Efficient and fast data compression codes for discrete sources with memory

EFFICIENT AND FAST DATA COMPRESSION CODES FOR DISCRETE SOURCES WITH MEMORY

TJALLING TJALKENS

EFFICIENT AND FAST DATA COMPRESSION CODES FOR DISCRETE SOURCES WITH MEMORY.

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de rector magnificus, prof.dr. F.N. Hooge, voor een commissie aangewezen door het college van dekanen in het openbaar te verdedigen op dinsdag 29 september 1987 te 16.00 uur

door

TJALLING JAN TJALKENS

geboren te Arnhem

Dit proefschrift is goedgekeurd door

de promotoren

prof.dr.ir. J.P.M. Schalkwijk

en

prof.dr. J.L. Massey

ISBN 90-71382-20-6

FEBO druk, Enschede

CONTENTS.

1. INTRODUCTION.

1.1. Problem statement and summary of results.

This thesis concerns the compression of data sequences generated by sources with memory. We will consider the (simplified) communication situation where a transmitter (source) selects messages (data sequences) from a message set and wants to send these to a receiver (destination). For this purpose it uses an error-free channel that is able to transfer symbols from a finite set (channel alphabet) at a fixed rate, or at a fixed cost. To transmit a message over the channel, it must first be converted into a string of channel symbols, using an invertible map (codebook). The fact that this map is invertible and that the channel is error-free allows the receiver to decode the message exactly.

The purpose of data compression or noiseless source coding is to minimize the number of channel digits needed to describe a message. Let us first define in more detail what a source, a message, and a message set is. All sources in Information Theory are modeled as stochastic processes. The simplest class of these models consists of the discrete memoryless sources (DMS). In later chapters we will also introduce the class of finite state Markov sources (FMS) and the class of stationary ergodic sources (SES). All these sources generate letters from a finite alphabet. The DMS generate sequences of letters, each letter drawn independently from the source alphabet according to some fixed probability vector. FMS and SES use probability vectors that are conditioned on the past, i.e. the previously generated letters. The definitions are such that the class of DMS is included in the class of FMS and both are included in the class of SES. In all cases we may define the message to be a string of source letters. The message set

is the set of strings over the source alphabet that we will consider as units we want to assign codewords to. For example, let $\mathcal{U}$ denote the source alphabet {0, 1}. Possible message sets would be: $\mathcal{M}_1 = \mathcal{U}$, $\mathcal{M}_2$ = {00, 01, 10, 11}, $\mathcal{M}_3$ = {0, 10, 11}, $\mathcal{M}_4 = \mathcal{U}^\infty$, i.e. the set of all infinite length strings $\underline{u}^\infty \triangleq u_1\, u_2\, u_3 \ldots$ over the alphabet $\mathcal{U}$.

With $\mathcal{M}_1$ we want to assign separate codewords to each single letter of the source string. $\mathcal{M}_2$ is an example of "blocking". Usually coding for a block of n ($n > 1$) symbols is more efficient than coding for single letters. The strings (segments) in the message set need not be of the same length, e.g. $\mathcal{M}_3$. In this way some advantage can be taken from the typical behaviour of a source. The last example designates a stream code. Actually, $\mathcal{M}_4$ states that codewords are assigned to each different infinite length source string, but in practice we will have to build up the codeword as the source letters "stream by" to avoid an infinite coding delay.

The last step in the code description is the assignment of codewords to the segments. Let $\mathcal{C}$ denote the codeword set then its cardinality $|\mathcal{C}|$ must be equal to $|\mathcal{M}|$.

Not every message set $\mathcal{M}$ and codeword set $\mathcal{C}$ is acceptable. We require that all possible infinite length source strings are separable in segments from $\mathcal{M}$. Usually, although not necessarily, we require that every string is separable in exactly one way. Also, since the codewords are concatenated, the decoder must be able to recognize the codewords. Two properties of sets of strings are of interest here. First we say that a string set is complete if every possible infinite length string has a prefix in the set. Also, a string set is proper if every possible infinite length string has at most one prefix in the set. Now we can say that $\mathcal{M}$ must be complete and usually is also proper. The codeword set must allow the source sequence to be reconstructed with finite delay from the encoded sequence. From the Kraft

Inequality ([1, Thm. 3.2.1]) and McMillans theorem ([1, Thm 3.2.2]) we can, without loss of generality, require $\mathcal{C}$ to be proper. So we only consider what are called prefix-free codes. It is not hard to see that if a proper codeword set $\mathcal{C}$ is not complete then there exists a proper set $\mathcal{C}'$ that is complete, has at least as many codewords, and, contains, for every codeword $\underline{x}$ in $\mathcal{C}$, a corresponding codeword $\underline{x}'$ with length $L(\underline{x}') \leq L(\underline{x})$. Since we are interested in a short description length $\mathcal{C}'$ is an improvement over $\mathcal{C}$. Thus one could consider only codeword sets that are proper and complete. However, we shall occasionally find it convenient to use codeword sets that are proper and not complete. So a code is described by the message set $\mathcal{M}$, the codeword set $\mathcal{C}$ and an invertible mapping from $\mathcal{M}$ to $\mathcal{C}$. In some cases a code uses many message sets or codeword sets and a rule for selecting the next set depending on the past.

According to the form of the message set and/or codeword set we can differentiate between several forms of source coding. First we discriminate between <u>definite coding schemes</u>, where $\mathcal{M}$ is a finite set, and <u>indefinite coding schemes</u> (or "<u>stream coding schemes</u>"), where $\mathcal{M}$ is an infinite set. Definite schemes can be further subdivided into four classes depending on whether or not the strings in $\mathcal{M}$ respectively $\mathcal{C}$ are of the same length. Thus we have the <u>fixed-to-variable length</u> (FV) schemes, here $\mathcal{M} = \mathcal{A}^n$, $(n \geq 1)$, and $\mathcal{C}$ contains variable length words. Next come the <u>variable-to-fixed length</u> (VF) schemes, where the source messages are of varying length and $\mathcal{C}$ contains fixed length words. In this case we might as well assume that the channel can transmit symbols from a "super alphabet" of cardinality $|\mathcal{C}|$. The third class contains the <u>variable-to-variable length</u> (VV) schemes. Here $\mathcal{M}$ as well as $\mathcal{C}$ contain varying length strings. Because the <u>fixed-to-fixed length</u> (FF) schemes are inherently inefficient unless one waives the requirement that the source sequence be uniquely reconstructable from the

encoded sequence (i.e. unless one allows "lossy coding"), FF schemes will not be considered further in this thesis.

The goal is to minimize the average number of channel letters per source letter (code rate). Thus for the FV schemes we must minimize the average codeword length and for the VF schemes the average message length must be maximized. In the case of a VV scheme the situation is more complex since the selection of the message set influences the best choice for $\mathcal{C}$ and as yet no algorithm is known for finding the best combination except by exhaustive search.

For the FV coding schemes the optimal code is generated by the Huffman procedure [2]. For the class SES this algorithm gives the best possible codeword set for a given number of codewords, i.e. it minimizes the average codeword length. A less known result is an VF coding algorithm by Tunstall [3], [4], that generates proper and complete message sets for a given number of messages. This algorithm generates the optimal set if the source is in the class of DMS. For a larger class no algorithm for optimal VF codes exists.

Stream coding schemes can be generated with the Elias algorithm [5]. This algorithm, although it is nearly optimal in the sense that the expected code rate can be made arbitrarily close to the source entropy, suffers from the fact that its per symbol complexity grows unbounded as the source entropy is approached.

In general there are two ways to implement a definite coding scheme. The first method is by table lookup. Depending on the form of the scheme we need tables to define the message set, codeword set, and the mapping between these sets. In the other method the messages are recognized by computational means. Also the corresponding codeword is computed by some algorithm.

In this thesis we shall describe two source coding algorithms of the last type. In chapter 2 we consider data sequences that are generated by (finite state) Markov sources, (see [1, section 3.6]). It is assumed that the Markov sources have the property that their state is uniquely determined by the previous output and the previous state and that their state set is irreducible.

The coding strategy we employ is of the variable-to-fixed length form. Each segment (message) is represented by its lexicographical index within the set of possible segments. Which segments are possible depend on the previous outputs of the Markov source. By knowing the lexicographical indices, it is possible to reconstruct the output sequence. Throughout this chapter we assume that the starting state of the Markov source is known to both the encoder and the decoder.

Variable-to-fixed length schemes for discrete memoryless sources instead of Markov sources were also investigated by Verhoeff [6], Jelinek and Schneider [4], Schalkwijk, Antonio and Petry [7], and Schalkwijk [8]. Verhoeff [6] reinvented Tunstall's algorithm. If the number of segments in the set grows to infinity, the code rate approaches the source entropy. This was demonstrated both by Verhoeff [6] and by Jelinek and Schneider [4]. A disadvantage of Tunstall's algorithm is that it requires a codebook that contains all segments with their lexicographical indices. This codebook has to be stored in memory at the encoder and at the decoder. Schalkwijk, et al. [7] devised a variable-to-fixed length source coding algorithm, that uses a multidimensional array instead of the complete codebook to determine the segment's index or vice versa. Schalkwijk [8] subsequently reported on a technique which uses a linear array and not a multidimensional one.

The algorithm we describe and analyse in this chapter uses as many seg-

ment sets as states of the Markov source. The lexicographical indices are computed in a way closely related to Schalkwijk's [8] recursive technique, however, now with a linear array for every state of the source. Upper and lower bounds are derived for the code rate of our algorithm. Using these bounds we show that code rates arbitrarily close to the source entropy are obtainable.

Chapter 3 describes a practical implementation of the Elias' algorithm. The resulting stream codes are known as arithmetic codes, introduced by Rissanen [9] and Pasco [10]. In these codes the code stream is treated as a number and encoding the next source symbol is done by adding a certain amount to this number. The arithmetic code is a noiseless compression technique applicable in all situations where the source statistics are known. Its main advantage, apart from its efficiency, lies in the flexibility of the algorithm. For sources with memory, the (conditional) probabilities change from symbol to symbol. Also, for not completely known sources we might estimate the probabilities from the generated source outputs. The encoder and decoder need these probabilities only when encoding respectively decoding that symbol, while any definite scheme needs them all in advance.

The arithmetic codes described here are based on both Elias' and Rissanen's work. We describe several code designs and analyse the resulting redundancy and complexity. It appears that the redundancy and the complexity can be upperbounded by functions of the design parameters only, i.e. they are independent of the source statistics. Since the redundancy is low at a moderate code complexity these codes are an attractive practical alternative to the definite schemes, e.g. the Huffman codes.

Finally, in chapter 4, we discuss the relations between the two coding schemes of the previous chapters. Here we show that the arithmetic codes

can be seen as approximations to the VF scheme of chapter 2.

1.2. General notations.

In this section we describe some notational conventions used throughout this thesis.

Script uppercase symbols denote sets.

For a finite set $\mathcal{Z}$ we use the following notations:

$|\mathcal{Z}|$ is the cardinality of $\mathcal{Z}$,

$\mathcal{Z}^n$, $n = 0, 1, 2, \ldots$ is the set of all strings of length n over $\mathcal{Z}$, i.e. $z_1 z_2 \ldots z_n \in \mathcal{Z}^n$ if $z_i \in \mathcal{Z}$, for $i = 1, 2, \ldots, n$, ($\mathcal{Z}^0$ contains the empty string),

$\mathcal{Z}^*$ denotes the set of all finite length strings over $\mathcal{Z}$, or $\mathcal{Z}^* = \mathcal{Z}^0 \cup \mathcal{Z} \cup \mathcal{Z}^2 \cup \ldots$,

$\mathcal{Z}^\infty$ contains all infinite length strings $z_1 z_2 \ldots$ over $\mathcal{Z}$.

We list some of the sets that are used here:

$\mathcal{U}$ is the source alphabet $\{0, 1, \ldots, c-1\}$, where c is the size of the alphabet,

$\mathcal{X}$ is the code alphabet $\{0, 1, \ldots, d-1\}$,

$\mathcal{M}$ is the message set; usually $\mathcal{M} \subset \mathcal{U}^*$ or $\mathcal{M} \subset \mathcal{U}^\infty$,

$\mathcal{C}$ is the codeword set; $\mathcal{C} \subset \mathcal{X}^*$ or $\mathcal{C} \subset \mathcal{X}^\infty$.

Strings are denoted by underscored italic symbols, e.g. $\underline{u}$. We write:

$\underline{u}^n \in \mathcal{U}^n$:	$\underline{u}^n$ is the string $u_1 u_2 \ldots u_n$ with $u_i \in \mathcal{U}$, $1 \leq i \leq n$.
$\underline{u}_i^{j-1} \in \mathcal{U}^{j-i}$:	$\underline{u}_i^{j-1} = u_i u_{i+1} \ldots u_{j-1}$ over $\mathcal{U}$.
$\underline{u}^0$:	is the empty string.
$\underline{u}^\infty \in \mathcal{U}^\infty$:	is an infinite length string $u_1 u_2 \ldots$ over $\mathcal{U}$.
$\underline{u} \in \mathcal{U}^*$:	$\underline{u}$ is a string of undefined but finite length over $\mathcal{U}$.

$(\underline{u})^m$: is a sequence of m strings $(\underline{u})_1\ (\underline{u})_2 \ldots (\underline{u})_m$ over $\mathcal{U}$.

Uppercase italics are used to name random variables, so

U_t: is the t^{th} random variable of the source.

$\underline{U}^n$: describes the first n outputs of the source.

Probabilities are denoted by the uppercase italics P and Q. P is a (conditional) probability and Q is a (conditional) cumulative probability, e.g. $P(\underline{u}|s)$ is the probability of the string $\underline{u}$ conditioned on the state s; $Q(u_i|\underline{u}^{i-1})$ is the cumulative symbol probability given the previous $i-1$ symbols, i.e. the probability that the next source letter is lexicographically less than u_i given that the previous output sequence is $\underline{u}^{i-1}$. $\tilde{P}$ and $\tilde{Q}$ denote approximated probabilities.

All specific notations are introduced in the text where needed.

2. A VARIABLE-TO-FIXED LENGTH CODE.

2.1. Introduction.

This chapter describes a variable-to-fixed length code for Markov sources. The technique used is known as enumerative coding. The first known enumerative source codes are the Lynch-Davisson code [11], [12] and Schalkwijk's Pascal triangle algorithm [13]. Later Cover [14] described some generalized applications of this technique.

In the next section we will define the class of (finite state) Markov sources for which the coding scheme applies. Then we introduce the algorithm and analyse its performance. Finally, we show the equivalence between these codes and Tunstall's algorithm for the class of DMS. Thus we also have a simple and optimal VF code for the discrete memoryless sources.

2.2. The Markov source.

A Markov source can be characterized by a source letter alphabet $\mathcal{U}$, a finite state set $\mathcal{S}$, a letter probability matrix $P(.|.)$ and a next state function $T(.,.)$. With each unit of time the source emits a letter and assumes a new state. The source sequence is denoted by $u_1, u_2, \ldots$ and the state sequence by $s_1, s_2, \ldots$. The conditional probability that an output u occurs given that the source is in state s is independent of all previous outputs and all previous states as follows:

$$\Pr(U_n=u|S_n=s,U_{n-1},S_{n-1},\ldots) \triangleq P(u|s),$$

where $n = 1, 2, \ldots$, $u \in \mathcal{U}$ and $s \in \mathcal{S}$. In this thesis we always assume that

the state of the source is uniquely determined by the previous state and the previous output letter, i.e.:

$$s_n \triangleq T(u_{n-1}, s_{n-1}), \text{ for } P(u_{n-1}|s_{n-1}) > 0, \qquad (2\text{-}1)$$

where $n = 2, 3, \ldots$. This property makes it possible to reconstruct the state sequence from the sequence of source outputs and the first state s_1.

Note that a more general type of (finite state) "Markov source" could be defined, in which the state need not be uniquely determined by the previous state and previous source letter. We will not study such sources here.

A set of states is irreducible if, from every state in the set every other state in the set can be reached in one or more transitions but no state outside this set can be reached. The period of an irreducible set of states is the largest integer p such that all possible recurrence times for states in the set are multiples of p. If $p \geq 2$ the set is called periodic; if $p = 1$ the set is aperiodic or ergodic. A state s is transient if from it another state can be reached in one or more transitions, but from there it is impossible ever to return to s. The sources we investigate in this thesis are those whose state set $\mathcal{S}$ contains only one irreducible subset of states, but we will not require that this subset be ergodic. These sources are started infinitely far in the past, so any transient state will never occur anymore and we may consider the irreducible subset of states as the complete state set $\mathcal{S}$.

Now define the state transition probability matrix $\mathbf{W}$ for $t, s \in \mathcal{S}$ such that

$$\mathbf{W}(t|s) \triangleq \sum_{u:T(u,s)=t} P(u|s).$$

Let $q(s)$, $s \in \mathcal{S}$ be the probability vector that is the solution of

$$q(t) = \sum_{s \in \mathcal{S}} q(s)W(t|s), \qquad t \in \mathcal{S}, \tag{2-2a}$$

$$1 = \sum_{s \in \mathcal{S}} q(s), \tag{2-2b}$$

where the uniqueness of $q(s)$ follows from the fact that the state set has a unique irreducible subset. The vector $q(s)$ is referred to as the stationary probability vector. We finally assume that the source starts in state s with probability

$$\Pr(S_1=s) \triangleq q(s).$$

This has the effect that the source, after it has started, behaves as if it was started infinitely far in the past (with each phase equiprobable if the state set is periodic). Such a source is stationary and ergodic and the notion of entropy for such a source makes sense. In fact for a Markov source with an irreducible state set the entropy

$$H_\infty(P,T) = \sum_{s \in \mathcal{S}} q(s)H(P(U|s)) \qquad \text{bit/source letter,}$$

where $H(Q(U|s)) \triangleq \sum_{u \in \mathcal{U}} - P(u|s)\log_2 P(u|s)$

and $0\log_2 0 \triangleq 0$. Before we give an example of a Markov source, we remark that in our short treatment of these sources we closely follow Gallager's approach [1].

Example (see Figure 2-1): Given $\mathcal{U} = \{0, 1, 2\}$ and $\mathcal{S} = \{a, b, c\}$ we have

$P(0|a) = 0.7$, $P(1|a) = 0.2$, $P(2|a) = 0.1$,

$P(0|b) = 0.3$, $P(1|b) = 0.3$, $P(2|b) = 0.4$,

$P(0|c) = 1.0$, $P(1|c) = 0$, $P(2|c) = 0$,

$T(0,a) = 2$, $T(1,a)=1$, $T(2,a)=3$,

$T(0,b) = 3$, $T(1,b)=3$, $T(2,b)=1$,

$T(0,c) = 1$, $T(1,c)=-$, $T(2,c)=-$.

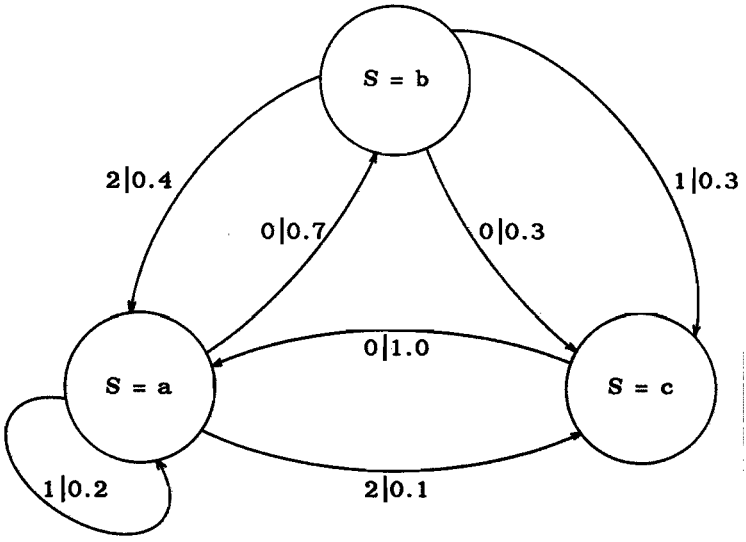

Figure 2-1. A three state Markov source

It is easily seen that the state set of our example source is aperiodic. The state transition probability matrix and the stationary probability vector turn out to be

$W(a|a) = 0.2$, $W(b|a) = 0.7$, $W(c|a) = 0.1$,

$W(a|b) = 0.4$, $W(b|b) = 0$, $W(c|b) = 0.6$,

$W(a|c) = 1.0$, $W(b|c) = 0$, $W(c|c) = 0$,

$$q(a) = 0.45045, \qquad q(b) = 0.31532, \qquad q(c) = 0.23423,$$

whereas the entropy of the source

$$H_\infty(P,T) = 1.01642 \text{ bit/source letter.}$$

2.3. The algorithm.

In a variable-to-fixed (VF) length source coding situation, Figure 2-2, the encoder chops the source output sequence into pieces (segments) of about the same probability. To inform the decoder, the encoder sends it the lexicographical index of the segments. The decoder is then able to

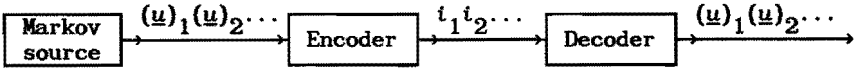

Figure 2-2. Variable-to-fixed length source coding situation.

reconstruct the segments. Two approaches exist to determine indices from segments and vice-versa. In the first approach the encoder uses a table that gives the lexicographical index with the segments as argument, and the decoder uses a table that tells him what the segment is with the index as argument. The encoder in the second approach computes the index from the segment and the decoder computes the segment using the index. Note that in the second approach no tables are used to chop up the output sequence. The encoder determines the end of a segment by some numerical method. The second approach is followed here.

