 M=130088: OK, 3013 tree nodes, 28 leaf nodes (0 safe).
E=12 N=57 M=130088: lift succeeded: 28 leaf nodes (0 safe).
E=13 N=60 M=130088: lift succeeded: 28 leaf nodes (3 safe).
E=14 N=63 M=130088: lift succeeded: 28 leaf nodes (13 safe).
E=15 N=66 M=130088: lift succeeded: 28 leaf nodes (21 safe).
E=16 N=69 M=130088: lift succeeded: 28 leaf nodes (26 safe).
E=17 N=72 M=130088: lift succeeded: 28 leaf nodes (28 safe).

E=1 N=25 M=1290555: Failed, 1 search nodes.
```

```
E=1 N=25 M=1290554: OK, 44 tree nodes, 3 leaf nodes (0 safe).

E=2 N=28 M=659546: OK, 82 tree nodes, 5 leaf nodes (0 safe).

E=3 N=31 M=430185: Failed, 1 search nodes.
E=3 N=31 M=430184: OK, 129 tree nodes, 6 leaf nodes (0 safe).

E=4 N=34 M=324417: Failed, 558 search nodes.
E=4 N=34 M=324416: OK, 224 tree nodes, 8 leaf nodes (0 safe).

E=5 N=37 M=269268: OK, 457 tree nodes, 10 leaf nodes (0 safe).

E=6 N=40 M=239103: Failed, 1 search nodes.
E=6 N=40 M=239102: OK, 868 tree nodes, 11 leaf nodes (0 safe).

E=7 N=43 M=223138: OK, 1236 tree nodes, 14 leaf nodes (0 safe).

E=8 N=46 M=216269: OK, 8760 tree nodes, 17 leaf nodes (0 safe).

E=9 N=49 M=215900: OK, 18900 tree nodes, 30 leaf nodes (0 safe).
E=10 N=52 M=215900: lift succeeded: 30 leaf nodes (0 safe).
E=11 N=55 M=215900: lift failed.

E=10 N=52 M=215900: OK, 18900 tree nodes, 30 leaf nodes (0 safe).
E=11 N=55 M=215900: lift failed.

E=11 N=55 M=215900: OK, 34232 tree nodes, 30 leaf nodes (0 safe).
E=12 N=58 M=215900: lift succeeded: 30 leaf nodes (0 safe).
E=13 N=61 M=215900: lift succeeded: 30 leaf nodes (0 safe).
E=14 N=64 M=215900: lift succeeded: 30 leaf nodes (6 safe).
E=15 N=67 M=215900: lift succeeded: 30 leaf nodes (18 safe).
E=16 N=70 M=215900: lift succeeded: 30 leaf nodes (26 safe).
E=17 N=73 M=215900: lift succeeded: 30 leaf nodes (30 safe).

E=1 N=26 M=2485513: Failed, 1 search nodes.
E=1 N=26 M=2485512: OK, 44 tree nodes, 3 leaf nodes (0 safe).

E=2 N=29 M=1231355: Failed, 1 search nodes.
E=2 N=29 M=1231354: OK, 79 tree nodes, 5 leaf nodes (0 safe).

E=3 N=32 M=782468: OK, 136 tree nodes, 6 leaf nodes (0 safe).
```

```
E=4 N=35 M=577125: Failed, 1 search nodes.
E=4 N=35 M=577124: OK, 241 tree nodes, 8 leaf nodes (0 safe).

E=5 N=38 M=469928: OK, 409 tree nodes, 10 leaf nodes (0 safe).

E=6 N=41 M=410374: OK, 893 tree nodes, 12 leaf nodes (0 safe).

E=7 N=44 M=377388: OK, 1554 tree nodes, 15 leaf nodes (0 safe).

E=8 N=47 M=361035: Failed, 1 search nodes.
E=8 N=47 M=361034: OK, 2285 tree nodes, 18 leaf nodes (0 safe).

E=9 N=50 M=356245: OK, 15868 tree nodes, 26 leaf nodes (0 safe).
E=10 N=53 M=356245: lift failed.

E=10 N=53 M=356245: OK, 15908 tree nodes, 38 leaf nodes (0 safe).
E=11 N=56 M=356245: lift succeeded: 38 leaf nodes (0 safe).
E=12 N=59 M=356245: lift failed.

E=11 N=56 M=356245: OK, 15908 tree nodes, 38 leaf nodes (0 safe).
E=12 N=59 M=356245: lift failed.

E=12 N=59 M=356245: OK, 23370 tree nodes, 38 leaf nodes (0 safe).
E=13 N=62 M=356245: lift succeeded: 38 leaf nodes (0 safe).
E=14 N=65 M=356245: lift succeeded: 38 leaf nodes (1 safe).
E=15 N=68 M=356245: lift succeeded: 38 leaf nodes (12 safe).
E=16 N=71 M=356245: lift succeeded: 38 leaf nodes (25 safe).
E=17 N=74 M=356245: lift succeeded: 38 leaf nodes (34 safe).
E=18 N=77 M=356245: lift succeeded: 38 leaf nodes (38 safe).

E=1 N=27 M=4793490: OK, 48 tree nodes, 3 leaf nodes (0 safe).

E=2 N=30 M=2304167: Failed, 1 search nodes.
E=2 N=30 M=2304166: OK, 92 tree nodes, 5 leaf nodes (0 safe).