The objective that all segments should have approximately the same probability can be replaced by the objective that the logarithm of the proba-

bility of all segments should be about the same. This logarithm can be computed by adding up the logarithms of the conditional probabilities of the letters in the segment. Since logarithms of probabilities are generally infinite precision real numbers, and so require high precision arithmetic in the additions, we replace them by integers. Hence checking whether or not the end of a segment is reached is a matter of adding up (small, as we will see later) integers. To introduce these integers, we will first extend the next state function and the letter probability matrix. Therefore, with some abuse of notation, define for $k = 1, 2, \ldots$, $\underline{u}^k \triangleq u_1, u_2, \ldots, u_k$ and then, recursively,

$$\begin{aligned} P(\underline{u}^k|s) &\triangleq P(u_k|T(\underline{u}^{k-1},s))P(\underline{u}^{k-1}|s), && P(\underline{u}^{k-1}|s) > 0; \\ &\quad 0, && P(\underline{u}^{k-1}|s) = 0; \\ T(\underline{u}^k,s) &\triangleq T(u_k,T(\underline{u}^{k-1},s)), && P(\underline{u}^k|s) > 0 \\ P(\underline{u}^0|s) &\triangleq 1, \quad T(\underline{u}^0,s) = s. \end{aligned} \tag{2-3}$$

We now assign a stepvalue function $V(u|s)$ for all $u \in \mathcal{U}$, $s \in \mathcal{S}$ with $P(u|s) > 0$, in the manner that this function takes values in the non-negative integers and the sum of this function over every closed circuit of states is positive. That a relation must exist between $V(u|s)$ and $P(u|s)$ is obvious. However, we will not explicitly describe this relation now.

When we are not initially interested in the length of $u_1, u_2, \ldots, u_k$, $(k \geq 1)$ we write $\underline{u}$ in stead of $\underline{u}^k$. Furthermore, if $\underline{u} = u_1, u_2, \ldots, u_k$, then the length of $\underline{u}$ is $L(\underline{u}) \triangleq k$.

Now assume that n is some fixed positive integer. Then for each state $s \in \mathcal{S}$ we define the segment set

$$\mathcal{M}_s(n) \triangleq \{\underline{u} \mid \quad P(\underline{u}|s) > 0$$
$$\text{and} \quad \sum_{k=1,L(\underline{u})} V(u_k|T(\underline{u}^{k-1},s)) \geq n$$
$$\text{and} \quad \sum_{k=1,L(\underline{u})-1} V(u_k|T(\underline{u}^{k-1},s)) < n\}. \qquad (2\text{-}4)$$

Note that such a segment set $\mathcal{M}_s$ is proper (no segment is the prefix of an other segment in the set) and complete (every possible infinite sequence has a prefix in the segment set).

Next we define an ordering of the source alphabet $\mathcal{U}$. This makes it possible to order the segments in a segment set, ($\underline{u} < \underline{v}$ if $u_k < v_k$ for the smallest k such that $u_k \neq v_k$).

The (lexicographical) index i of a segment $\underline{u}$ in $\mathcal{M}_s(n)$ is now given by

$$i_s(\underline{u},n) \triangleq |\{\underline{v} \in \mathcal{M}_s(n) \text{ such that } \underline{v} < \underline{u}\}|. \qquad (2\text{-}5)$$

This means that the index of a segment is equal to the number of segments in the set "smaller" than that segment.

Now we have defined for each state $s \in \mathcal{S}$ the segment set and the lexicographical index of a segment in the set. It will be clear that a one-to-one relation exists between the segments and the indices. Note that both the encoder and the decoder can keep track of the state in which a new segment starts since (2-1) holds for our Markov source and the initial state is known.

How does the encoder compute the index of a segment (and the end of a segment)? To demonstrate the algorithm, we first introduce for $s \in \mathcal{S}$ and for $m \geq 1$:

$$M_s(m) \triangleq |\mathcal{M}_s(m)|.$$

With the convention that $M_s(m) = 1$ if $m \leq 0$, these cardinalities can be found recursively as follows:

$$M_s(m) \triangleq \sum_{u:P(u|s)>0} M_{T(u,s)}(m-V(u|s)), \qquad 1 \leq m \leq n \qquad (2\text{-}6)$$

We remark here that $M_s(m)$ for each m must be computed in the right order since stepvalues equal to zero can exist. Such a "right order" can easily be found because no circuits whose stepvalues sum to zero exist. In what follows we will neglect this aspect of the algorithm.

Now assume that at time instant k the next segment starts. The source is now in state s_k, and both the encoder and the decoder are informed about this state s_k. The encoder now observes the next outputs of the source. In principle, this is a semi-infinite sequence $\underline{u}_k^{\infty} \triangleq u_k, u_{k+1}, \ldots$. The index i and the end of the next segment are now determined as follows:

<u>initialization</u>: $i_k := 0$,
$n_k := n$,
$j := k$,

<u>step</u>: $i_{j+1} := i_j + \sum_{v<u_j:P(v|s_j)>0} M_{T(v,s_j)}(n_j - V(v|s_j))$,
$s_{j+1} := T(u_j, s_j)$,
$n_{j+1} := n_j - V(u_j|s_j)$,
$j := j+1$.
if $n_j \leq 0$ stop; otherwise return to <u>step</u>. (2-7)

If ℓ is the first time instant for $n_\ell \leq 0$, then $u_{\ell-1}$ is the last letter of the segment and i_ℓ is the lexicographical index (i) of the segment. This index is now sent to the decoder. Using his knowledge of the state s_k, the decoder determines the next segment as follows:

$$\underline{\text{initialization}}: \quad i_k := i,$$

$$n_k := n,$$

$$j := k,$$

$$\underline{\text{step}}: \quad u_j := u \text{ iff } \sum_{v<u:P(v|s_j)>0} M_{T(v,s_j)}(n_j - V(v|s_j)) \le i_j$$

$$< \sum_{v\le u:P(v|s_j)>0} M_{T(v,s_j)}(n_j - V(v|s_j)),$$

$$i_{j+1} := i_j - \sum_{v<u_j:P(v|s_j)>0} M_{T(v,s_j)}(n_j - V(v|s_j)),$$

$$s_{j+1} := T(u_j, s_j),$$

$$n_{j+1} := n_j - V(u_j|s_j),$$

$$j := j+1.$$

if $n_j \le 0$ stop; otherwise return to step. (2-8)

Again, if ℓ is the first time instant for which $n_\ell \le 0$, then $u_{\ell-1}$ is the last letter of the segment.

It should be clear that the encoder, by carrying out (2-7), generates the index defined in (2-5) and that the decoder finds the encoded segment performing (2-8). Note that since the encoder and the decoder now know the starting state and the segment, they can both form the starting state of the following segment. We now demonstrate our algorithm using the Markov source of Figure 2-1.

Example: First, we choose the following stepvalue function $V(u|s)$:

$$V(0|a) = 1, \quad V(1|a) = 2, \quad V(2|a) = 3,$$

$$V(0|b) = 2, \quad V(1|b) = 2, \quad V(2|b) = 1,$$

$$V(0|c) = 0, \quad V(1|c) = -, \quad V(2|c) = -,$$

where "-" implies that $V(u|s)$ is not defined for these values of u and s. Note that there are no circuits whose stepvalues sum to zero (see also

Figure 2-3). The segment sets $M_s(2)$, s = a, b, c, are given in Figure 2-4.

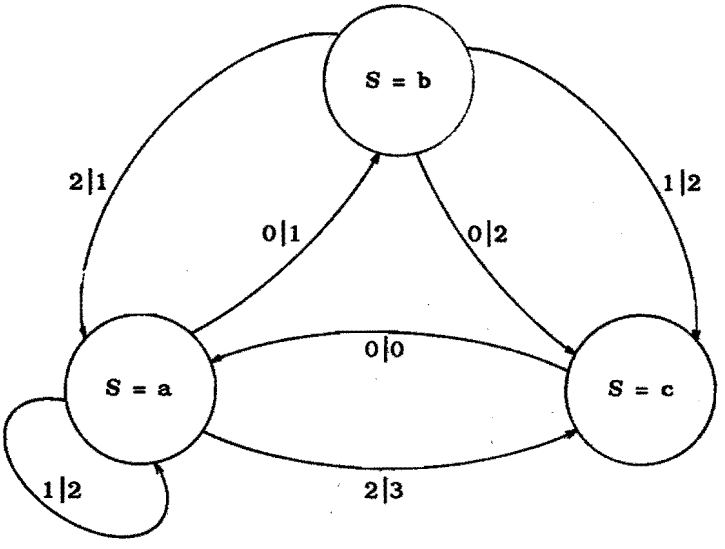

Figure 2-3. The Markov source with the steps.

As we see the number of segments in all sets is equal to 5 for $n = 2$. In general, these numbers may differ from each other.

Now we define the ordering $0 < 1 < 2$ over $\mathcal{A}$. In Figure 2-4 the segments for $n = 2$ are already plotted in this order. The cardinalities of the segment sets can be found from

$$
\begin{aligned}
M_a(m) &= M_a(m-2) + M_b(m-1) + M_c(m-3) \\
M_b(m) &= M_a(m-1) \qquad\qquad +2M_c(m-2) \\
M_c(m) &= M_a(m) \qquad\qquad\qquad\qquad , \; 1 \le m \le n
\end{aligned}
$$

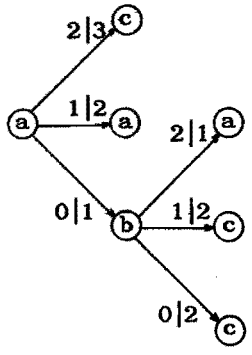

Figure 2-4[a]. Segment set $M_a(2)$.

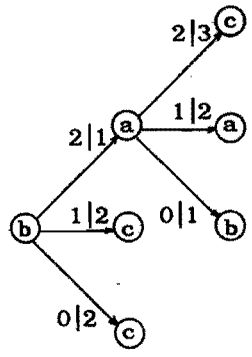

Figure 2-4[b]. Segment set $M_b(2)$.

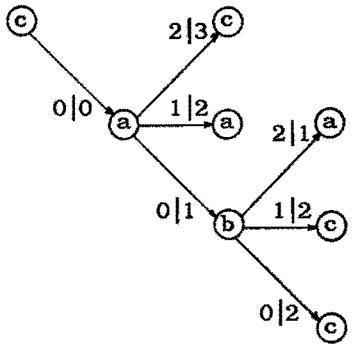

Figure 2-4[c]. Segment set $M_c(2)$.

We tabulate the first steps of this recursion as follows:

m	1	2	3	4	5	6	7	8	9	10
$M_a(m)$	3	5	9	19	33	65	123	229	441	827
$M_b(m)$	3	5	11	19	37	71	131	253	475	899
$M_c(m)$	3	5	9	19	33	65	123	229	441	827.

Consider next the segment sets for $n = 10$. Assume that the source is in

state s = b, and let the output of the source be 211200100... . What now is the next segment and its index? It follows from Figure 2-5 that the next segment is 2112001 and that

$$\begin{aligned} i_b(2112001) &= M_c(8) + M_c(8) + M_b(8) + \\ &\qquad M_b(6) + M_b(4) + M_a(3) + M_c(-1) \\ &= 229 + 229 + 253 + 71 + 19 + 9 + 1 \\ &= 811. \end{aligned}$$

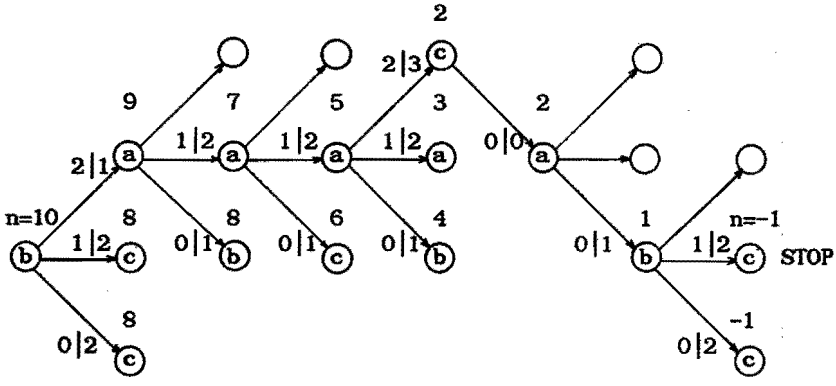

Figure 2-5. Illustration of the encoding process.

2.4. The performance.

A VF length coding scheme for a Markov source (a segment set coding scheme) consists of a number of (proper and complete) segment sets $\mathcal{M}_s$, one for each state of the source. The message set is $\mathcal{M}_s$ when the source is in state s at the time a new segment starts. Let M_{max} be defined as

$$M_{max} \triangleq \max \{|\mathcal{M}_s| \mid s \in \mathcal{S}\}$$

then $\lceil \log_2 M_{max} \rceil$ is the smallest length of a binary block code that can be used to encode $\mathcal{M}_s$ for all s. Thus it is natural to define the rate of this VF length coding scheme as:

$$R \triangleq \frac{\log_2 M_{max}}{EL} \qquad \text{bits/source letter} \qquad (2\text{-}9)$$

where EL is the expected average segment length. Here we have omitted the rounding up of $\log_2 M_{max}$ to an integer, which represents the mismatch of the binary coding alphabet to the size of the message set, but this has negligible effect when the message set is large.

To find EL we will introduce the concept "segment source". Note that the starting state of a segment and the segment uniquely determine the starting state of the next segment. Since the probability of a certain segment, given all previous segments and previous starting states, depends only on the starting state of the segment, the source with segments as output letters is again a Markov source, or for $m = 1, 2, \ldots$, $s \in \mathcal{S}$ and $\underline{u} \in \mathcal{M}_s$.

$$\Pr((\underline{U})_m = \underline{u} | S_m = s, (\underline{U})_{m-1}, S_{m-1}, \ldots) \triangleq P(\underline{u}|s);$$
$$s_{m+1} \triangleq T((\underline{u})_m, s_m), \qquad P((\underline{u})_m | s_m) > 0.$$

Here we denote the m^{th} segment by $(\underline{u})_m$ and the sequence of segments by $(\underline{u})^M$ etc. . The starting probabilities of the segment source are the same as the starting probabilities of the "basic source". They equal the stationary vector $q(s)$, $s \in \mathcal{S}$ defined in (2-2). The state set $\mathcal{S}$ of the segment source need not be irreducible (see Appendix I for an example) and hence the seg-

ment source need not have unique stationary probabilities. But, because the starting state distribution is known, we can determine these probabilities unambiguously. Assume that $\mathcal{G} \triangleq \{\mathcal{G}_r\}$ is the finite set of all irreducible subsets $\mathcal{G}_r$ of $\mathcal{S}$. Then for the expectation of the average segment length we can write

$$\begin{aligned} EL &= \lim_{M\to\infty} E_{(\underline{U})^M|s_1}\{\frac{1}{M}\sum_{m=1,M} L((\underline{U})_m)\} \\ &= \sum_{\mathcal{G}_r\in\mathcal{G}} Pr(\mathcal{G}_r) \\ &\qquad \cdot \sum_{s\in\mathcal{G}_r} \lim_{M\to\infty} \frac{1}{M}\sum_{m=1,M} Pr\{S_m=s\,|\mathcal{G}_r\}E_{\underline{U}|s}\{L(\underline{U})\} \\ &= \sum_{\mathcal{G}_r\in\mathcal{G}} Pr(\mathcal{G}_r) \sum_{s\in\mathcal{G}_r} q_r(s)E_{\underline{U}|s}\{L(\underline{U})\}. \end{aligned} \tag{2-10}$$

Here $Pr(\mathcal{G}_r)$ is the probability that the source will eventually enter subset $\mathcal{G}_r$, and $q_r(s)$ is the stationary probability vector of the segment source when started in subset $\mathcal{G}_r$. This vector $q_r(s)$ can be found, for each r, in a way similar to the method to find the stationary probability vector of the basic source. Instead of $\mathcal{S}$ we now use $\mathcal{G}_r$. The expected segment length for segments starting in s is given by

$$E_{\underline{U}|s}\{L(\underline{U})\} \triangleq \sum_{\underline{u}\in\mathcal{M}_s} P(\underline{u}|s)L(\underline{u}).$$

We will now give an example to illustrate the foregoing formulas. Again we use the source in Figure 2-1.

Example: Consider $\mathcal{M}_s(2)$, s = a, b, c (see Figure 2-4). The segment probabilities are

$P(00\|a) = 0.21,$	$P(0\|b) = 0.3,$	$P(000\|c) = 0.21,$
$P(01\|a) = 0.21,$	$P(1\|b) = 0.3,$	$P(001\|c) = 0.21,$
$P(02\|a) = 0.28,$	$P(20\|b) = 0.28,$	$P(002\|c) = 0.28,$
$P(1\|a) = 0.2,$	$P(21\|b) = 0.08,$	$P(01\|c) = 0.2,$
$P(2\|a) = 0.1,$	$P(22\|b) = 0.04,$	$P(02\|c) = 0.1.$

The state transition probabilities can be found in Figure 2-6 in which the segment source is depicted.

It will be clear that the segment source consists of only one irreducible subset $\mathcal{G}_1 = \{a, c\}$. State b is a transient state. The probability that the source will enter $\mathcal{G}_1$ equals 1. The stationary probability vector over

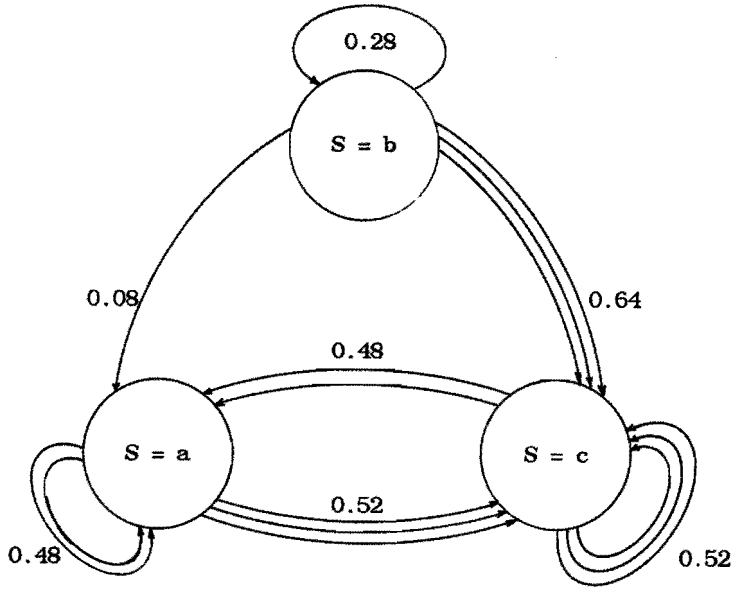

Figure 2-6. The segment source.

this irreducible subset is

$$q_1(a) = 0.48 \quad \text{and} \quad q_1(c) = 0.52.$$

Furthermore,

$$\begin{aligned} E_{\underline{U}|s}\{L(\underline{U})\} &= 1.7 \text{ letter/segment for } s = a, \\ &= 1.4 \text{ letter/segment for } s = b, \text{ and} \\ &= 2.7 \text{ letter/segment for } s = c. \end{aligned}$$

Therefore, $EL = 0.48 \times 1.7 + 0.52 \times 2.7 = 2.22$ letter/segment. Now since $M_{max} = 5$, the rate of this segment coding scheme is

$$R = \frac{\log_2 5}{2.22} = 1.04591 \text{ bits/source letter.}$$

Note that this rate is rather close to the source entropy (1.01642 bits/source letter). In the next section we will study the performance of the scheme by deriving upper and lower bounds for its rate as functions of the code parameters and the source statistics.

2.5. Bounds on the rate of the code.

First we define the number of visits to state t for a given segment $\underline{u}$ starting in state s as follows:

$$L_t(\underline{u},s) \triangleq \sum_{k=1,L(\underline{u})} \theta(T(\underline{u}^{k-1},s) = t),$$

with $\theta(\text{true}) \triangleq 1$ and $\theta(\text{false}) \triangleq 0$. Subsequently, in analogy with (2-10) we define

$$EL_t \triangleq \lim_{M\to\infty} E_{(\underline{U})^M|s_1}\{\frac{1}{M}\sum_{m=1,M} L_t((\underline{u})_m, T((\underline{u})^{m-1}, s_1))\}$$

$$= \sum_{\mathcal{S}_r\in\mathcal{S}} \Pr(\mathcal{S}_r) \sum_{s\in\mathcal{S}_r} q_r(s) E_{\underline{U}|s}\{L_t(\underline{U},s)\}, \qquad (2\text{-}11)$$

where the expected number of visits to state t for segments starting in s is given by

$$E_{\underline{U}|s}\{L_t(\underline{U},s)\} \triangleq \sum_{\underline{u}\in\mathcal{M}_s} P(\underline{u}|s) L_t(\underline{u},s). \qquad (2\text{-}12)$$

We will now state a crucial lemma which is proved in Appendix II.

Lemma 2-1: For any segment set coding scheme for a Markov source

$$EL_t = q(t)EL, \quad \text{for all } t \in \mathcal{S}.$$

Note that the segment sets need <u>not</u> be defined as in (2-4) for the lemma to hold, but can be defined in an arbitrary way subject to the proper and completeness conditions. Next we define the <u>pseudoprobability vector</u> (with $\alpha>0$)

$$\tilde{P}(u|s) \triangleq 2^{-\alpha V(u|s)}, \quad \text{for all } u, s \text{ with } P(u|s) > 0. \qquad (2\text{-}13)$$

The parameter α is to be determined later. We call $\tilde{P}$ a pseudoprobability vector since $\Sigma_u \tilde{P}(u|s)$ need not be one; negative pseudoprobabilities, however, do not occur. For $k = 1, 2, \ldots,$ we define $\tilde{P}(\underline{u}^k|s)$ analogous to $P(\underline{u}^k|s)$ defined in (2-3).

Now for each $s \in \mathcal{S}$ regard the segment set $\mathcal{M}_s$ as the leaves of a rooted <u>tree</u>. Then $\mathcal{M}_s^{int}$ is defined as the set of all internal nodes of this tree, including the root node. So:

$$\mathcal{M}_s^{int} = \{\underline{u} \mid \exists \underline{v} : L(\underline{v}) > 0 \text{ and } \underline{uv} \in \mathcal{M}_s\}$$

We now have

$$\begin{aligned}
&\sum_{\underline{u}\in\mathcal{M}_s} P(\underline{u}|s)\log_2 \frac{1}{\tilde{P}(\underline{u}|s)} = \\
&= \sum_{\underline{u}\in\mathcal{M}_s} P(\underline{u}|s)\left[\log_2 \frac{1}{P(\underline{u}|s)} + \log_2 \frac{P(\underline{u}|s)}{\tilde{P}(\underline{u}|s)}\right] \\
&\overset{(a)}{=} \sum_{\underline{u}\in\mathcal{M}_s^{int}} P(\underline{u}|s)\left[H(P(U|T(\underline{u},s))) + D(P(U|T(\underline{u},s))\|\tilde{P}(U|T(\underline{u},s)))\right] \\
&= \sum_{t\in\mathcal{S}} \sum_{\substack{\underline{u}\in\mathcal{M}_s^{int} \\ T(\underline{u},s)=t}} P(\underline{u}|s)\left[H(P(U|t)) + D(P(U|t)\|\tilde{P}(U|t))\right] \\
&\overset{(b)}{=} \sum_{t\in\mathcal{S}} \sum_{\underline{u}\in\mathcal{M}_s} P(\underline{u}|s)L_t(\underline{u},s)\left[H(P(U|t)) + D(P(U|t)\|\tilde{P}(U|t))\right] \\
&\overset{(c)}{=} \sum_{t\in\mathcal{S}} E_{\underline{U}|s}\{L_t(\underline{U},s)\}\left[H(P(U|t)) + D(P(U|t)\|\tilde{P}(U|t))\right]. \qquad (2\text{-}14)
\end{aligned}$$

where for $t \in \mathcal{S}$

$$D(P(U|t)\|\tilde{P}(U|t)) \triangleq \sum_{u\in\mathcal{U}} P(u|t)\log_2 \frac{P(u|t)}{\tilde{P}(u|t)}.$$

We call the quantity D a conditional <u>pseudodivergence</u> since $\tilde{P}$ is a pseudo-probability vector. However, P is a probability vector $(\Sigma_u P(u|s) = 1)$. In (a) and (b) we use Massey's [15] "leaf-node theorem" and in (c) we use (2-12).

For a segment $\underline{u} \in \mathcal{M}_s(n)$ as defined in (2-4)

$$\tilde{P}(\underline{u}|s) = 2^{-\alpha \sum_{k=1,L(\underline{u})} V(u_k|T(\underline{u}^{k-1},s))}$$

and therefore, with $V_{max} \triangleq \max\{V(u|s) | u \in \mathcal{U},\ s \in \mathcal{S},\ P(u|s) > 0\}$ we find that

$$\alpha n \leq \log_2 \frac{1}{\tilde{P}(\underline{u}|s)} \leq \alpha(n + V_{max} - 1). \tag{2-15}$$

If we combine (2-14) with (2-15) we obtain

$$\begin{aligned} \alpha n &\leq \sum_{t \in \mathcal{S}} E_{\underline{U}|s}\{L_t(\underline{U},s)\}[H(P(U|t)) + D(P(U|t)\|\tilde{P}(U|t))] \\ &\leq \alpha(n + V_{max} - 1). \end{aligned} \tag{2-16}$$

If we now set the pseudodivergence

$$D_\infty(P,\tilde{P},T) \triangleq \sum_{s \in \mathcal{S}} q(s) D(P(U|s)\|\tilde{P}(U|s))$$

it follows that

$$\begin{aligned} &EL \cdot (H_\infty(P,T) + D_\infty(P,\tilde{P},T)) \\ &\quad = EL \cdot \sum_{t \in \mathcal{S}} q(t)[H(P(U|t)) + D(P(U|t)\|\tilde{P}(U|t))] \\ &\quad \overset{(d)}{=} \sum_{t \in \mathcal{S}} EL_t[H(P(U|t)) + D(P(U|t)\|\tilde{P}(U|t))] \\ &\quad \overset{(e)}{=} \sum_{t \in \mathcal{S}} \sum_{\mathcal{G}_r \in \mathcal{G}} \Pr(\mathcal{G}_r) \cdot \sum_{s \in \mathcal{G}_r} q_r(s) \\ &\qquad \cdot E_{\underline{U}|s}\{L_t(\underline{U},s)\}[H(P(U|t)) + D(P(U|t)\|\tilde{P}(U|t))] \\ &\quad = \sum_{\mathcal{G}_r \in \mathcal{G}} \Pr(\mathcal{G}_r) \cdot \sum_{s \in \mathcal{G}_r} q_r(s) \\ &\qquad \cdot \sum_{t \in \mathcal{S}} E_{\underline{U}|s}\{L_t(\underline{U},s)\}[H(P(U|t)) + D(P(U|t)\|\tilde{P}(U|t))] \end{aligned} \tag{2-17}$$

Here (d) follows from Lemma 2-1 and (e) from (2-11). Lemma 2-2 is now obtained if we combine (2-16) with (2-17).

Lemma 2-2: For a segment set scheme as defined in (2-4) with $\tilde{P}$ as in (2-13):

$$\alpha n \leq \mathrm{EL}^*(H_\infty(P,T) + D_\infty(P,\tilde{P},T)) \leq \alpha(n + V_{max} - 1),$$

Now we want to relate $\log_2 M_{max}(n)$ to αn. Therefore, we introduce for $\lambda \geq 1$ the matrix $A(\lambda) \triangleq [A_{st}(\lambda)]$ such that

$$A_{st}(\lambda) \triangleq \sum_{\substack{u:P(u|s)>0 \\ T(u,s)=t}} \lambda^{-V(u|s)}, \qquad \lambda \geq 1, \quad s \in \mathcal{S}, \quad t \in \mathcal{S}. \quad (2\text{-}18)$$

and define I to be the identity matrix, which leads to the following lemma.

Lemma 2-3: (Gantmacher [16]): Let

$$\lambda_{max} \triangleq \max\{\lambda | \det[A(\lambda)-I]=0\}$$

and $\underline{e}$ be the characteristic vector of $A(\lambda_{max})$ for characteristic number 1 such that $\underline{e}$'s largest component is 1. Then all components of $\underline{e}$ are positive. This lemma is proved in Appendix III. Index the components of $\underline{e}$ with $s \in \mathcal{S}$, i.e., $\underline{e} \triangleq (e_s : s \in \mathcal{S})$. Then define

$$e_{min} \triangleq \min\{e_s | s \in \mathcal{S}\}$$

So $e_{min} > 0$. The next lemma is proved in Appendix IV.

Lemma 2-4: For a segment set code as defined in (2-4) for $s \in \mathcal{S}$ and $n \geq 1 - V_{max}$ (n is an integer),

$$(\lambda_{max})^n e_s \leq M_s(n) \leq \frac{1}{e_{min}} (\lambda_{max})^{n + V_{max} - 1} e_s.$$

where λ_{max} is defined in Lemma 2-3.

We will now work towards the main theorem of this section. First set $\eta \triangleq (-\log_2 e_{min})/\log_2 \lambda_{max}$. Then for $n = 1, 2, 3, \ldots$ it follows from Lemma 2-4 that

$$(n - \eta)\log_2 \lambda_{max} \leq \log_2 M_{max}(n) \leq \\ \leq (n + V_{max} - 1 + \eta)\log_2 \lambda_{max}. \qquad (2\text{-}19)$$

We now can choose the constant α in the definition of the pseudoprobability:

$$\alpha \triangleq \log_2 \lambda_{max}. \qquad (2\text{-}20)$$

Then

$$\frac{\log_2 M_{max}(n)}{EL} \leq \frac{(n + V_{max} - 1 + \eta)\log_2 \lambda_{max}}{EL} \\ = \frac{n + V_{max} - 1 + \eta}{n} \cdot \frac{\alpha n}{EL} \\ \leq \frac{n + V_{max} - 1 + \eta}{n} \cdot (H_\infty(P,T) + D_\infty(P,\tilde{P},T)), \qquad (2\text{-}21a)$$

and

$$\frac{\log_2 M_{max}(n)}{EL} \geq \frac{(n - \eta)\log_2 \lambda_{max}}{EL} \\ = \frac{n - \eta}{n + V_{max} - 1} \cdot \frac{\alpha(n + V_{max})}{EL} \\ \geq \frac{n - \eta}{n + V_{max} - 1} \cdot (H_\infty(P,T) + D_\infty(P,\tilde{P},T)). \qquad (2\text{-}21b)$$

From (2-21) and definition (2-9) we obtain Theorem 2-1.

Theorem 2-1: For a segment set code as defined in (2-4) for $n \geq 1$ (n is an integer) the rate $R(n)$ is bounded as follows:

$$\frac{n-\eta}{n+V_{max}-1} \cdot (H_\infty(P,T) + D_\infty(P,\tilde{P},T)) \leq R(n) \leq \frac{n+V_{max}-1+\eta}{n} \cdot (H_\infty(P,T) + D_\infty(P,\tilde{P},T)).$$

In the definition of the pseudoprobabilities α is set equal to $\log_2\lambda_{max}$ where λ_{max} is formed as in Lemma 2-3. Then η can be computed. Note that the stepvalue function V (together with the next-state function T) determines λ_{max}, α, $\tilde{P}$, $\underline{e}$ and η. All these parameters do <u>not</u> depend on n or P.

Example: For the source in Figure 2-1 with the stepvalue function as in Figure 2-3, we can determine $A(\lambda)$ and λ_{max} as follows:

$$A(\lambda) = \begin{bmatrix} \lambda^{-2} & \lambda^{-1} & \lambda^{-3} \\ \lambda^{-1} & 0 & 2\lambda^{-2} \\ 1 & 0 & 0 \end{bmatrix}, \qquad \lambda_{max} = 1.89329.$$

Now we determine $\underline{e}$ and η:

$$e_a = 0.92070, \quad e_b = 1.00000, \quad e_c = 0.92070, \quad \eta = 0.12944.$$

With $\alpha \triangleq \log_2\lambda_{max} = 0.92090$ we can find the pseudoprobabilities

$$\begin{aligned} &\tilde{P}(0|a) = 0.52818, \quad \tilde{P}(1|a) = 0.27898, \quad \tilde{P}(2|a) = 0.14735, \\ &\tilde{P}(0|b) = 0.27898, \quad \tilde{P}(1|b) = 0.27898, \quad \tilde{P}(2|b) = 0.52818, \\ &\tilde{P}(0|c) = 1.00000, \quad \tilde{P}(1|c) = -, \quad \tilde{P}(2|c) = -. \end{aligned}$$

For the pseudodivergence we find that

$$D_\infty(P,\tilde{P},T) = 0.02892 \text{ bit/source letter.}$$

Now we evaluate the bound in Theorem 2-1 for n = 10, 100, 1000 and 10000 and compare these values with the exact value of R:

$n = 10$	$0.85984 \leq R \leq 1.26794,$	$R = 1.07397,$
$n = 100$	$1.02352 \leq R \leq 1.06760,$	$R = 1.04839,$
$n = 1000$	$1.04312 \leq R \leq 1.04757,$	$R = 1.04565,$
$n = 10000$	$1.04512 \leq R \leq 1.04556,$	$R = 1.04537.$

From our example it is clear that for $n \to \infty$ the rate R approaches $H_\infty(P,T) + D_\infty(P,\tilde{P},T) = 1.04534$. This asymptotical behaviour of our segment schemes is the subject of the next section. We will also investigate the properties of the scheme when we choose the steps more proportional to the logarithm of the corresponding probabilities.

2.6. Asymptotic behaviour.

The first result is an immediate consequence of Theorem 2-1.

Corollary: For segment set coding schemes as defined in (2-4) we have

$$\lim_{n\to\infty} R(n) = H_\infty(P,T) + D_\infty(P,\tilde{P},T),$$

where the pseudoprobabilities are determined with $\alpha = \log_2\lambda_{max}$ and λ_{max} is defined in Lemma 2-3.

From the Corollary we see that $D_\infty(P,\tilde{P},T)$ plays an important role. What

are the values that this pseudodivergence can take on? A partial answer to this question is given in Lemma 2-5. The proof of Lemma 2-5 can be found in Appendix V.

Lemma 2-5: For segment set schemes as defined in (2-4), with pseudoprobabilities that are determined with $\alpha = \log_2\lambda_{max}$ and λ_{max} defined as in Lemma 2-3,

$$D_\infty(P,\tilde{P},T) \geq 0.$$

A code designer would like to choose the stepvalue function such that the pseudodivergence is as small as possible. What is the lowest possible value of this pseudodivergence for which a stepvalue function still exists? Theorem 2-2 states that pseudodivergences arbitrarily close to zero are achievable.

Theorem 2-2: Let $\gamma > 0$. Choose $V_\gamma(u|s) \triangleq \lceil -\gamma\log_2 P(u|s) \rceil$ for $u \in \mathcal{U}$, $s \in \mathcal{S}$ such that $P(u|s) > 0$. For segment set schemes as defined in (2-4) with pseudoprobabilities $\tilde{P}_\gamma$ that are determined with α_γ set equal to $\log_2\lambda_{max,\gamma}$, and $\lambda_{max,\gamma}$ as in Lemma 2-3,

$$\lim_{\gamma\to\infty} D_\infty(P,\tilde{P}_\gamma,T) = 0.$$

Here $\lceil x \rceil$ stands for the smallest integer not less than x. Note that for obvious reasons we added a subscript γ to quantities depending on γ. For the proof of Theorem 2-2 we refer to Appendix VI.

At the end of this section we conclude that with segment set schemes as defined in (2-4) we can achieve rates arbitrarily close to the source entropy $H_\infty(P,T)$. The encoding and decoding algorithms for these schemes are easy to implement. However, it could be possible that with segment set

schemes that are not defined as in (2-4) even lower rates than $H_\infty(P,T)$ are achievable. In the next section we show that this is not the case. With the following example we demonstrate the results in the present section.

Example: For the source in Figure 2-1 and for a few values of γ, $V_\gamma(u|s)$ and $D_\infty(P,\tilde{P}_\gamma,T)$ are listed:

	$-\log_2 P(u\|s)$	$V_{\gamma=1}(u\|s)$	$V_{\gamma=4}(u\|s)$	$V_{\gamma=16}(u\|s)$
u=0|s=a	0.51457	1	3	9
u=1|s=a	2.32193	3	10	37
u=2|s=a	3.32193	4	14	53
u=0|s=b	1.73697	2	7	28
u=1|s=b	1.73697	2	7	28
u=2|s=b	1.32193	2	6	21
u=0|s=c	0	0	0	0
$D_\infty(P,\tilde{P}_\gamma,T)$	bit|lett.	0.00855	0.00216	0.00017

2.7. The converse.

In this section we prove the following.

Theorem 2-3: For an arbitrary segment set coding scheme for a Markov source with entropy $H_\infty(P,T)$,

$$R \triangleq \frac{\log_2 M_{max}}{EL} \geq H_\infty(P,T).$$

Proof: Along the lines of (2-14) we obtain for the segment set $\mathcal{M}_s$, by setting $\tilde{P}(u|s) \triangleq P(u|s)$ instead of (2-13), that

$$\log_2 M_{max} \geq \sum_{\underline{u}\in\mathcal{M}_s} P(\underline{u}|s)\log_2 \frac{1}{P(\underline{u}|s)}$$

$$= \sum_{t\in\mathcal{T}} \mathbf{E}_{\underline{U}|s}\{L_t(\underline{U},s)\}\cdot H(P(U|t)). \qquad (2\text{-}22)$$

Now as in (2-17),

$$\mathbf{E}L\cdot H_\infty(P,T) = \mathbf{E}L\cdot \sum_{t\in\mathcal{T}} q(t)H(P(U|t))$$

$$= \sum_{t\in\mathcal{T}} \mathbf{E}L_t\cdot H(P(U|t))$$

$$= \sum_{\mathcal{G}_r\in\mathcal{G}} \Pr(\mathcal{G}_r) \sum_{s\in\mathcal{G}_r} q_r(s)$$

$$\cdot \sum_{t\in\mathcal{T}} \mathbf{E}_{\underline{U}|s}\{L_t(\underline{U},s)\}\cdot H(P(U|t))$$

$$\leq \log_2 M_{max}$$

where the last step follows from (2-22).

If we combine Theorems 2-2 and 2-3 we find that for Markov sources segment set schemes as defined in (2-4) can be found with rates arbitrarily close to the source entropy and that no (arbitrarily defined) segment set schemes exist with a rate less than the source entropy. Therefore, we conclude that the segment set schemes described and analysed here are not only easy to implement but also asymptotically optimal with respect to their compression capabilities.

Note that Theorem 2-3 sometimes gives a better lower bound for R than the lower bound in Theorem 2-1. Also note that Lemma 2-5 follows indirectly from the converse in this section.

2.8. Complexity aspects.

The $M_s(m)$ array, $s \in \mathcal{S}$, $1 \leq m \leq n$, must be present in memory both at the encoder and at the decoder. From Lemma 2-4 we observe that

$$\begin{aligned} \log M_s(n) &\leq -\log e_{min} + (V_{max} - 1 + n)\log\lambda_{max} \\ &\cong (V_{max} + n)\log\lambda_{max} \end{aligned} \qquad (2\text{-}23)$$

So we need about $n\log\lambda_{max}$ bits to store an array value. So for this array we need approximately

$$C^{stor} \simeq |\mathcal{S}|.n^2\log_2\lambda_{max} \quad \text{bit locations.}$$

We call C^{stor} the storage complexity. The computational complexity is equal to the maximum number of operations that have to be performed per source symbol, hence

$$C^{comp} \simeq |\mathcal{U}| - 1 \qquad \text{operations/symbol.} \qquad (2\text{-}24)$$

This result holds for the encoder and only shows the maximum number of additions and table indexings needed to add the next increment to the index, see (2-7). The state update etc. is performed once per symbol and is neglected here. The computational complexity of the decoder is also proportional to $|\mathcal{U}| - 1$ but the basic operation here is more complex. (We also need comparisons). See (2-8).

The storage complexity can be decreased to

$$C^{stor} \simeq |\mathcal{S}|n(r + \log_2(n\log_2\lambda_{max})) \quad \text{bit locations} \qquad (2\text{-}25)$$

if we compute the $M'_s(m)$ as follows:

$$M'_s(m) \triangleq \begin{cases} \left\lceil \sum_{u:P(u|s)>0} M'_{T(u,s)}(m - V(u|s)) \right\rceil_r, & 1 \le m \le n \\ 1, & m \le 0. \end{cases} \qquad (2\text{-}26)$$

where $\lceil A \cdot 2^B \rceil_r$ with B an integer, $1/2 \le A < 1$, equals $\lceil A \cdot 2^r \rceil \cdot 2^{B-r}$. Thus the $M'_s(m)$'s are the r bit precision analogues of the integers $M_s(m)$ defined in (2-6). When represented as floating point numbers the mantissa of M'_s requires only r bits, and the characteristic (or exponent) requires about $\log_2(n \log_2 \lambda_{max})$ bits. This follows from (2-23). In (2-26) the rounding up is necessary to guarantee decodability of the scheme. What happens is that we overestimate the cardinalities of some of the subsets $M_s(m)$. This also increases M_{max}, but not by much, so (2-23) will still hold approximately. It is also possible to decrease the computational complexity to

$$C^{comp} \simeq 1 \qquad \text{operation/symbol},$$

if we store, for every state $s \in \mathcal{S}$, every m such that $1 \le m \le n$ and every possible source symbol $u \in \mathcal{U}$

$$f_s(m,u) \triangleq \sum_{v<u:P(v|s)>0} M_{T(v,s)}(m - V(v|s)). \qquad (2\text{-}27)$$

Now the additions for every symbol of the source are carried out in advance, but since the results $f_s(m,u)$ have to be stored, the storage complexity is increased by a factor $|\mathcal{U}|$. Thus a trade-off exists here between computational complexity and storage complexity.