E=3 N=33 M=1427373: Failed, 1 search nodes.
E=3 N=33 M=1427372: OK, 150 tree nodes, 6 leaf nodes (0 safe).

E=4 N=36 M=1030091: Failed, 1 search nodes.
E=4 N=36 M=1030090: OK, 294 tree nodes, 8 leaf nodes (0 safe).

E=5 N=39 M=823076: OK, 486 tree nodes, 9 leaf nodes (0 safe).
```

```
E=6 N=42 M=706994: OK, 1134 tree nodes, 12 leaf nodes (0 safe).

E=7 N=45 M=640756: OK, 1380 tree nodes, 15 leaf nodes (0 safe).

E=8 N=48 M=605094: OK, 2386 tree nodes, 18 leaf nodes (0 safe).

E=9 N=51 M=590174: OK, 4145 tree nodes, 23 leaf nodes (0 safe).
E=10 N=54 M=590174: lift failed.

E=10 N=54 M=590174: OK, 4034 tree nodes, 36 leaf nodes (0 safe).
E=11 N=57 M=590174: lift succeeded: 36 leaf nodes (0 safe).
E=12 N=60 M=590174: lift failed.

E=11 N=57 M=590174: OK, 4034 tree nodes, 36 leaf nodes (0 safe).
E=12 N=60 M=590174: lift failed.

E=12 N=60 M=590174: OK, 8544 tree nodes, 36 leaf nodes (0 safe).
E=13 N=63 M=590174: lift succeeded: 36 leaf nodes (0 safe).
E=14 N=66 M=590174: lift failed.

E=13 N=63 M=590174: OK, 9471 tree nodes, 36 leaf nodes (0 safe).
E=14 N=66 M=590174: lift failed.

E=14 N=66 M=590174: OK, 11852 tree nodes, 36 leaf nodes (0 safe).
E=15 N=69 M=590174: lift succeeded: 36 leaf nodes (5 safe).
E=16 N=72 M=590174: lift succeeded: 36 leaf nodes (19 safe).
E=17 N=75 M=590174: lift succeeded: 36 leaf nodes (29 safe).
E=18 N=78 M=590174: lift succeeded: 36 leaf nodes (34 safe).
E=19 N=81 M=590174: lift succeeded: 36 leaf nodes (36 safe).

E=1 N=28 M=9256395: Failed, 1 search nodes.
E=1 N=28 M=9256394: OK, 50 tree nodes, 3 leaf nodes (0 safe).

E=2 N=31 M=4320892: OK, 94 tree nodes, 5 leaf nodes (0 safe).

E=3 N=34 M=2610922: OK, 161 tree nodes, 6 leaf nodes (0 safe).

E=4 N=37 M=1844347: Failed, 532 search nodes.
E=4 N=37 M=1844346: OK, 297 tree nodes, 8 leaf nodes (0 safe).

E=5 N=40 M=1446537: Failed, 1 search nodes.
```

```
E=5 N=40 M=1446536: OK, 444 tree nodes, 9 leaf nodes (0 safe).

E=6 N=43 M=1222401: Failed, 1 search nodes.
E=6 N=43 M=1222400: OK, 699 tree nodes, 12 leaf nodes (0 safe).

E=7 N=46 M=1091975: Failed, 1 search nodes.
E=7 N=46 M=1091974: OK, 1860 tree nodes, 15 leaf nodes (0 safe).

E=8 N=49 M=1017991: Failed, 1 search nodes.
E=8 N=49 M=1017990: OK, 2301 tree nodes, 18 leaf nodes (0 safe).

E=9 N=52 M=981465: Failed, 1 search nodes.
E=9 N=52 M=981464: OK, 4444 tree nodes, 23 leaf nodes (0 safe).

E=10 N=55 M=972164: OK, 14922 tree nodes, 33 leaf nodes (0 safe).
E=11 N=58 M=972164: lift succeeded: 33 leaf nodes (0 safe).
E=12 N=61 M=972164: lift failed.

E=11 N=58 M=972164: OK, 14930 tree nodes, 47 leaf nodes (0 safe).
E=12 N=61 M=972164: lift failed.

E=12 N=61 M=972164: OK, 16684 tree nodes, 47 leaf nodes (0 safe).
E=13 N=64 M=972164: lift failed.

E=13 N=64 M=972164: OK, 22001 tree nodes, 47 leaf nodes (0 safe).
E=14 N=67 M=972164: lift succeeded: 47 leaf nodes (0 safe).
E=15 N=70 M=972164: lift failed.

E=14 N=67 M=972164: OK, 19902 tree nodes, 47 leaf nodes (0 safe).
E=15 N=70 M=972164: lift succeeded: 47 leaf nodes (0 safe).
E=16 N=73 M=972164: lift succeeded: 47 leaf nodes (10 safe).
E=17 N=76 M=972164: lift succeeded: 47 leaf nodes (26 safe).
E=18 N=79 M=972164: lift succeeded: 47 leaf nodes (37 safe).
E=19 N=82 M=972164: lift succeeded: 47 leaf nodes (44 safe).
E=20 N=85 M=972164: lift succeeded: 47 leaf nodes (47 safe).

E=10 N=56 M=1048577: OK, 190 tree nodes, 1 leaf nodes (1 safe).
```