Decreasing the storage complexity to (2-25) increases the rate of the scheme, since we need a larger codeword set to accomodate the estimated

segment set cardinalities $M'_s(n)$. However, if r is not too small, this effect can be neglected. Decreasing the computational complexity to 1 operation/symbol has no influence on the rate. It is, of course, possible to combine the methods in (2-26) and (2-27).

2.9. Discrete memoryless sources.

Consider a discrete memoryless source with source letter alphabet $\mathcal{U}$ and probability vector $P(u)$, $u \in \mathcal{U}$. The entropy of the source is

$$H(P(U)) = \sum_{u \in \mathcal{U}} - P(u) \log_2 P(u) \qquad \text{bit/source letter.}$$

Choose for all $u \in \mathcal{U}$ stepvalues $V(u) \in \{1, 2, 3, \ldots\}$, and assume that $P(u) > 0$ for all $u \in \mathcal{U}$. Then let n be some positive integer. Now define the segment set scheme for n as follows:

$$\mathcal{M}(n) \triangleq \{\underline{u} \mid \sum_{k=1, L(\underline{u})} V(u_k) \geq n \text{ and } \sum_{k=1, L(\underline{u})-1} V(u_k) < n\}. \qquad (2\text{-}28)$$

Note that encoding and decoding is as in section 2.3 if we assume that only one state exists. Therefore, only one array $M(m)$ exists which can be filled recursively. Now let α be the solution of

$$\sum_{u \in \mathcal{U}} 2^{-\alpha V(u)} = 1.$$

If we now set

$$\tilde{P}(u) \triangleq 2^{-\alpha V(u)} \tag{2-29}$$

then we obtain for the rate

$$\frac{n}{n + V_{max} - 1} \cdot (H(P(U)) + D(P(U)\|\tilde{P}(U))) \leq R(n) \leq \frac{n + V_{max} - 1}{n} \cdot (H(P(U)) + D(P(U)\|\tilde{P}(U))),$$

where

$$D(P(U)\|\tilde{P}(U)) \triangleq \sum_{u \in \mathcal{U}} P(u) \log_2 \frac{P(u)}{\tilde{P}(u)} .$$

Note that $\tilde{P}(u)$, $u \in \mathcal{U}$ is not a pseudoprobability vector but a vector whose components sum to one, and, therefore, $D(P(U)\|\tilde{P}(U))$ is a ordinary divergence.

Just as for Markov sources it can be proved that rates arbitrarily close to the entropy of the source are achievable with properly chosen stepvalues. Again no codes exist with rates lower than the source entropy. The latter was already proved in [4].

The schemes described in this section equal the optimal (Tunstall) codes, [3], [4], if

i) the stepvalues $V(u)$ can be chosen such that $\tilde{P}(u)$, as given in (2-29), equals $P(u)$ for all $u \in \mathcal{U}$, and

ii) the size of the Tunstall code equals $|\mathcal{M}(n)|$.

In addition, for probability sets $P(\cdot)$ "close" to the resulting $\tilde{P}(\cdot)$, $\mathcal{M}(n)$ is still optimal. This in turn implies that for each source in the class DMS, many optimal codes exist that are equivalent to schemes as defined in (2-24) and, therefore, easy to implement.

We will now set out to prove these statements. First we repeat

Tunstall's theorem.

Theorem 2-4 (Tunstall [3]): $\mathcal{M}_T(M)$ is a proper and complete segment set over the source alphabet $\mathcal{U}$ with M segments, $M < \infty$. If $\mathcal{M}_T(M)$ is obtained by the following algorithm, then the average segment length of $\mathcal{M}_T(M)$, with respect to the source probability vector $P(\cdot)$, is maximal over all proper and complete segment sets of size M.

Algorithm:

$$\mathcal{M}_T(|\mathcal{U}|) \triangleq \mathcal{U},$$

$$\mathcal{M}_T(M + |\mathcal{U}| - 1) \triangleq (\mathcal{M}_T(M) - \{\underline{u}\}) \cup \{\underline{u}u \,|\, u \in \mathcal{U}\}$$

where $\underline{u} \in \mathcal{M}_T(M)$ with $P(\underline{u}) = \max\{P(\underline{v}) \,|\, \underline{v} \in \mathcal{M}_T(M)\}$.

We say that $\mathcal{M}_T(M)$ is <u>extended</u> at $\underline{u}$.

We will give a alternative proof of this theorem, using what we call the "<u>test lemma</u>".

Lemma 2-6 (test lemma): Let $\mathcal{M}$ be a proper and complete segment set over $\mathcal{U}$, $(|\mathcal{M}| = M < \infty)$. Over all possible segment sets $\mathcal{M}$ has the largest average segment length with respect to a probability vector $P(\cdot)$ <u>iff</u> the P-probability of a segment in $\mathcal{M}$ does <u>not</u> exceed the P-probability of any proper prefix of any segment in $\mathcal{M}$.

The proof of this lemma is given in Appendix VII. Now it remains to show that $\mathcal{M}_T(M)$ satisfies the test lemma.

Proof: Consider $\mathcal{M}_T(|\mathcal{U}|)$. The only proper prefix is the empty string $\underline{u}^0$. Now $P(\underline{u}^0) \triangleq 1$, so the test lemma shows that the set is maximal. (Also, $\mathcal{M}_T(|\mathcal{U}|)$ is the only possible set). Assume that $\mathcal{M}_T(M_0)$ is maximal. Set $M = M_0 + |\mathcal{U}| - 1$. Let $\underline{u} \in \mathcal{M}_T(M_0)$ be the segment that is extended. The set of proper prefixes for $\mathcal{M}_T(M)$ consists of the set of proper prefixes for $\mathcal{M}_T(M_0)$ plus the segment $\underline{u}$. Since $P(\underline{u})$ is the largest probability over the set $\mathcal{M}_T(M_0)$ and $P(\underline{u}u) < P(\underline{u})$, $u \in \mathcal{U}$, it follows with the test lemma that $\mathcal{M}_T(M)$ is also maximal. This proves the Tunstall algorithm, Theorem 2-4.

From the definition (2-28) it follows that the set $\mathcal{M}(n)$ also satisfies the test lemma if $\tilde{P}(u) = P(u)$ for all $u \in \mathcal{U}$. This holds because for every proper prefix $\underline{v}$, $P(\underline{v}) > 2^{-\alpha n}$ and for every segment $\underline{u}$ in the set, $P(\underline{u}) \leq 2^{-\alpha n}$. It is not hard to see that if $\tilde{P}(u)$ is "close" to $P(u)$ then the test lemma still holds, proving the second claim above. Appendix VIII shows this result in more detail.

2.10. Conclusions and remarks.

We conclude that the variable-to-fixed length coding schemes that were described and analysed in this chapter perform well for Markov sources. Their rate can be close to the source entropy, and the storage and computational complexities are low.

A nice advantage of these schemes is that they are robust. When the letter probabilities of the Markov source change slightly the difference between the code rate and the source entropy does not increase too quickly as follows from Theorem 2-1.

We remark that it is necessary, to guarantee a good performance, that both the encoder and the decoder keep track of the state of the source. This means that the first state of the source has to be known by both.

Although it is important that for growing array lengths and growing stepvalues the entropy can be achieved, it turns out that sometimes, for reasonably small stepvalues and array lengths, schemes can be found with a rate very close to source entropy. For our "example source" with the steps as in Figure 2-3, we find for $n = 2$ a scheme with rate $R = 1.04591$ bit/source letter, which is only 0.02950 bit away from the entropy of the source. It is not known whether such good and simple schemes exist for all Markov sources.

3. ARITHMETIC CODES.

3.1. Introduction.

The arithmetic coding scheme is a stream code based on Elias' algorithm [5]. It is a noiseless compression technique applicable in all situations where the source statistics are known. Like the scheme described in chapter 2, this scheme computes the codeword from the message and back.

The practical application of the Elias algorithm is hampered by the fast growing arithmetical precision requirements. Pasco [10] solved this problem using a rounding technique. He showed that, if properly performed, rounding is allowable and incurs a small redundancy penalty. At the same time Rissanen [9] introduced a similar technique that used an exponential table to avoid the precision problem. Other authors, eg. Jones [17], Rubin [18], and Guazzo [19], presented other arithmetic coding schemes.

In [20] Rissanen and Langdon generalized the arithmetic coding technique, and specifically treated the problem of optimizing the table values. However, they did not deal with the problem of designing a scheme for a given source. Also, because they optimized the table, the flexibility of the algorithm was lost.

We approach the code design starting from a finite size and precision table, thus retaining the flexibility. The upperbounds on the code redundancy, or inefficiency, that result are functions of the two table parameters, namely the size of the table and the precision of a table entry.
In the sections 3.2 and 3.3 we describe the Elias algorithm. In these sections, the important notion of source- and code intervals are introduced. Also the decoding of these schemes is discussed. Section 3.4 describes some previous arithmetic coding schemes like Pasco's and Rissanen's finite pre-

cision algorithms. It also introduces a new carry-blocking technique. We need this to allow the transmission of parts of the codeword before the whole codeword is completed. In section 3.5 our finite precision and muli-plication-free schemes are described. Two methods are given. The P-method has the better efficiency of the two and the Q-method is the fastest. In section 3.6 we discuss the decodability criteria for these methods. In the next two sections these criteria are used to give two code designs for each method and to bound the resulting redundancies. Section 3.9 describes "fast" designs. It is shown that the computational burden can be reduced with an acceptable penalty in redundancy increase. The complexity of these schemes is exposed in section 3.10 and the next section gives some numerical results. Section 3.12 describes the implementation of the encoder and decoder. Here we discuss the trade-off between the extra size of the adder-unit and the redundancy for low entropy sources. The last section summarizes the results of this chapter.

3.2. The Elias algorithm.

The Elias algorithm represents source strings and codewords by subintervals of [0, 1). The coding scheme is defined by relating source intervals to code intervals.

A source in the class SES is defined by a finite, ordered alphabet $\mathcal{U} = \{0, 1, \ldots, c-1\}$ and the string probabilities $P(\underline{u}^n)$, for $n = 0, 1, 2, \ldots$. For a detailed treatment of the class of discrete stationary and ergodic sources we refer to [1]. The most important properties are

stationarity; the probability of a string is independent of the time origin, and

ergodicity; this says that time averages of functions over all sample

source output sequences equal the ensemble averages, except possibly for a set of sequences with probability zero.

The (block-) entropy of a string of n symbols is defined as:

$$H(P(\underline{U}^n)) \triangleq - \sum_{\underline{u}^n \in \mathcal{U}^n} P(\underline{u}^n) \log P(\underline{u}^n)$$

The per letter source entropy is defined as the limit:

$$H_\infty(P) \triangleq \lim_{n \to \infty} \frac{1}{n} H(P(\underline{U}^n))$$

The most important result is that the entropy of a source in the class SES can be approached arbitrarily close. (Shannon - McMillan, see [1]).

We can extend the ordering on single letters of $\mathcal{U}$ to a lexicographical ordering of strings over $\mathcal{U}$. Now $Q(\underline{u}^n)$ is the cumulative probability given by:

$$Q(\underline{u}^n) = \sum_{\underline{v}^n < \underline{u}^n} P(\underline{v}^n).$$

Define the source interval $I(\underline{u}^n)$ as:

$$I(\underline{u}^n) = [Q(\underline{u}^n),\ Q(\underline{u}^n) + P(\underline{u}^n)) \qquad (3\text{-}1)$$

See Figure 3-1.

Now, since $Q(\underline{u}^n) + P(\underline{u}^n)$ equals $Q(\underline{v}^n)$, where $\underline{v}^n$ is the "next" string in the ordering, we observe that the set $\{I(\underline{u}^n) \mid \underline{u}^n \in \mathcal{U}^n\}$, $n = 0, 1, \ldots$, completely subdivides the unit interval.

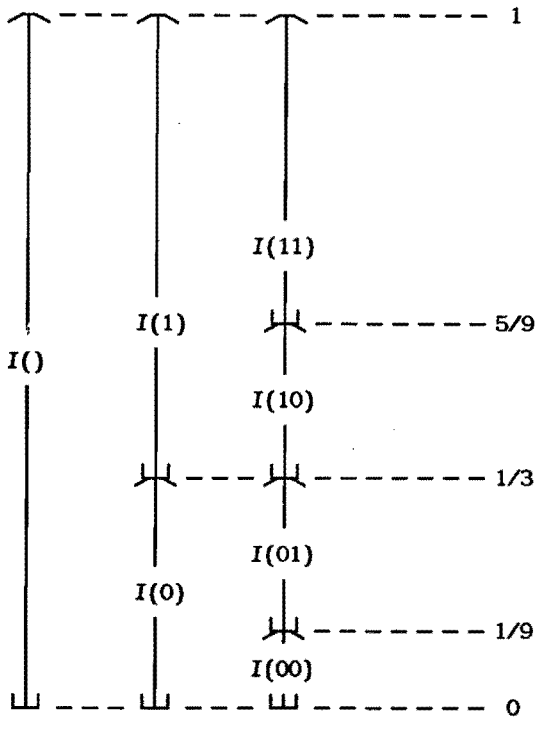

Figure 3-1. Subdivision of [0, 1) by $I(\underline{u}^0)$, $I(\underline{u}^1)$, $I(\underline{u}^2)$ with a binary memoryless source, $P(0) = 1/3$.

Elias' main contribution is the following recursive generation of the successive source intervals $I(\underline{u}^i)$, $1 \leq i \leq n$. Denote by $P(u|\underline{u}^i)$ the conditional symbol probability and by $Q(u|\underline{u}^i)$ the conditional cumulative probability, or

$$P(u|\underline{u}^i) = \frac{P(\underline{u}^i u)}{P(\underline{u}^i)}, \quad P(\underline{u}^i) > 0,$$

$$Q(u|\underline{u}^i) = \sum_{v<u} P(v|\underline{u}^i), \quad P(\underline{u}^i) > 0.$$

Then

$$Q(\underline{u}^{i+1}) = Q(\underline{u}^{i}) + P(\underline{u}^{i}) \cdot Q(u_{i+1} | \underline{u}^{i}), \tag{3-2}$$

$$P(\underline{u}^{i+1}) = P(\underline{u}^{i}) \cdot P(u_{i+1} | \underline{u}^{i}). \tag{3-3}$$

The repeated application of (3-2) and (3-3), starting with the empty string $\underline{u}^{0}$, $Q(\underline{u}^{0}) = 0$, $P(\underline{u}^{0}) = 1$, gives us the interval $I(\underline{u}^{n})$ defined by (3-1).

So we now have an invertible mapping from source strings into subintervals of $[0, 1)$. The subdivision is determined by the source probabilities. The usual description of this algorithm, c.f. [21] and [10], now continues by constructing a codeword $\underline{x}^{m}$ from a point $a \in I(\underline{u}^{n})$, where a can be represented by an m digits d-ary fraction. The codeword length m is shown to be upperbounded by $\lceil -\log_d P(\underline{u}^{n}) \rceil$, however, the codewords that result from this algorithmic construction cannot be concatenated. The standard solution is to append a length defining prefix, increasing the codeword length by about $\log_d n$, [21], [10]. We will give a different algorithmic construction that requires no more than one extra symbol above $\lceil -\log_d P(\underline{u}^{n}) \rceil$ to guarantee a prefix-free code.

Let $\mathfrak{X} = \{0, 1, \ldots, d-1\}$ be the finite code alphabet. Any d-ary prefix-free code can be seen as a d-ary subdivision of the unit interval as follows:

For $\underline{x}^{m} \in \mathfrak{X}^{m}$ define the rational number $x(m)$ by:

$$x(m) \triangleq \sum_{i=1}^{m} x_i \cdot d^{-i}.$$

Define the code interval $J(\underline{x}^{m})$ by:

$$J(\underline{x}^{m}) \triangleq [x(m), x(m) + d^{-m}).$$

Note that $\underline{x}^m$ is a prefix of $\underline{u}^\ell$ if and only if $J(\underline{x}^m) \supset J(\underline{u}^\ell)$. Thus we have the following lemma.

Lemma: the code is prefix-free if and only if no two code intervals have a point in common.

Our construction of codewords for the Elias message set assigns to each source string $\underline{u}^n$ a variable length code string $\underline{x}^m$ with

$$I(\underline{u}^n) \supset J(\underline{x}^m). \tag{3-4}$$

As can be seen in Figure 3-2.

The inclusion (3-4) defines a mapping from $\underline{x}^m$ to $\underline{u}^n$, but not from $\underline{u}^n$ to the codeword $\underline{x}^m$ without an additional rule that we will give later. Figure 3-2 gives an example of such a code. Note that some of the code intervals are not used. These unused intervals are called gaps. They result in an inefficiency of the scheme. Remember that a scheme can be inefficient even without gaps if the codeword lengths are not assigned correctly. The inefficiency of this coding scheme will be upperbounded using the following theorem.

Theorem 3-1: $\underline{u}^n$ is a source string with probability $P(\underline{u}^n)$. There exists a prefix-free code satisfying (3-4) such that the length m of the corresponding code string $\underline{x}^m$ satisfies:

$$m \leq \lceil -\log_d P(\underline{u}^n) \rceil + 1 < -\log_d P(\underline{u}^n) + 2.$$

Here $\lceil x \rceil$, where x is a real number, denotes the smallest integer not less than x.

Proof: consider the code defined as follows.

Set $m = \lceil -\log_d P(\underline{u}^n) \rceil + 1$ (3-5)

and $\alpha = d^{-m} \cdot \lceil d^m \cdot Q(\underline{u}^n) \rceil$ (3-6)

Let $\underline{x}^m$ be the codeword such that $x(m) = \alpha$, or

$$\alpha = \sum_{i=1}^{m} x_i \cdot d^{-i} \tag{3-7}$$

So $J(\underline{x}^m) = [\alpha, \ \alpha + d^{-m})$. We complete the proof by showing that $I(\underline{u}^n) \supset J(\underline{x}^m)$.

From (3-6): $\alpha \geq Q(\underline{u}^n)$ (3-8)

From (3-6): $\alpha + d^{-m} < Q(\underline{u}^n) + 2 \cdot d^{-m}$ (3-9)

From (3-5): $d^{-m} < d^{-1} \cdot P(\underline{u}^n)$ (3-10)

From (3-9), (3-10): $\alpha + d^{-m} < Q(\underline{u}^n) + \frac{2}{d} \cdot P(\underline{u}^n)$

$\leq Q(\underline{u}^n) + P(\underline{u}^n)$ (3-11)

From (3-8), (3-11): $I(\underline{u}^n) \supset J(\underline{x}^m)$ Q.E.D.

So $\mathop{E}_{\underline{U}^n}\{m\} < \mathop{E}_{\underline{U}^n}\{-\log_d P(\underline{U}^n)\} + 2,$

and for stationary sources we get

$$\frac{1}{n} \cdot \mathop{E}_{\underline{U}^n}\{m\} < \frac{1}{n} \cdot H(P(\underline{U}^n)) + \frac{2}{n} \tag{3-12}$$

which approaches the source entropy as $n \to \infty$.

Remark 1: The formulas (3-5), (3-6), and (3-7) give us the additional rule to make the codeword assignment unique. While this rule guarantees the decodability, it isn't always the best possible choice. See Figure 3-2 for a better selection of $J(\cdot)$, not in accordance with (3-5), but still decodable.

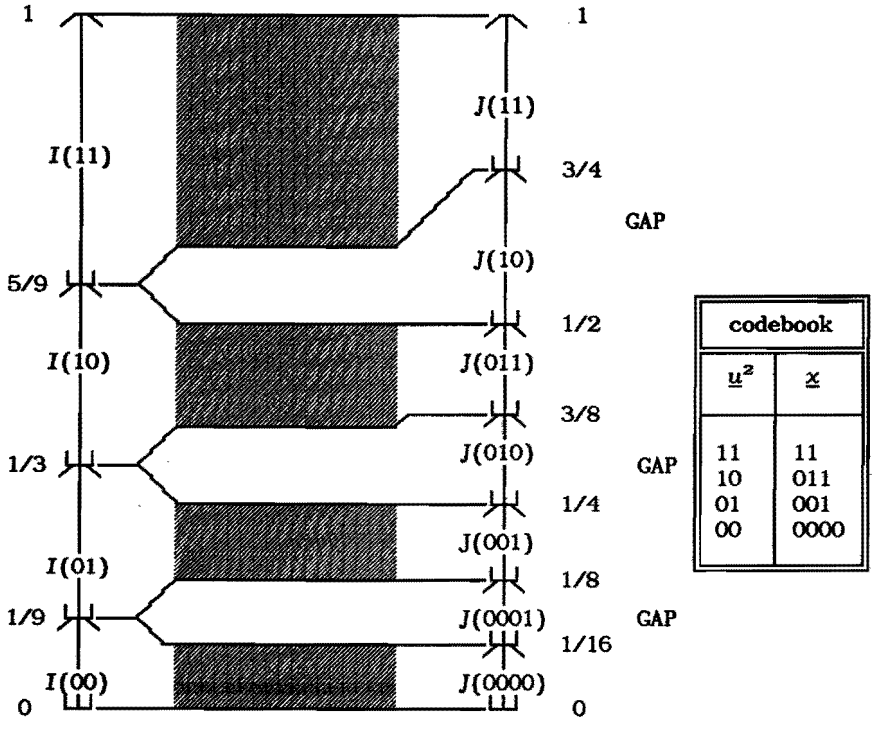

Figure 3-2. Codeword assignment.

So the Elias algorithm first computes the source interval and then, by Theorem 3-1, finds the corresponding code interval and codeword. With (3-12) we conclude that although this algorithm is not optimal, i.e. a Huffman code [2] would be better, its implementation as n becomes large is less complex and the resulting code rate still approaches the source entropy.

In the above description we formulated the algorithm as a FV (block) scheme. It can be made into a stream coding scheme if we set n equal to infinity. In this case we know with (3-12) that the source entropy will be achieved.

Now we have to deal with two new problems. The coding delay and the

arithmetical precision in (3-2) and (3-3) become infinite. In the next sections we shall treat these problems.

3.3. Partial encoding and decoding.

In this section we discuss the coding delay problem occurring when we use the Elias algorithm as a stream coding scheme. First observe that the successive source intervals are included in each other. The same holds for the code intervals, or

$$I(\underline{u}^j) \subset I(\underline{u}^i), \quad 0 \leq i \leq j \leq n$$
$$J(\underline{x}^\ell) \subset J(\underline{x}^k), \quad 0 \leq k \leq \ell \leq m \qquad (3\text{-}13)$$

So, if $I(\underline{u}^i) \subset J(\underline{x}^k)$ then by (3-4) and (3-13): $J(\underline{x}^m) \subset J(\underline{x}^k)$, so $\underline{x}^k$ is a prefix of $\underline{x}^m$ and can be transmitted.

Decoding is done by simulating the encoder, that is, the decoder tries to build its own source interval $I(\hat{\underline{u}}^n)$ in accordance with the received code string $\underline{x}^m$. For this purpose it uses the same formulas etc. as the encoder. We will give a recursive description of the decoding. Let $\underline{x}^\ell$ be the received part of the codeword. Let $\hat{\underline{u}}^i$ be decoded correctly from $\underline{x}^\ell$, $(\hat{\underline{u}}^i = \underline{u}^i)$. So, $I(\hat{\underline{u}}^i) \supset J(\underline{x}^\ell)$. Now let $m \geq k \geq \ell$ such that $\underline{x}^k$ is the shortest extension of $\underline{x}^\ell$ with $I(\hat{\underline{u}}^i\hat{u}) \supset J(\underline{x}^k)$ for some symbol $\hat{u} \in \mathcal{U}$. Then $\hat{u}_{i+1} = \hat{u} = u_{i+1}$, the corresponding source symbol. Also the decoder can decide whether or not a received code string part is sufficient to decode the next symbol, because if $\underline{x}^k$ is insufficient then no $\hat{u} \in \mathcal{U}$ exists with $I(\hat{\underline{u}}^i\hat{u}) \supset J(\underline{x}^k)$.

This partial encoding and decoding is a useful property of the scheme. However, the determination whether or not a code interval is included in a

source interval might require arbitrarily high precision arithmetic and arbitrarily long delays. In the next section we will introduce a modification that allows the use of bounded precision arithmetic.

3.4. Finite precision algorithms.

Pasco [10] gave the following description of the precision problem:

Suppose that all conditional probabilities $P(u|\underline{u})$ and $Q(u|\underline{u})$ are approximated by q digits precise numbers $\tilde{P}(u|\underline{u})$ resp. $\tilde{Q}(u|\underline{u})$. With (3-3) we observe that $\tilde{P}(\underline{u}^i)$ needs $q \cdot i$ digits, thus preventing the encoding of large source strings.

Pasco's solution was to replace (3-2) and (3-3) by a rounded down version:

$$\tilde{P}(\underline{u}^{i+1}) = \lfloor \tilde{P}(\underline{u}^i) \cdot \tilde{P}(u_{i+1}|\underline{u}^i) \rfloor_k \qquad (3\text{-}14)$$

$$\tilde{Q}(\underline{u}^{i+1}) = \tilde{Q}(\underline{u}^i) + \tilde{P}(\underline{u}^i) \cdot \tilde{Q}(u_{i+1}|\underline{u}^i) \qquad (3\text{-}15)$$

resulting in a source interval:

$$I(\underline{u}^n) = [\tilde{Q}(\underline{u}^n),\ \tilde{Q}(\underline{u}^n) + \tilde{P}(\underline{u}^n))$$

Here $\lfloor x \rfloor_k$ stands for the largest number $y \leq x$, where y is a k digits precise, floating point number. This effectively reduces the source interval lengths by a small amount, thus increasing the code rate. Pasco proved that:

$$\frac{1}{n} \cdot \mathbf{E}_{\underline{U}^n}\{m\} < \frac{1}{n} \cdot H(P(\underline{U}^n)) - \log_d(1 - d^{1-k}) + \frac{2}{n} \qquad (3\text{-}16)$$

where m is again the length of the codeword $\underline{x}^m$, see (3-12).

In this derivation the influence of the approximation of the probability vectors P and Q has not been taken into account. Pasco shows that if the effect on the codeword length due to this approximation is to be upperbounded by a value δ then it suffices to take q digits in the fractional part of $\tilde{P}(u|\underline{u})$ and $\tilde{Q}(u|\underline{u})$ such that:

$$q = \lceil \log_d \frac{1}{P_{min}} + \log_d \frac{1}{\delta} + 1 \rceil. \qquad (3\text{-}17)$$

Here P_{min} is the smallest probability over all vectors $P(u|\underline{u})$.

From (3-15) we note that we must add a $q + k$ digit floating point augend to $\tilde{Q}(\underline{u})$. Figure 3-3 depicts this process for a binary code alphabet. From this figure we see that, except from the occurring carry, only $q + k$

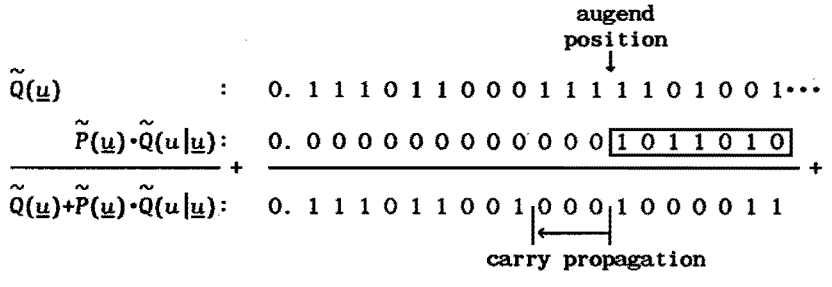

Figure 3-3. Encoding a symbol.

digits of $\tilde{Q}(\underline{u})$ are affected. We define the augend position to be the position just to the left of the first non-zero digit in the d-ary representation of $\tilde{P}(\underline{u})$, c.f. Figure 3-3. Now we would like to be able to transmit those symbols of $\tilde{Q}(\underline{u})$ that are to the left of this augend position as soon as they are generated. However, since a carry may change an arbitrary number of these digits this cannot be done. To prevent this carry propagation

Langdon and Rissanen [22] described a carry-blocking technique.

In short: they retain the last r ($r \cong 16$) digits to the left of the augend position. All symbols to the left of these r digits are not allowed to be changed anymore and can be transmitted. As long as the r digits, just before adding $\tilde{P}(\underline{u})\cdot\tilde{Q}(u|\underline{u})$, are not all equal to d-1, a possible carry will be stopped somewhere in these r positions. If all digits are d-1's, an extra zero is inserted at the augend position. It is obvious that for each addition at most one carry can occur. So, a resulting carry will never propagate further than r symbols to the left of the augend position. The decoder removes this inserted symbol and processes any occuring carry thatwas blocked in it. This technique fails, since, without extra precautions, r or more d-1's can be transmitted without an extra inserted zero and the decoder cannot distinguish between real code symbols and inserted ones.

We propose a somewhat different method, that is more in line with the whole algorithm. Just like Langdon and Rissanen, we save the last r symbols directly to the left of the augend position in a special carry blocking register C. Whenever, before encoding the next symbol, this register contains only d-1's, we shift the augend to the right, (or, equivalently, the adder register $\tilde{Q}(\underline{u})$ and the C register to the left), until the C register contains a symbol different from d-1.

So, whenever we encode a symbol any occuring carry will be stopped in the C register. This method retains the relative positions of the different source intervals, (it only shortens them when necessary). The decoder, since it simulates the encoder, knows when these shifts occur and performs them too.

Now, how often does this occur?

The symbol probability at the output of the encoder will be about $1/d$ if the encoder is efficient. Thus, the probability of a shift event is

about d^{-r}. So, these events happen about once per d^r source symbols. Assuming the same probabilities in the $\tilde{Q}(\underline{u})$ register, we need circa $\frac{d}{d-1}$ shifts per event. Experiments indicate that this is a slightly conservative estimate.

Another adapted version of the Elias algorithm is described by Rissanen in [9]. The scheme given there also uses fixed precision arithmetic as in Pasco's scheme. It differs from the previous coding schemes by the fact that the coding scheme builds its codewords backwards. By this we mean that if $\underline{x}^m$ is the codeword for $\underline{u}^n$ then u_n is the first symbol that can be decoded from $\underline{x}^m$ and so on until, finally, u_1 is decoded. This implies that the partial encoding and decoding technique described in section 3.3 cannot be used. It also means that the carry-over problem cannot occur because the augend is added to the most significant part of $\tilde{Q}(\underline{u})$ as we will see.

The scheme uses a rational-valued approximated exponential table $e(x) \cong 2^x$ in the computation of $\tilde{P}(\underline{u}^i)$. Rissanen uses a finite precision, approximated cumulative symbol probability vector $\tilde{Q}(u_{i+1})$ and instead of the symbol probability vector $P(u_{i+1})$ he uses a "length parameter" vector $l(u_{i+1})$ that is a rational-valued approximation to $-\log_2 P(u_{i+1})$ with q binary digits in the fractional part.

The $l(u)$ must satisfy the following condition for some $\epsilon > 0$:

$$\sum_{u \in \mathcal{U}} 2^{-l(u)} \leq 2^{-\epsilon}. \tag{3-18}$$

Let $p(u)$ be a rational number such that:

$$p(u) = 2^{-l(u)+\epsilon(u)}; \quad \frac{2\epsilon}{3} \leq \epsilon(u) \leq \epsilon, \quad u \in \mathcal{U}.$$

Now the finite precision cumulative probabilities are:

$$\tilde{Q}(u) = \sum_{v<u} p(u); \quad u \in \mathcal{U}$$

and the table $e(x)$ is defined for all q bit fractions x such that:

$$e(x) = 2^{x+\delta(x)}; \quad 0 \leq \delta \leq \frac{\epsilon}{2}.$$

Assume that $p(u)$, $\tilde{Q}(u)$, and $e(x)$ can be described with r digits in their fractional parts, where r will be a function of ϵ and q. The formulas become:

$$\begin{aligned}
\hat{Q}(\underline{u}^{i+1}) &= \hat{Q}(\underline{u}^{i}) + \tilde{P}^{-1}(\underline{u}^{i+1})\cdot\tilde{Q}(u_{i+1}) \\
L(\underline{u}^{i+1}) &= L(\underline{u}^{i}) + l(u_{i+1}) \\
\tilde{P}^{-1}(\underline{u}^{i+1}) &= 2^{\lfloor L(\underline{u}^{i+1})\rfloor}\cdot e(L(\underline{u}^{i+1}) - \lfloor L(\underline{u}^{i+1})\rfloor) \\
\hat{Q}(\underline{u}^{0}) &= 0 \\
L(\underline{u}^{0}) &= 2\cdot r
\end{aligned}$$

Rissanen proves that

$$\mathbf{E}_{\underline{U}^n}\{\frac{\log_2\hat{Q}(\underline{U}^n)}{n}\} < \frac{1}{n}\cdot H(P(\underline{U}^n)) + \epsilon + 2^{-q} + O(\frac{r}{n}). \qquad (3\text{-}19)$$

where $f(n) = O(g(n))$ means that there is a constant c such that $|f(n)| \leq c\cdot|g(n)|$ for all n sufficiently large.

In a later paper, Rissanen and Langdon [20], generalize these coding schemes. They discuss the differences and similarities between the First In - First Out type of scheme, like the Elias scheme and the Last In - First Out coding scheme of [9]. Pasco, in his thesis, also treats this aspect.

Rissanen and Langdon then propose to precompute $\tilde{P}^{-1}(\underline{u}^{i+1})\cdot\tilde{Q}(u_{i+1})$ and

to store these in an augend table. The remainder of this article discusses the selection of optimal values for these augends given a length parameter vector. A severe disadvantage of this optimization is that it is a very complex operation and the optimal values strongly depend on the length parameters. This implies that this technique can hardly be used for sources with memory. In this case namely, we either precompute and store a table for every possible length parameter vector or we have to perform a complex optimization algorithm every time the vector changes.

3.5. Multiplication-free codes.

Rissanens arithmetic code [9] uses the exponential table only to manage the precision problem. As an extra it eliminates the multiplication present in (3-3). In the remainder of this chapter we introduce and discuss arithmetic coding schemes where the exponential table will be used to limit the precision requirements as well as to eliminate all multiplications. The justification for the latter is the fact that a multiplication operation is inherently more complex than an addition.

Not only shall we state the encoding and decoding equations but we will give algorithms that design the code for a given probability vector. We will consider the efficiency of these designs as well as their complexity. The following example introduces the exponential table.

Example 1: Let A[i] be the table approximating λ^{-i}, ($\lambda > 1$), and a and b be positive reals. Now choose two integers i and j such that:

$$a \cong A[i] \cong \lambda^{-i},$$

$$b \cong A[j] \cong \lambda^{-j}.$$

So, i and j are proportional to the logarithm of a resp. b.

The multiplication $a \cdot b$ is approximated by the table value $A[i+j] \cong \lambda^{-i-j}$.

The table A, as will be explained below, is generated using a finite number of elements and each element has a finite precision. So there are two sources for imprecisions in the representation of a number.

Because the table is indexed by integers, not all the "logarithms" are exactly representable, and, because of the finite precision, the "multiplication" $A[i] * A[i] \triangleq A[i+j]$ is inexact.

These imprecisions result in an inefficiency of the code; this will be the topic of the next chapters.

Summarizing: the unbounded precision problem is solved because we add table entries of a fixed and finite precision and the complexity of the multiplication is reduced since it is replaced by table referencing and an addition.

We will now define the table and then proceed to describe the algorithm. Let $\tilde{A}[i]$ be a finite precision d-ary table, $0 \leq i < N$, for some integer N. $\tilde{A}$ is nonincreasing in i. N denotes the <u>length</u> of the table. This table approximates, in some way, the exponential function λ^{-i}, where

$$\lambda \triangleq d^{1/N}. \tag{3-20}$$

We extend this table to $A[\ell]$ for all integers ℓ by the following:

$$\begin{aligned} A[\ell] &\triangleq \tilde{A}[\ell] \qquad \text{if } 0 \leq \ell < N \\ &\triangleq d^{-j} \cdot \tilde{A}[i], \qquad \ell = jN + i,\ 0 \leq i < N \end{aligned} \tag{3-21}$$

Note that, since $\tilde{A}$ is a d-ary table, d^{-j} is a simple shift over j places. Now there exist two reals α and β such that for all i, $0 \leq i < N$:

$$\alpha \cdot \lambda^{-i} \leq \tilde{A}[i] \leq \beta \cdot \lambda^{-i} \qquad (3\text{-}22)$$

Of course this also holds for $A[\ell]$ as defined in (3-21).

Example 2: Define $\tilde{A}[i]$ as $d^{-k} \cdot \lceil d^{k} \cdot \lambda^{-i} \rceil$, where k is a positive integer. It is easily seen that $\lambda^{-i} \leq A[i] < (1 + d^{1-k}) \cdot \lambda^{-i}$, so $\alpha = 1$, $\beta = 1 + d^{1-k}$. Every table entry is now described with a k digit precision.

$A[\ell]$ will be used to facilitate the multiplications in the algorithm. We now describe the encoding formulas for these schemes. There are two possibilities:

P-method: This construction uses approximations to the conditional symbol probabilities. Define integer stepvalues $s(u|\underline{u}^i)$, preferably such that $A[s(u|\underline{u}^i)] \cong P(u|\underline{u}^i)$. The conditions on and methods for selecting the stepvalues are discussed in a later section. We also need an integer $S(\underline{u}^i)$ It performs the role of $P(\underline{u}^i)$ in the algorithm, or better, $A[S(\underline{u}^i)] \cong P(\underline{u}^i)$. The algorithm computes an approximation to $Q(\underline{u}^i)$ named $B(\underline{u}^i)$. The formulas are, cf. (3-2), (3-3):

$$B(\underline{u}^{i+1}) = B(\underline{u}^i) + \sum_{v<u_{i+1}} A[S(\underline{u}^i) + s(v|\underline{u}^i)] \qquad (3\text{-}23)$$

$$S(\underline{u}^{i+1}) = S(\underline{u}^i) + s(u_{i+1}|\underline{u}^i) \qquad (3\text{-}24)$$

With initial values: $B(\underline{u}^0) = S(\underline{u}^0) = 0$.

Q-method: This construction uses approximated cumulative probabilities as well. Apart from the stepvalues $s(u|\underline{u}^i)$ we also need integer stepsums $T(u|\underline{u}^i)$ such that $A[T(u|\underline{u}^i)] \cong Q(u|\underline{u}^i)$. The formulas become:

$$B(\underline{u}^{i+1}) = B(\underline{u}^i) + A[S(\underline{u}^i) + T(u_{i+1}|\underline{u}^i)] \qquad (3\text{-}25)$$

$$S(\underline{u}^{i+1}) = S(\underline{u}^i) + s(u_{i+1}|\underline{u}^i) \qquad (3\text{-}26)$$

$$B(\underline{u}^0) = S(\underline{u}^0) = 0. \qquad (3\text{-}27)$$

Now we define the source intervals for the two methods. With Theorem 3-1 we can find the code intervals and thus complete the code description, since we then have a mapping from the source strings into the codewords, c.f. chapters 3.2 and 3.3.

First, consider the two source strings $\underline{u}^i$ and $\underline{v}^i$ with $\underline{u}^{i-1} = \underline{v}^{i-1}$, $v_i = u_i + 1$. From (3-23) and (3-24) we get for the P-method: $B(\underline{v}^i) = B(\underline{u}^i) + A[S(\underline{u}^i)]$. It seems reasonable to stack the intervals, because then we leave no gaps in between and, as will be discussed in the next section, we do not want the intervals to overlap. Define:

$$I(\underline{u}^i) \triangleq [B(\underline{u}^i),\ B(\underline{u}^i) + A[S(\underline{u}^i)]) \qquad (3\text{-}28)$$

both for the P- and the Q-method. See figure 3-4.

Remark 2: If, in the two methods, the stepvalues and stepsums are reasonable approximations to their corresponding probabilities, then $B(\underline{u}^i) \cong Q(\underline{u}^i)$ and $A[S(\underline{u}^i)] \cong P(\underline{u}^i)$ and the source interval of (3-28) approximates the interval as defined by (3-1).

Remark 3: Although we use the same symbols in (3-23), (3-24) and (3-25)-(3-27) there is no numerical correspondence. Wherever it is clear from the context, we shall use symbols like B, S, and s without discrimination.

Remark 4: The augend $\Sigma A[S(\underline{u}) + s(v|\underline{u})]$ in (3-23) is computed more accurately than the corresponding term $A[S(\underline{u}) + T(u|\underline{u})]$ in (3-25). So, we may expect the P-method to be a better approximation to the Elias algorithm and thus have a better performance than the Q-method. However, the augend in (3-23) takes more work to compute if the source alphabet is large. These statements will be quantified in the coming sections.

Remark 5: Instead of (3-5) - (3-7) to find the codeword we could use

the interval boundary $B(\underline{u}^i)$ as the codeword for $\underline{u}^i$. In the above Example 2 this would introduce less than k extra digits per codeword, which is negligible when the source string length becomes large.

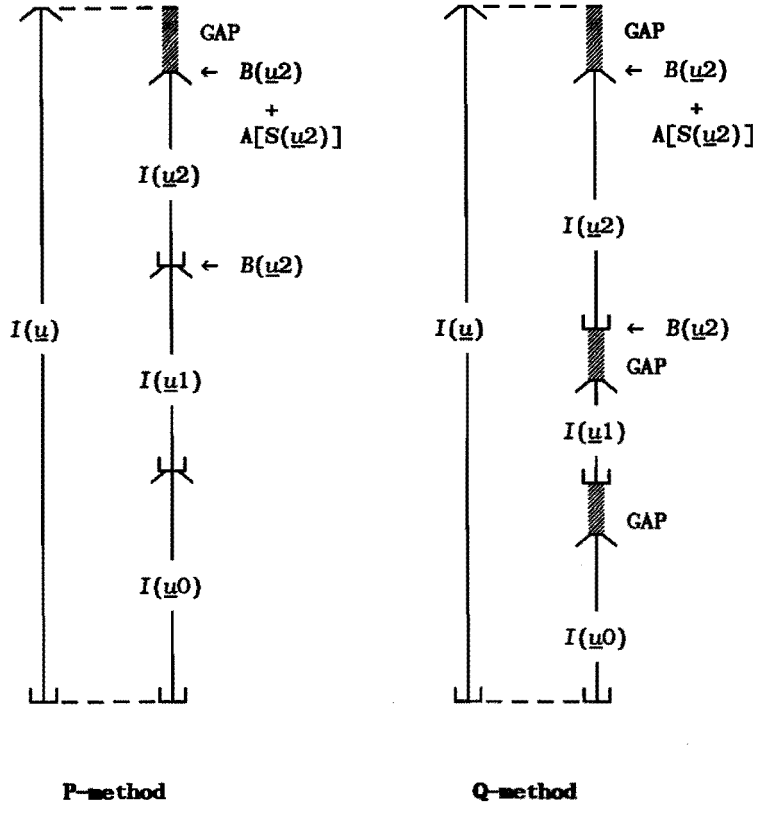

Figure 3-4. The source intervals.

3.6. Local and global tests.

In this section we discuss the condition for decodability. A code interval uniquely specifies a source string of length n, if it is included in exactly one source interval $I(\underline{u}^i)$. If the source intervals $I(\underline{u}^i)$ and $I(\underline{v}^i)$

do not overlap then the code is certainly decodable. In the case of the Elias algorithm, this condition is satisfied because of the definition (3-1). For the P- and Q-method arithmetic codes this restricts the choice of the code parameters $s(u|\underline{u})$ and $T(u|\underline{u})$.

Two types of <u>overlap prevention</u> are needed:

i) Let $\underline{u}^{i-1} = \underline{v}^{i-1}$, $v_i = u_i + 1$. So $\underline{v}^i$ is the string "next to" $\underline{u}^i$. The intervals $I(\underline{u}^i)$ and $I(\underline{v}^i)$ are adjacent and they do not overlap if (with (3-28)):

$$B(\underline{v}^i) \geq B(\underline{u}^i) + A[S(\underline{u}^i)] \qquad (3\text{-}29)$$

See Figure 3-5.

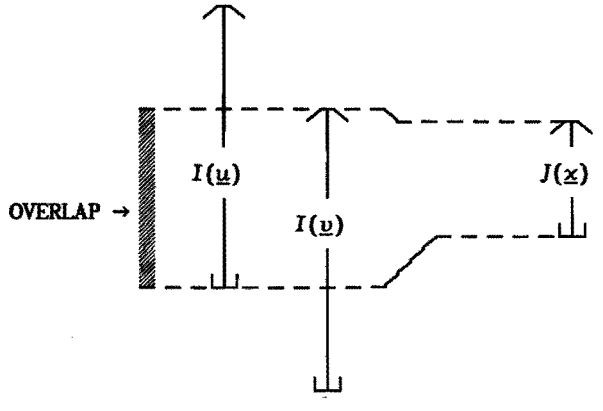

Figure 3-5. Decoding error due to overlapping intervals. $J(\underline{x})$ fits both in $I(\underline{u})$ and in $I(\underline{v})$.

ii) Let $\underline{u}^{i-1}$ be any source string. Set $u_i = \omega$. Here $\omega \triangleq c - 1$ is the <u>largest source symbol</u>. If the interval $I(\underline{u}^i)$ is not totally included in

$I(\underline{u}^{i-1})$ then an overlap with an other interval $I(\underline{v}^{i})$ can occur. See Figure 3-6. If $I(\underline{u}^{i})$ exceeds $I(\underline{u}^{i-1})$ then continuations of $\underline{u}^{i}$ and $\underline{v}^{i}$, where $\underline{v}^{i}$ is the string "next to" $\underline{u}^{i}$, might be assigned the same code interval. With (3-28) we get the condition:

$$B(\underline{u}^{i-1}) + A[S(\underline{u}^{i-1})] \geq B(\underline{u}^{i}) + A[S(\underline{u}^{i})] \qquad (3\text{-}30)$$

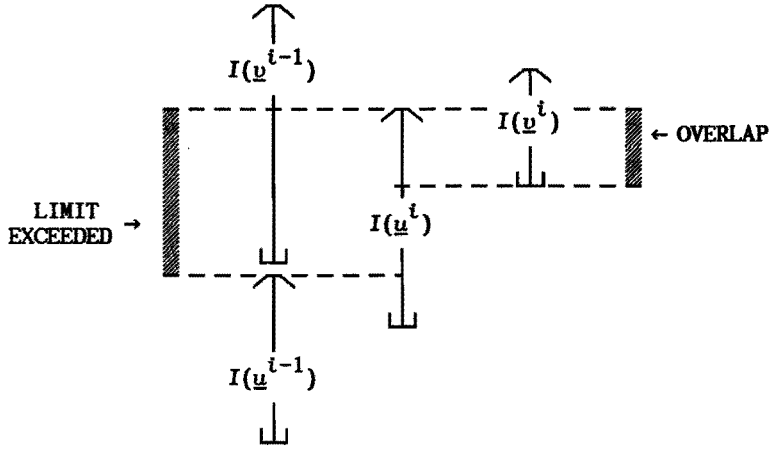

Figure 3-6. Decoding error due to exceeding an interval bound. $I(\underline{u}^{i})$ exceeds $I(\underline{u}^{i-1})$ and conflicts with $I(\underline{v}^{i})$.

The conditions (3-29) and (3-30), if satisfied for all $i \leq n$, are a sufficient condition for decodability. Because we want to use the arithmetic schemes as stream coding schemes, i.e. $n = \infty$, we require (3-29) and (3-30) to hold for all i.

We study the restrictions that the conditions (3-29) and (3-30) impose on the parameters of the P- and Q-methods. The Lemmas 3-1 and 3-2 are the

translations of (3-29) and (3-30) to the P-method and the Q-method. The conditions in these two lemmas depend on the "local position" $S(\underline{u}^i)$, and the actual table values $A[\cdot]$.

In some cases it is required that the code parameters $s(\cdot|\cdot)$ and $T(\cdot|\cdot)$ are selected such that a decodable code results independent of the local position. This is the case if we want to select the code parameters a-priori and use them for the encoding of all the source symbols. This is studied next and the results are stated in the Lemmas 3-3 and 3-4.

Lemma 3-1: If for all $i < n$ and $\underline{u}^i \in \mathcal{U}^i$ holds:

$$A[S(\underline{u}^i)] \geq \sum_{u \in \mathcal{U}} A[S(\underline{u}^i) + s(u|\underline{u}^i)] \qquad (3\text{-}31)$$

then the P-method code is decodable.

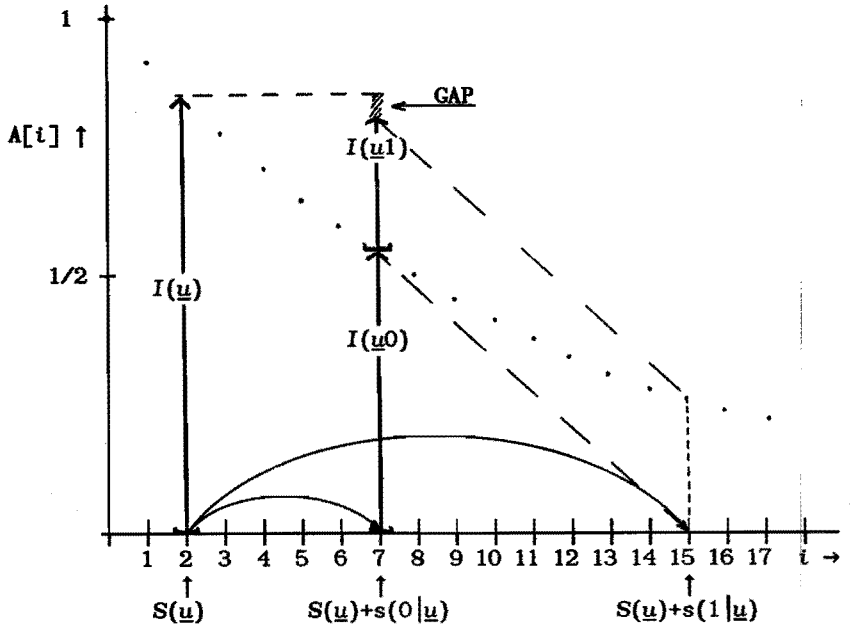

Figure 3-7. P-method interval generation.

Remark 6: Formula (3-31) is called the local P-test at position $S(\underline{u}^i)$. See Figure 3-7.

Proof: (3-29) is satisfied due to the definition (3-28) of the source intervals. For $B(\underline{u}^i)$ and $S(\underline{u}^i)$ with $u_i = \omega$ we may write, (see (3-23) and (3-24)):

$$B(\underline{u}^i) = B(\underline{u}^{i-1}) + \sum_{u<\omega} A[S(\underline{u}^{i-1}) + s(u|\underline{u}^{i-1})]$$
$$S(\underline{u}^i) = S(\underline{u}^{i-1}) + s(\omega|\underline{u}^{i-1})$$

Substituting this in (3-30) and subtracting $B(\underline{u}^{i-1})$ from both sides gives us (3-31). Q.E.D.

Lemma 3-2: If for all $i < n$, $\underline{u}^i \in \mathcal{U}^i$, $u \in \mathcal{U}$ holds

$$A[S(\underline{u}^i) + T(u+1|\underline{u}^i)] \geq A[S(\underline{u}^i) + T(u|\underline{u}^i)] + A[S(\underline{u}^i) + s(u|\underline{u}^i)] \qquad (3\text{-}32)$$

and also

$$T(\omega+1|\underline{u}^i) \geq 0 \qquad (3\text{-}33)$$

then the Q-method code is decodable.

Remark 7: The formulas (3-32) and (3-33) are called the local Q-test at position $S(\underline{u}^i)$. See Figure 3-8.

Proof: First we show that (3-29) holds if (3-32) holds. Let $\underline{u}^i$ and $\underline{v}^i$ be defined as in (3-29). Then from (3-25) and (3-26):

$$B(\underline{u}^i) = B(\underline{u}^{i-1}) + A[S(\underline{u}^{i-1}) + T(u_i|\underline{u}^{i-1})]$$
$$B(\underline{v}^i) = B(\underline{u}^{i-1}) + A[S(\underline{u}^{i-1}) + T(u_i+1|\underline{u}^{i-1})]$$
$$S(\underline{u}^i) = S(\underline{u}^{i-1}) + s(u_i|\underline{u}^{i-1})$$

Substituting this in (3-29) and subtracting $B(\underline{u}^{i-1})$ results in (3-32), so both are equivalent.

Now we show that (3-30) holds. Let $\underline{u}^{i-1}$ be any source string and $u_i = \omega$. Then, from (3-25) and (3-26):

$$B(\underline{u}^i) = B(\underline{u}^{i-1}) + A[S(\underline{u}^{i-1}) + T(\omega|\underline{u}^{i-1})]$$
$$S(\underline{u}^i) = S(\underline{u}^{i-1}) + s(\omega|\underline{u}^{i-1})$$

After substituting this in (3-30) and subtracting $B(\underline{u}^{i-1})$ we obtain the Q-method equivalent of condition (3-30):

$$A[S(\underline{u}^{i-1})] \geq A[S(\underline{u}^{i-1}) + T(\omega|\underline{u}^{i-1})] + A[S(\underline{u}^{i-1}) + s(\omega|\underline{u}^{i-1})] \qquad (3\text{-}34)$$

Now, since $A[i]$ is non increasing in i, we obtain (3-34) from (3-32) and (3-33). So (3-30) is true if (3-33) holds and (3-32) holds for $u = \omega$.

Q.E.D.

As said before, these local conditions depend on the current position in the table and the table values. In what follows we remove this dependency and obtain more restrictive <u>global conditions</u>.

Lemma 3-3: (Global P-test). If for all $i < n$, $\underline{u}^i \in \mathcal{U}^i$, the stepvalues are selected such that

$$\sum_{u \in \mathcal{U}} \lambda^{-s(u|\underline{u}^i)} \leq \frac{\alpha}{\beta} \qquad (3\text{-}35)$$

holds, then the P-method code is decodable. For α and β see (3-22).

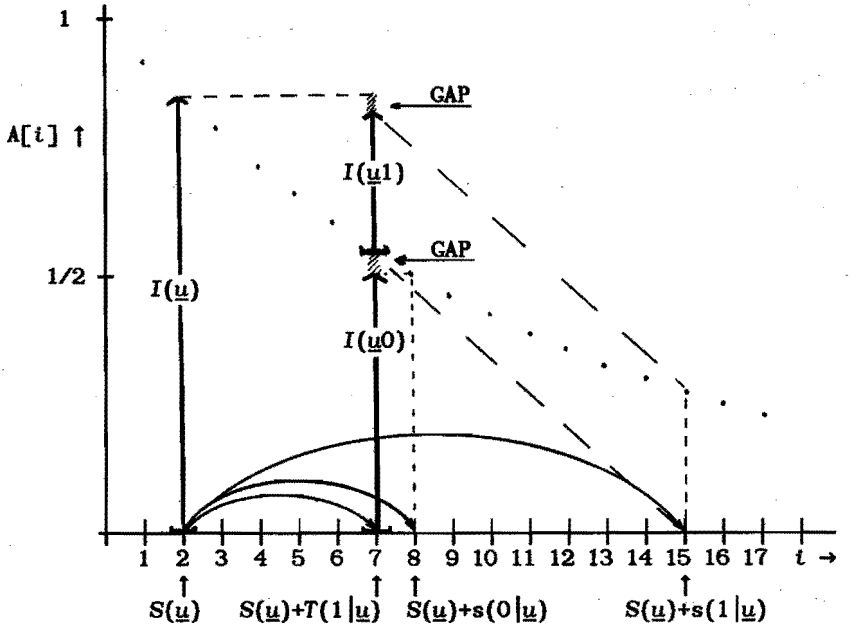

Figure 3-8. Q-method interval generation.

Lemma 3-4: (Global Q-test). If for all $i < n$, $\underline{u}^i \in \mathcal{U}^i$ the stepvalues are selected such that

$$\sum_{u \in \mathcal{U}} \left(\frac{\lambda\beta}{\alpha}\right)^{c-u} \cdot \lambda^{-s(u|\underline{u}^i)} \leq 1 \qquad (3\text{-}36)$$

holds, then stepsums $T(u|\underline{u}^i)$, for all $u \in \mathcal{U}$, can be found and the Q-method code is decodable.

The proofs of the Lemmas 3-3 and 3-4 are contained in the Appendix IX. Note the similarity between the two global tests and the Kraft inequality [1], [9], and especially between (3-35) and (3-18).

The four lemmas discussed in this section state conditions on the code

parameters such that decodable codes result. The conditions are not necessary and counterexamples are easily constructed (we give an example at the end of this section). However, with these conditions codes can be designed that perform well for most sources. The design of codes is the topic of the next section.

Example: let the table A contain $N = 8$ binary entries, each described with $k = 5$ digits, so:

A[0] = 1.0000, A[1] = 0.11110, A[2] = 0.11011, A[3] = 0.11001,
A[4] = 0.10111, A[5] = 0.10101, A[6] = 0.10100, A[7] = 0.10010.

So, $\alpha = 1$, $\beta = 1.0625$, and $\lambda = 1.0905$.

Let the source alphabet be $\mathcal{U} = \{ 0, 1, 2\}$ and assume that the source is memoryless, so all stepvalues s are functions only of the current source letter. Also, suppose that s(0) = 6 and s(1) = 13. Then, the smallest value for s(2), such that (3-35) is satisfied, is 44. However, by checking the actual table values and given that s(0) = 6 and s(1) = 13, we find that s(2) = 36 never conflicts with Lemma 3-1, but violates (3-35). So The global condition (Lemma 3-3) is not necessary. The same can be shown for Lemma 3-4.

3.7. Global and local designs.

A code design is a rule for selecting the stepvalues and stepsums, if needed, depending on the source probabilities, such that a decodable code results. One important distinction between designs is whether or not the local position $S(\underline{u}^i)$ influences the selection. If a rule selects the stepvalues independent of the local position and such that the global test is

satisfied, then we call this rule a global design. A local design results if we select the stepvalues according to the local test at the current position $S(\underline{u}^i)$. In this section we will give a design for each of the four cases mentioned above. At every time instant i+1 we assume that the probability vectors $P(u|\underline{u}^i)$ and $Q(u|\underline{u}^i)$ are available to the en- and decoder.

Design 1: (Global design for the P-method). For all $u \in \mathcal{U}$, set

$$s(u|\underline{u}^i) = \lceil \log_\lambda \frac{\beta}{\alpha} - \log_\lambda P(u|\underline{u}^i) \rceil. \tag{3-37}$$

Verification:

From (3-37) we find that $\lambda^{-s(u|\underline{u}^i)} \leq \frac{\alpha}{\beta} P(u|\underline{u}^i)$ so $\sum_{u\in\mathcal{U}} \lambda^{-s(u|\underline{u}^i)} \leq \frac{\alpha}{\beta}$ and the global P-test is satisfied. The code is decodable.

Remark 8: If u_{i+1} is the symbol to be encoded then (3-37) must be evaluated for all $u \in \mathcal{U}$, $u \leq u_{i+1}$. Then we can compute the new interval $I(\underline{u}^{i+1})$ by (3-23) and (3-24).

Design 2: (Global design for the Q-method). Set

$$s(u|\underline{u}^i) = \lceil (c-u)\cdot \log_\lambda \frac{\lambda\beta}{\alpha} - \log_\lambda P(u|\underline{u}^i) \rceil \tag{3-38}$$

$$T(u|\underline{u}^i) = \lceil (c-u)\cdot \log_\lambda \frac{\lambda\beta}{\alpha} - \log_\lambda Q(u|\underline{u}^i) \rceil \tag{3-39}$$

The verification of this design is contained in Appendix X.

Remark 9: If u_{i+1} is the symbol to be encoded then (3-38) and (3-39) must be computed for this value only. Compared with Design 1 this is much faster if $u_{i+1} \gg 1$ i.e., if the source alphabet is large.

Both designs make no use of the current position. In these global designs we take into account the worst-case deviations as defined by (3-22). In a local design we optimize the stepvalues by searching through the table for the smallest stepvalues that satisfy the local tests. We proceed with

two more examples.

Design 3: (Local design for the P-method). First we evaluate (3-37) for all $u \in \mathcal{U}$. Then we repeatedly decrease the stepvalues by one until any extra decrement would violate the local P-test. The search order could be: First decrease $s(0|\underline{u}^i)$ as much as possible, then $s(1|\underline{u}^i)$, etc. A slightly better method would be; select the smallest stepvalue and decrease it by one. Repeat this until no new decrement is possible.

Remark 10: Compared to Design 1, the amount of work has increased enormously.

Design 4: (Local design for the Q-method). Assume we want to encode u_{i+1}. First compute $T(u_{i+1}|\underline{u}^i)$ and $T(u_{i+1}+1|\underline{u}^i)$ by (3-39). Now find the smallest $s(u_{i+1}|\underline{u}^i)$ under the restriction of the local Q-test at position $S(\underline{u}^i)$.

Remark 11: This method is not much more complex than Design 2. Any search operation in the table can easily be performed in $\log_2 N$ operations since the table is ordered. See section 3.10.

3.8. Bounds on the redundancy.

In the previous sections we described some code designs and decodability criteria. Here we discuss the achievable rates of these designs. Actually, we are interested in the difference between the code rate and the source entropy. This quantity is known as the code redundancy.

Denote by $L(\underline{u}^n)$ the length of the codeword assigned to $\underline{u}^n$ by some code, then the rate of this code is given as

$$R(n) \triangleq \frac{\mathbf{E}_{\underline{U}^n}\{L(\underline{U}^n)\}}{n} \qquad (3\text{-}40)$$

and the redundancy is given by

$$r(n) \triangleq R(n) - \frac{1}{n} H(P(\underline{U}^n)) \tag{3-41}$$

For the length $L(\underline{u}^n)$ we show that, both in the P- and Q-methods, holds:

$$L(\underline{u}^n) \leq \frac{S(\underline{u}^n)}{N} + O(1).$$

Proof: by (3-28), the length of the source interval $I(\underline{u}^n)$ is given by $A[S(\underline{u}^n)]$. Theorem 3-1 then results in

$$L(\underline{u}^n) \leq \lceil -\log_d A[S(\underline{u}^n)] \rceil + 1$$

Which, with (3-20)-(3-22) results in

$$L(\underline{u}^n) \leq \lceil \frac{S(\underline{u}^n)}{N} - \log_d \alpha \rceil + 1$$

Q.E.D.

Now we may write:

$$\mathbf{E}_{\underline{U}^n}\{L(\underline{U}^n)\} \leq \mathbf{E}_{\underline{U}^n}\{-\log_d P(\underline{U}^n)\} + \mathbf{E}_{\underline{U}^n}\{\log_d \frac{P(\underline{U}^n)}{\lambda^{-S(\underline{U}^n)}}\} + O(1) \tag{3-42}$$

For stationary sources the first term on the righthand side equals the entropy $H(P(\underline{U}^n))$ and the second term is almost a divergence (see for instance Csiszár and Körner [23]). We will define the pseudodivergence:

$$\tilde{D}(P(\underline{U}^n)\|\lambda^{-S(\underline{U}^n)}) = \mathbf{E}_{\underline{U}^n}\{\log_d \frac{P(\underline{U}^n)}{\lambda^{-S(\underline{U}^n)}}\} \tag{3-43}$$

Now with (3-40), (3-41), and (3-43) we get:

$$r(n) \leq \frac{1}{n}\tilde{D}(P(\underline{U}^n)\|\lambda^{-S(\underline{U}^n)}) + O(1/n) \tag{3-44}$$

Remark 12: If we define the codewords as described in Remark 5, we still obtain (3-44), although the tolerance term $O(1/n)$ is larger in this case.

We will now upperbound this divergence. Therefore, assume the existence of a function $t(u) \triangleq \Gamma \cdot \Delta^{c-u}$, such that:

$$t(u_i)\cdot\lambda^{-s(u_i|\underline{u}^{i-1})} \geq P(u_i|\underline{u}^{i-1}) \tag{3-45}$$

for all i, $0 \leq i < n$, and all $\underline{u}^i \in \mathcal{U}^i$. Γ and Δ are positive constants determined by the design. Then

$$\lambda^{-S(\underline{u}^n)} = \lambda^{-\Sigma_{i=1}^n s(u_i|\underline{u}^{i-1})} \geq \Gamma^{-n}\Delta^{-n(c-\frac{1}{n}\Sigma_{i=1}^n u_i)} P(\underline{u}^n)$$

Now define the <u>average symbol</u> $\bar{u}$ by

$$\bar{u} \triangleq \frac{1}{n}\sum_{\underline{u}^n\in\mathcal{U}^n} P(\underline{u}^n)\cdot(\sum_{i=1}^n u_i). \tag{3-46}$$

With (3-44) we obtain:

$$r(n) \leq \log_d \Gamma + (c-\bar{u})\cdot\log_d \Delta + O(1/n) \tag{3-47}$$

For an arithmetic code, n can be made very large. So we might ignore the term $O(1/n)$. Γ and Δ will be expressed in the table parameters α and β and thus we obtain a source-independent upperbound. We give two examples.

Example 3: (Redundancy bound for Design 1). From (3-37) we find:

$$\lambda^{-s(u|\underline{u}^i)} > \lambda^{-1} \cdot \alpha\beta^{-1} \cdot P(u|\underline{u}^i) \tag{3-48}$$

so $t(u) = \frac{\lambda\beta}{\alpha}$ and the redundancy is upperbounded as:

$$r(n) < \log_d \frac{\lambda\beta}{\alpha} + O(1/n) \tag{3-49}$$

Example 4: (Redundancy bound for Design 2). From (3-38) we obtain:

$$t(u_i) = \lambda \cdot (\frac{\lambda\beta}{\alpha})^{c-u_i} \tag{3-50}$$

which gives:

$$r(n) < (c - \bar{u}) \cdot \log_d \frac{\lambda\beta}{\alpha} + \log_d \lambda + O(1/n) \tag{3-51}$$

This is in the order of c times worse than (3-49).

For the global methods we can also derive similar lowerbounds. From the fact that the code interval $J(\underline{x}^m)$ is included in the source interval $I(\underline{u}^n)$ we find that:

$$L(\underline{u}^n) \geq -\log_d A[S(\underline{u}^n)]$$

or

$$L(\underline{u}^n) \geq \frac{S(\underline{u}^n)}{N} - \log_d \beta.$$

As in (3-42) this results in:

$$\mathbf{E}_{\underline{U}^n}\{L(\underline{U}^n)\} \geq H(P(\underline{U}^n)) + \tilde{D}(P(\underline{U}^n)\|\lambda^{-S(\underline{U}^n)}) + O(1)$$

and

$$r(n) \geq \frac{1}{n}\tilde{D}(P(\underline{U}^n)\|\lambda^{-S(\underline{U}^n)}) + O(1/n). \qquad (3\text{-}52)$$

Example 5: (Lowerbound for global P-designs). First, using the log-sum inequality [23], which states that $\Sigma_i\ a_i \cdot \log\frac{a_i}{b_i} \geq (\Sigma_i\ a_i)\cdot\log\frac{\Sigma_i\ a_i}{\Sigma_i\ b_i}$, where all a_i's and b_i's are non-negative numbers, we obtain:

$$\begin{aligned}
&\tilde{D}(P(\underline{U}^n)\|\lambda^{-S(\underline{U}^n)}) \\
&\quad \geq -\log_d\Big(\sum_{\underline{u}^n\in\mathcal{U}^n}\lambda^{-S(\underline{u}^n)}\Big) \\
&\quad = -\log_d\Big(\sum_{\underline{u}^n\in\mathcal{U}^n}\prod_{i=1}^{n}\lambda^{-s(u_i|\underline{u}^{i-1})}\Big) \\
&\quad = -\log_d\Big(\sum_{u_1\in\mathcal{U}}\lambda^{-s(u_1|\underline{u}^0)}\cdot\sum_{u_2\in\mathcal{U}}\lambda^{-s(u_2|u_1)}\cdot\ \ldots \\
&\qquad\qquad \cdot\sum_{u_n\in\mathcal{U}}\lambda^{-s(u_n|\underline{u}^{n-1})}\Big)
\end{aligned} \qquad (3\text{-}53)$$

Since any global P-design satisfies the global P-test, Lemma 3-3, we substitute (3-35) in (3-53) and obtain the following:

$$\tilde{D}(P(\underline{U}^n)\|\lambda^{-S(\underline{U}^n)}) \geq n\cdot\log_d\frac{\beta}{\alpha}.$$

And with (3-52) we find the lowerbound to the redundancy:

$$r(n) \geq \log_d \frac{\beta}{\alpha} + O(1/n). \tag{3-54}$$

Example 6: (Lowerbound for global Q-designs). The global Q-test, Lemma 3-4, holds for a global Q-design so we use (3-36) in the following:

$$\begin{aligned}
&\tilde{D}(P(\underline{U}^n)\|\lambda^{-S(\underline{U}^n)}) \\
&= \sum_{\underline{u}^n \in \mathcal{U}^n} P(\underline{u}^n) \cdot \log_d \frac{P(\underline{u}^n)}{\prod_{i=1}^n \lambda^{-s(u_i|\underline{u}^{i-1})}} \\
&= \sum_{\underline{u}^n \in \mathcal{U}^n} P(\underline{u}^n) \cdot \log_d \prod_{i=1}^n \left(\frac{\lambda\beta}{\alpha}\right)^{c-u_i} + \\
&\qquad \sum_{\underline{u}^n \in \mathcal{U}^n} P(\underline{u}^n) \cdot \log_d \frac{P(\underline{u}^n)}{\prod_{i=1}^n \left(\frac{\lambda\beta}{\alpha}\right)^{c-u_i} \cdot \lambda^{-s(u_i|\underline{u}^{i-1})}} \\
&= n \cdot (c - \bar{u}) \cdot \log_d \frac{\lambda\beta}{\alpha} + \\
&\qquad \sum_{\underline{u}^n \in \mathcal{U}^n} P(\underline{u}^n) \cdot \log_d \frac{P(\underline{u}^n)}{\prod_{i=1}^n \left(\frac{\lambda\beta}{\alpha}\right)^{c-u_i} \cdot \lambda^{-s(u_i|\underline{u}^{i-1})}} \\
&\overset{(a)}{\geq} n \cdot (c - \bar{u}) \cdot \log_d \frac{\lambda\beta}{\alpha} - \log_d \left(\sum_{\underline{u}^n \in \mathcal{U}^n} \prod_{i=1}^n \left(\frac{\lambda\beta}{\alpha}\right)^{c-u_i} \cdot \lambda^{-s(u_i|\underline{u}^{i-1})} \right) \\
&\overset{(b)}{\geq} n \cdot (c - \bar{u}) \cdot \log_d \frac{\lambda\beta}{\alpha}.
\end{aligned}$$

In (a) we use the log-sum inequality and in (b) we use (3-36).

Again with (3-52) we obtain the lowerbound to the global Q-design redundancy:

$$r(n) \leq (c - \bar{u}) \cdot \log_d \frac{\lambda\beta}{\alpha} + O(1/n) \tag{3-55}$$

We want to stress the point that the lowerbounds only hold for global designs satisfying the global P- or Q-test.

If we compare the upperbounds (3-49) and (3-51) with the corresponding lowerbounds (3-54) and (3-55), we observe a difference between the bounds of about $\log_d \lambda = 1/N$. The difference between the upper- and lowerbound becomes smaller as the size of the table increases. This, however, does not imply that the bounds become tight. We will demonstrate the fact that the bounds are not tight with the table defined as in Example 2.

Assume $d = 2$, i.e. a binary code alphabet, and let the size of the table $\tilde{A}[\cdot]$ be C binary digits. We are interested in the minimum of $\log_2 \frac{\lambda\beta}{\alpha}$ under the restriction that $N \cdot k = C$. Allowing non-integer values for N and k, this optimum is achieved by $k_{opt} = 1 + \log_2 C-1$. Denote by r_P^+ the upperbound value (3-49) at the optimum for global P-designs, and by r_P^- the lowerbound value (3-54), still with non-integer k_{opt}. For the relative difference

$$\delta_P \triangleq \frac{r_P^+ - r_P^-}{r_P^-}$$

we find:

$$\delta_P = \frac{1 + \log_2 C-1}{C \cdot \log_2 1+\frac{2}{C-1}}$$

So, for $C \to \infty$, $\delta_P \approx \frac{\ln C}{2} \to \infty$.

The global Q-design upperbound achieves its optimum close to k_{opt} if the source alphabet size c becomes large. See section 3.11. Define r_Q^+ as the value of (3-51) at k_{opt} and r_Q^- as the corresponding value of (3-55). The relative difference δ_Q is given as:

$$\delta_Q = \frac{r_Q^+ - r_Q^-}{r_Q^-} = \frac{\log_2 \lambda}{(c - \bar{u})(\log_2 \lambda + \log_2 \beta)}$$

$$= \frac{1}{(c - \bar{u})(1 + 1/\delta_P)}.$$

So, for $C \to \infty$, we find that $\delta_Q \approx \frac{1}{c-\bar{u}}$.

In both cases, the relative difference will not approach zero demonstrating the fact that the bounds are not tight in the case where we select the table parameters N and k such that the upperbound is minimized.

From these bounds we conclude that the P-method redundancy is insensitive to the source alphabet size and that the redundancy for the Q-method increases linearly with c. This is not unexpected if we consider the way we let the two methods stack their source intervals. In the P-method all c^n intervals fit on top of each other, so this leaves only one gap at the top of [0, 1) resulting in an inefficiency of about $\log_d \frac{\lambda\beta}{\alpha}$. In the Q-method we approximate the augends by one table value, $A[S(\underline{u})+T(u|\underline{u})]$, and $T(u|\underline{u})$ is selected such that the u subintervals $I(\underline{u}v)$, $0 \leq v < u$, always fit below $I(\underline{u}u)$. See Figure 3-4. From the Global Q-test (3-36) and the bounds (3-51) and (3-55) we are led to the conclusion that the gap for symbol u, on the average, introduces an inefficiency of about $(c - u) \cdot \log_d \frac{\lambda\beta}{\alpha}$. With $\log_d \lambda = \frac{1}{N}$, we see that every imprecision in the multiplication $A[i] * A[j]$, as mentioned in section 3.5, is accounted for. We can 'equate' the redundancy caused by approximating $\log_d P(\underline{u})$ with $\frac{1}{N}$, and the redundancy $\log_d \frac{\beta}{\alpha}$ with the imprecise multiplication and the provisions we have to take to ensure decodability. The experiments described in section 3.11 support this conclusion.

The local designs perform at least as well as the global designs they are based on. These designs try to close the gaps left by the global de-

signs. Generally, it will be impossible to close the gaps completely, but because the global designs consider the worst case situation we can expect a significant reduction in the redundancy. This is also confirmed by the experiments in section 3.11.

3.9. Fast designs.

The codes resulting from the global designs of section 3.7 perform very well with respect to the code rate but the designs require complex and precise arithmetic.

In this section we describe four designs that make use of the exponential table $A[\cdot]$. The only operations required are searching in this table and simple additions. The price that must be paid for the reduction in complexity is an increased redundancy.

We will first introduce and analyse a global design for the P-method.

Design 5: Set

$$s_0 = \min\{s \mid A[s] \leq \frac{\alpha^3}{\beta}\} \tag{3-56}$$

$$s_1(u|\underline{u}) = \min\{s \mid A[s] \leq P(u|\underline{u})\} \tag{3-57}$$

and

$$s(u|\underline{u}) = s_0 + s_1(u|\underline{u}) \tag{3-58}$$

In this design s_0 can be precomputed.

First we show that this design results in decodable codes. From (3-22), (3-56), (3-57), and (3-58) we find:

$$\frac{\alpha^3}{\lambda^2\beta^3} \cdot P(u|\underline{u}) \leq \lambda^{-s(u|\underline{u})} \leq \frac{\alpha}{\beta} \cdot P(u|\underline{u}). \tag{3-59}$$

From the righthand part of (3-59) it follows that the global P-test, Lemma 3-3, is satisfied, and with the lefthand part and (3-45) - (3-47) we find the following upperbound to the code redundancy:

$$r(n) \leq 3 \cdot \log_d \frac{\lambda\beta}{\alpha} - \log_d \lambda + O(1/n) \tag{3-60}$$

So the price here is a threefold increase in redundancy as compared to Design 1.

The global Q-design is as follows:

Design 6: Set

$$s_0(u) = \min\{s \,|\, A[s] \leq \alpha^2 (\frac{\lambda^2\beta^3}{\alpha^3})^{u-c}\} \tag{3-61}$$

$$s_1(u|\underline{u}) = \min\{s \,|\, A[s] \leq P(u|\underline{u})\} \tag{3-62}$$

$$T_0(u) = s_0(u) \tag{3-63}$$

$$T_1(u|\underline{u}) = \min\{t \,|\, A[t] \leq Q(u|\underline{u})\} \tag{3-64}$$

and

$$s(u|\underline{u}) = s_0(u) + s_1(u|\underline{u}) \tag{3-65}$$

$$T(u|\underline{u}) = T_0(u) + T_1(u|\underline{u}) \tag{3-66}$$

Again, the c constants $s_0(u)$ can be precomputed. The decodability of this design is proved in Appendix XI.

With (3-22) and (3-61) - (3-66) we find:

$$\lambda^{-s(u|\underline{u})} \geq \frac{\alpha^2}{\lambda^2\beta^2} \cdot (\frac{\lambda^2\beta^3}{\alpha^3})^{u-c} \cdot P(u|\underline{u}) \tag{3-67}$$

resulting in the upperbound:

$$r(n) \leq 2 \cdot \log_d \frac{\lambda\beta}{\alpha} + (c - \bar{u})\{3 \cdot \log_d \frac{\lambda\beta}{\alpha} - \log_d \lambda\} + O(1/n) \quad (3\text{-}68)$$

Again, as compared to Design 2, the redundancy is about thrice as large.

As with the Designs 1 and 2, we can formulate local versions of the fast designs. We introduce:

Design 7: (Local P-design based on Design 5). First evaluate (3-56) - (3-58) for all $u \in \mathcal{U}$. Then, as in Design 3, decrease the stepvalues until the local P-test would be violated by any further decrease.

Design 8: (Local Q-design based on Design 6). Let u_{i+1} be the symbol to be encoded. Compute $T(u_{i+1} + 1|\underline{u}^i)$ and $T(u_{i+1}|\underline{u}^i)$ using (3-61), (3-63), (3-64), and (3-66) and find the smallest $s(u_{i+1}|\underline{u}^i)$ under the restriction of the local Q-test at position $S(\underline{u}^i)$. See Design 4.

3.10. Complexity aspects.

A definite scheme e.g. the Huffman code, needs a memory to store the codewords. Encoding and decoding is performed by indexing in, resp. searching through, the codebook. Two types of complexity result from this observation; the storage requirements and the amount of work needed in the encoding and decoding process. We will consider these complexities for the P- and Q-methods, and then compare it with the complexity of a Huffman code.

First, consider the storage requirements. As said in a previous section assume that each table entry $A[i]$ is expressable in k d-ary digits. Then, we need $N \cdot k$ digits to store this table. Because the parameters α, β, and λ are functions of N and k, so are the redundancy bounds (3-49), (3-51), (3-60), and (3-68), and we might solve for the minimum redundancy bound given a constraint on $N \cdot k$. An example thereof was already discussed in section 3.8.

Now, we turn to the amount of work needed to encode and decode a source symbol. We will illustrate the complexity using the same eight designs as given in the previous sections.

Let $u \in \mathcal{U}$ be the symbol to be encoded and decoded, and $\underline{u}$ is the string of source symbols preceding u.

Design 1. Encoding: (3-37) must be computed for all values $v \leq u$. Then, the P-method (3-23) requires u table references and additions. (3-24) must be computed once. So, the encoding time is proportional to $\bar{u}$, (3-46), and thus related to the source alphabet size.

Decoding: Assume we received enough code symbols to decode u. See section 3.3. Successively we must compute the values $B(\underline{u}0)$, $B(\underline{u}1)$, ... , $B(\underline{u}u)$. This requires $u + 1$ evaluations of (3-37) and u references and additions in (3-23). Also $u + 1$ comparisons are needed. (3-24) is computed once. So the decoding time is also proportional to $\bar{u}$, although the constant of proportionality is larger.

Design 2. Encoding: Compute (3-38) and (3-39) for u only. The Q-method (3-25) requires one table reference and one addition. (3-26) is computed once also. The encoding time is independent of the cardinality of the source alphabet.

Decoding: (3-39) and (3-25) must be computed for all $v \leq u$ and every time a comparison is made. (3-38) and (3-26) are computed only once. The decoding time is, as in Design 1, proportional to $\bar{u}$. However, a better performance is obtained using a binary search resulting in at most $\log_2 c$ evaluations and comparisons. In some cases, e.g. when the source is memoryless and the probabilities are ordered, this search could even be optimized to, on the average, $H(P(U))$ evaluations etc. See Massey [24].

Design 3. Encoding: Compute all global stepvalues. This requires c evaluations of (3-37). Now for every successive symbol $v \leq u$ compute the sum

$W = \sum_{w \neq v} A[S(\underline{u}) + s(w|\underline{u})]$, and then find the smallest s such that $A[S(\underline{u}) + s] \leq A[S(\underline{u})] - W$. Replace $s(v|\underline{u})$ by s and repeat the above for the next symbol v. The encoding time is $K_1 \cdot c + \bar{u} \cdot (K_2 \cdot (c-1) + K_3 \cdot \log_2 N)$, where K_1 is the time needed to evaluate (3-37) once, and $K_2 \cdot (c-1)$ the time needed to compute the sum W. $K_3 \cdot \log_2 N$ denotes the time spend in obtaining a minimal stepvalue s, as we show in the discussion of Design 4 below. If we accept that $\bar{u}$ is proportional to c, the encoding time is proportional to c^2.

Decoding: As with the discussion of Design 1, the encoding and decoding processes are similar, resulting in a decoding time proportional to c^2.

Design 4. Encoding: Compute (3-39) for the symbols u and u+1. Then search through the table for the smallest index satisfying the local Q-test. This search is performed in two steps; first determine the order of the difference of the two augends, i.e. find an integer ℓ such that $d^{\ell} \cdot (T(u+1|\underline{u}) - T(u|\underline{u})) \in (d^{-1}, 1]$. Then search for the smallest index $i \in \{0, 1, \ldots, N-1\}$ such that $A[i] \leq d^{\ell} \cdot (T(u+1|\underline{u}) - T(u|\underline{u}))$. Using a binary search we need at most $\log_2 N$ tries. So the search complexity is $\log_2 N$. Then compute the stepvalue $s(u|\underline{u}) = \ell \cdot N + i$, and now find the new interval using (3-25) and (3-26). Although the amount of work has incresed compared to Design 2 it is still a constant.

Decoding: the decoding is similar to the decoding of Design 2. Only, per comparison we must perform all the computations we need to encode a symbol, i.e. compute (3-39) twice, search through the table etc. So the proportionality constant increases, but the decoding time is still proportional to $\log_2 c$ or $H(P(U))$.

Design 5. Encoding: As in Design 1, compute (3-57) for all $v \leq u$. This is done by searching through the table as described above. Then, add the precomputed constant s_0, (3-56). The P-method requires u additions and table references for (3-23) and (3-24) is computed once. So the encoding

time is proportional to $\bar{u}$, but compared to Design 1, the constant of proportionality is much smaller.

Decoding: The decoding is also similar to the decoding of Design 1. However, the complex computations in (3-37) are again replaced by the searches in (3-57) and the additions in (3-58). So decoding too is faster than in Design 1.

Design 6. Here the encoding and decoding are similar to those for Design 2, again replacing the complex computations (3-38) and (3-39) by searches etc.

The same holds for the local designs. Design 7 is similar to Design 3 and Design 8 looks like Design 4, again replacing the computations by searches through the table.

Summarizing we may say that the the amount of work for the global P-designs is proportional to $\bar{u}$ or, in a worst-case analysis, proportional to the cardinality c. The local P-Designs are proportional to c^2. In the global and local Q-designs the encoding times are independent of c and the decoding times are proportional to $\log_2 c$. The designs of section 3.9 are faster due to the replacement of the, rather precise, computation of logarithms by a search in $A[\cdot]$.

We will compare this with the complexity of a Huffman code. There are $M = c^n$ different source messages of length n. A codebook implementation requires a table of M entries, each able to store a codeword. More efficient storage is achieved using a tree structure. Then, the storage complexity is of $O(M)$.

Another problem is the design of a Huffman code given a probability vector $P(\underline{u}^n)$. Van Voorhis, [25], described an algorithm that is $O(M^2)$ in time and space (storage). However, it only applies to binary codes. The famous Hu-Tucker algorithm, [26], generates binary Huffman codes in $O(M^2)$ time and $O(M)$ space. The Garsia-Wachs implementation of the Hu-Tucker algo-

rithm [27] reduces the time complexity to $O(M \cdot \log_2 M)$ while retaining a $O(M)$ complexity in space. Van Leeuwen, [28], shows that $O(M \cdot \log_2 M)$ in time is optimal for the design of binary Huffman codes. Now remember that $M = c^n$; thus the best Huffman design has a time complexity of $O(n \cdot c^n)$ and $O(c^n)$ in space. So we see that the complexities of a Huffman code are at least exponential in the source string length, thus prohibiting the use of large source strings and the resulting low redundancies.

In order to illustrate the latter in some detail we first remark that from the source coding theorem, [1], we can upperbound the redundancy of a Huffman code by $1/n$, where n again denotes the source block length. Gallager in [29] improved upon this resulting in the bound for a binary Huffman code:

$$r \leq P_1 + 0.0861, \quad 0 \leq P_1 < 0.5$$

$$r \leq 2 - h(P_1) - P_1, \quad 0.5 \leq P_1 \leq 1$$

where P_1 is the largest probability in the vector $P(\underline{u}^n)$ and $h(x) \triangleq -x \cdot \log_2 x - (1-x) \cdot \log_2 (1-x)$, the binary entropy function. For a d-ary code he obtained:

$$r \leq \sigma_d + P_1 \cdot \frac{d}{\ln d}$$

with $\sigma_d = \log_d d - 1 + \log_d(\log_d e) - \log_d e + \frac{1}{d-1}$. Unfortunately $\sigma_d \to \infty$ as d gets large, but not too fast. For example $\sigma_3 = 0.135$, $\sigma_5 = 0.194$, and $\sigma_{10} = 0.269$.

Johnsen [30], and Capocelli et al. [31], improved the bound for a binary code in the case $2/9 \leq P_1 \leq 0.5$. Still these bounds are inferior to $1/n$

as n becomes large. And because the redundancy of the P- and Q-method schemes are functions of the table size only, and rather small for acceptable table sizes, these schemes compare favourably with the Huffman code designs.

3.11. Some numerical examples.

In this section we discuss some simulations done with the eight designs. The binary codes were designed using a table $\tilde{A}$ as given in Example 2, so $\tilde{A}[i] \triangleq d^{-k} \cdot \lceil d^{k} \cdot d^{-i/N} \rceil$, $0 \leq i < N$; k and N are positive integers.

The storage complexity for these codes is $N \cdot k$ binary digits. We designed tables with $N \cdot k \cong 1000$ and $N \cdot k \cong 10000$, and k several values in the range from 10 to 20. We selected values that should be representative for the algorithms. For instance, the minimum of $\log \frac{\lambda\beta}{\alpha}$ under the restriction that $N \cdot k = C$ occurs at $k_{opt} = 1 + \log_d C - 1$. For $C = 1000$ this gives $k_{opt} \cong 11$ and if $C = 10000$ then $k_{opt} \cong 14$. Since the local designs generally performed best with rather precise tables, i.e. large k, we also selected values in that range.

We evaluated the performance of the eight designs using sources with cardinalities 2, 8, and 16. For each of these cardinalities we selected two memoryless sources, one with a high entropy and one with a low entropy. Every source was considered twice, once with ascending probabilities (high $\bar{u}$) and once with decending probabilities (low $\bar{u}$). The detailed results are stated in Appendix XII. The following observations can be made from these results:

- The global redundancy bounds appear to be tight in the sense that for some combinations of table parameters and symbol probabilities the resulting redundancy is close to either the lower- or the upperbound.

- From the Tables 5 and 9, Appendix XII, we see that the global P-designs are insensitive to the alphabet size c and, by definition they are independent of the ordering of the probabilities.
- The Tables 6 and 10 indicate that the global Q-designs behave as expected, i.e. proportional to $(c - \bar{u})$.
- The actual improvements of the local designs over their global counterparts vary a lot. In some cases the improvement is only a few percent, in other cases the redundancy decreases by a factor of ten or more. Note that in some cases the local P-designs 3 and 7 and in most cases the local Q-designs 4 and 8 achieve a lower redundancy than the lower-bounds in the Tables 1 and 2; so they are essentially better than the global methods, with respect to the redundancy.
- If we compare the global Designs 1 and 2 with their fast counterparts 5 and 6 we find, on the average, a redundancy increase by a factor 2 to 3 as expected, although the actual values vary more. The variation in redundancies between the local designs using logarithms and those using the table is even higher, although here too we find an average factor of 2 to 3.
- The local P-designs achieve the smallest redundancy for a given table compared to all other designs but do so at the cost of a very high computational complexity.
- The fast local Q-design 8 appears to be the best compromise, especially if the probabilities are ordered to achieve a maximal $\bar{u}$. However, as we will see in the next section, we cannot sort the probabilities without paying a penalty in increased complexity.

For a comparison of these designs with other arithmetic codes we refer to section 3.13.

3.12. Implementation details.

In section 3.3 we discussed the partial encoding and decoding of arithmetic codes in a rather general setting. Section 3.4 then introduced a carry-blocking technique and briefly indicated how to implement the encoder. See Figure 3-3. In this section we describe the implementation of the encoder and decoder for both methods.

To implement the equations (3-23) and (3-24) respectively (3-25) and (3-26) we need an accumulator B that contains the relevant part of the codestring $B(\underline{u}^i)$. Again, as in Example 2, assume that each table entry $\tilde{A}[\cdot]$ is described in k digits, then the size of the accumulater B is at least k symbols. However, the smaller the symbol probability is, the larger is the corresponding stepvalue and the augend will contain leading zeroes, relative to the left end of the accumulator. We allow for l extra symbol positions in B, resulting in a total accumulator length of $k + l$ symbols. Later on in this section, we will return to the importance of the exact value of l, for now we just assume that l is large enough to accomodate all occurring additions.

In section 3.4 we loosely defined the augend position and indicated in Figure 3-3 that we only had to deal with a few symbols around that position. Now we are able to define this in all necessary detail. Observe the augends in (3-23) and (3-25). We see that we want to add table entries with indices larger than $S(\underline{u})$. Also, by the definition of the source intervals $I(\underline{u})$, (3-28), we know the maximal value of the augend, namely $A[S(\underline{u})]$. Now the number of leading zeroes in $A[S(\underline{u})]$ is $\lfloor S(\underline{u})/N \rfloor$, and except for a possible carry the first $\lfloor S(\underline{u})/N \rfloor$ symbols in the codestring will not change anymore. So we can define the augend position I_A as $\lfloor S(\underline{u})/N \rfloor$, where we implicitely assume the dependence of I_A on the source string u. This defini-

tion would suffice if we did not have to provide for the blocking of an occurring carry. Now carry blocking, as discussed in section 3.4, repeatedly divides the interval by d without changing its lowerbound value. This results in an increase over $\lfloor S(\underline{u})/N \rfloor$ by one for each carry-blocking shift that occurs, so:

$$I_A \triangleq \lfloor S(\underline{u})/N \rfloor + N_C(\underline{u}) \tag{3-69}$$

where $N_C(\underline{u})$ denotes the number of carry-blocking shifts that occurred during the encoding of $\underline{u}$.

The implementation of the encoder is depicted in Figure 3-9. The

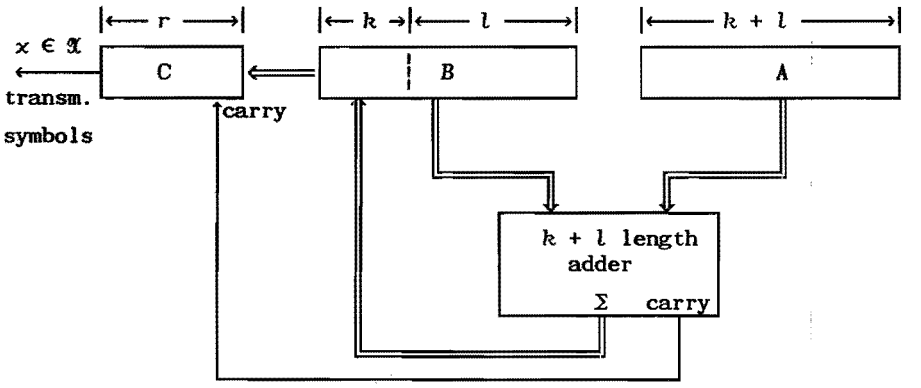

Figure 3-9. The implementation of the encoder.

register C is carry blocking register. B is the accumulator and A contains the table value, relative to I_A, that must be added to the string. So A contains $A[S(\underline{u}) + N \cdot N_C(\underline{u}) - N \cdot I_A + s(u|\underline{u})]$ if we must add $A[S(\underline{u}) + s(u|\underline{u})]$.

The new augend position is computed by (3-69) and the resulting shifts, dictated by the difference between the new and the old position, are per-

formed. If, after the process of encoding the source symbol the C register contains a string of r d-1 symbols, then both the C and the B registers are shifted to the left until a symbol other than d-1 is shifted into C. Every shift increases the shiftcount $N_C(\underline{u})$ by one.

When all the source symbols have been encoded the C and B registers still contain a part of the codestring. The final interval length is given by $A[S(\underline{u}^n) + N \cdot N_C(\underline{u}^n)] \geq d^{-(I_A+1)}$. With Theorem 3-1 we know that it suffices to take all the code symbols in C plus the leftmost two in B after rounding B up. Alternatively, we could transmit all $k+l$ symbols in B, increasing the codestring length by $k+l-2$ digits, which should be negligible compared to the source string length n.

The decoder is depicted in Figure 3-10. It consists of an encoder C', B', and A', that mimicks the real encoder and a comparator that compares this string with the received string in the receiver registers RC and RB. This decoder implements the partial decoding scheme as outlined in the sections 3.3 and 3.10. Further details should be obvious. We end with the remark that every shift dictated by either the carry-blocking mechanism or the augend position update also shifts the RC and RB receiver registers, and loads the RB registers with newly received code digits.

We now give an algorithmic description of the encoder and decoder operations for both methods. In the Q-method decoder we shall implement the binary search technique.

Where applicable, we use the following variables:

U: for encoders: the symbol to be encoded. (input).
for decoders: the decoded symbol. (output).

S[•]: the stepvalue array $s(\bullet|\underline{u})$, generated by a design. (input).

T[•]: the stepsums $T(\bullet|\underline{u})$, generated by a Q-design. (input).

A[•]: the exponential table with an index range 0 ... N-1.

N: the table length.

C: the carry-blocking register.

B: the accumulator.

FS: the retained fraction of the local position.

$FS = S(\underline{u}) + N \cdot N_C(\underline{u}) - N \cdot I_A$. ($= S(\underline{u}) \bmod N$)

RC: receiver carry part register

RB: receiver accumulator part register.

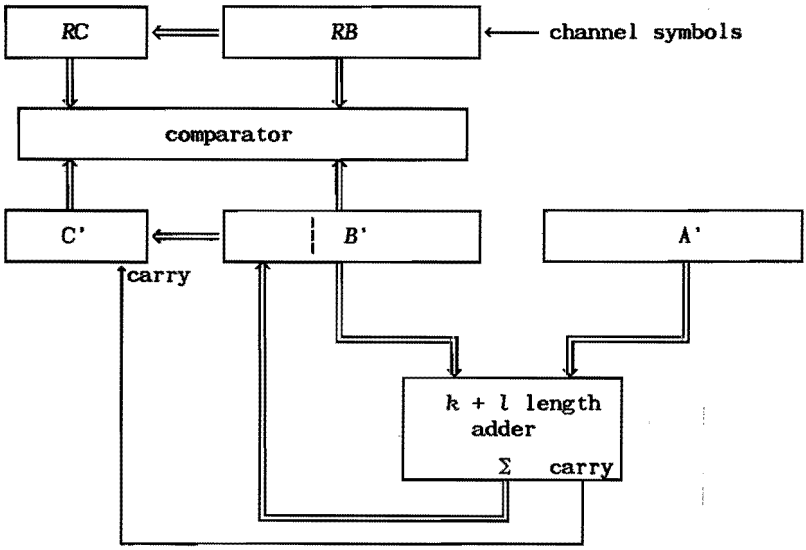

Figure 3-10. The decoder implementation.

Remark: The arithmetic in the algorithms is done in number base d. (d is the code alphabet size).

P-method encoder:

```
begin { first add the augend parts to the codestring }
    for v:=0 to U-1   { if U=0 then do not enter this loop }
    do begin   { perform the addition of A[S(u) + s(v|u)] }
               { note that if P(u|u) = 0 then "s(u|u) = ∞"
                 implying that nothing is added to C&B }
        sft := (FS + S[v]) div N;   { this is the number of leading
                                      zeroes w.r.t the augend position. }
        pos := (FS + S[v]) mod N;   { index in Ã, sft ≤ l ! }
        C&B := C&B + Shiftright(A[pos], sft);   { perform the
                  'shifted' addition and process the possible carry }
      end;
    { now we continue with the computation of the new position }
    sft := (FS + S[U]) div N;   { augend position update # shifts }
    FS  := (FS + S[U]) mod N;   { update the retained fraction }
    Shiftleft( C&B, sft);   { shift the C and B registers to the left
                              and load B with zeroes. }
    { now perform the carry-blocking operation(s) }
    while C = ( d-1, d-1, ... , d-1)   { fault condition ! }
    do Shiftleft( C&B, 1);
end;   { the symbol U is encoded into the codestring }
```

Q-method encoder:

```
begin { first add the augend to the codestring if U > 0 }
    if U > 0
    then begin { compute and add augend }
        sft := (FS + T[U]) div N;   { this is the number of leading
                                      zeroes w.r.t the augend position. }
```

```
            pos := (FS + T[U]) mod N;    { index in Ã, sft ≤ l ! }
            C&B := C&B + Shiftright(A[pos], sft);    { perform the
                        'shifted' addition and process the possible carry }
            end;
        { now we continue with the computation of the new position }
        sft := (FS + S[U]) div N;    { augend position update # shifts }
        FS  := (FS + S[U]) mod N;    { update the retained fraction }
        Shiftleft( C&B, sft);    { shift the C and B registers to the left
                                   and load B with zeroes. }
        { now perform the carry-blocking operation(s) }
        while C = ( d-1, d-1, ... , d-1)    { fault condition ! }
        do Shiftleft( C&B, 1);
    end;    { the symbol U is encoded into the codestring }
```

P-method decoder:

```
    begin    { find the next symbol by sequential search }
        U := 0;
        { simulate the encoder until the reconstructed register contents
          are larger than the received codestring part. Again we have no
          problem with possible "zero probability" symbols. }
      loop:  sft := (FS + S[U]) div N;    { determine augend part }
             pos := (FS + S[U]) mod N;
        if (C&B + Shiftright( A[pos], sft)) > (RC&RB)
        then goto decoded;
        C&B := C&B + Shiftright( A[pos], sft);
        U := U + 1;
        goto loop;
```

```
    decoded:  { sft and pos are already computed and U is the correct
                symbol, so }
      FS := pos;   { the new augend position. }
      Shiftleft( C&B, sft);
      Shiftleft( RC&RB, sft);   { also shift the next digits from the
                                  channel into RB. }
      { now again the carry blocking }
      while C = ( d-1, d-1, ... , d-1)
      do begin Shiftleft( C&B, 1);
               Shiftleft( RC&RB, 1)  { don't forget the channel digits }
         end
  end;
```

Q-method decoder:

```
  begin   { this decoder implements a binary search }
      high := c;   { c is the source alphabet cardinality }
      low  := 0;
      { now and later we assure that:
        B(u, low) ≤ RC&RB < B(u, high) }
      while high - low > 1
      do begin U := (high + low) div 2;   { pivot point }
               { here we must ensure that P(U|u) > 0 }
           sft := (FS + T[U]) div N;
           pos := (FS + T[U]) mod N;
           if (C&B + Shiftright( A[pos], sft)) ≤ (RC&RB)
           then low := U
           else high:= U
         end;   { here low contains the decoded symbol }
```

```
        U := low;
        sft := (FS + T[U]) div N;
        pos := (FS + T[U]) mod N;
        C&B := C&B + Shiftright( A[pos], sft);   { update C and B for next
                                                   symbol }
        { again, compute next augend position and process carry }
        sft := (FS + S[U]) div N;
        FS  := (FS + S[U]) mod N;   { the new augend position. }
        Shiftleft( C&B, sft);
        Shiftleft( RC&RB, sft);   { also shift the next digits from the
                                    channel into RB. }
        { the carry blocking }
        while C = ( d-1, d-1, ... , d-1)
        do begin Shiftleft( C&B, 1);
                 Shiftleft( RC&RB, 1)  { don't forget the channel digits }
           end
    end;
```

Remark: The 'mod' and 'div' operations in the algorithms can be replaced by simple shifts if the table length N is a power of d.

We return to the relation between the extra register length l required by the algorithm and the stepvalues. The number of shifts in the summations in the algorithms is upperbounded by l or:

$$(FS + S[v]) \text{ div } N \leq l, \quad (FS + T[U]) \text{ div } N \leq l.$$

And with $FS < N$ we find

$$S[v] \leq l \cdot N, \quad T[U] \leq l \cdot N$$

or

$$\lambda^{-s(u|\underline{u})} \geq d^{-l}, \quad \lambda^{-T(u|\underline{u})} \geq d^{-l} \tag{3-70}$$

We are interested in the relation between l and $P(u|\underline{u})$, $Q(u|\underline{u})$. Thus we have to consider how the different designs assign stepvalues and stepsums to symbols. Formula (3-45) enables us to a worst-case analysis for the four global designs. Because the local designs have stepvalues and stepsums not larger than those of the corresponding global designs, they require no separate analysis.

First we study the P-method Designs 1 and 5. Observe that if $P(u|\underline{u}) \geq P(v|\underline{u})$ then $s(u|\underline{u}) \leq s(v|\underline{u})$, so the largest stepvalue occurs for $P_{min} \triangleq \min\{P(u|\underline{u}) | u \in \mathcal{U}, \underline{u} \in \mathcal{U}^*\}$. With (3-48) we find for Design 1:

$$P_{min} \geq \frac{\lambda\beta}{\alpha} d^{-l}$$

and with (3-59) the same for Design 5 becomes:

$$P_{min} \geq \frac{\lambda^2\beta^3}{\alpha^3} d^{-l}$$

In a realistic situation we must assume l to be fixed by e.g. the hardware designer. So we must ensure that $P_{min} \geq \epsilon \cdot d^{-l}$, ($\epsilon = \epsilon_1 \triangleq \frac{\lambda\beta}{\alpha}$, or $\epsilon = \epsilon_5 \triangleq \frac{\lambda^2\beta^3}{\alpha^3}$). We shall examine the following approach, <u>probability clipping</u>:

Define the clipped probability $\tilde{P}(u|\underline{u})$ as:

$$\tilde{P}(u|\underline{u}) = \frac{P(u|\underline{u}) + \delta}{1 + c \cdot \delta}$$

for some positive constant δ.

We observe that

$$\tilde{P}(u|\underline{u}) \geq \frac{\delta}{1 + c \cdot \delta},$$

and so we shall use $\tilde{P}(u|\underline{u})$ in our designs.

From the lowerbound on P_{min} we solve:

$$\delta = \frac{\epsilon \cdot d^{-l}}{1 - c\epsilon \cdot d^{-l}}.$$

As in (3-45) - (3-47) we bound:

$$t(u) \cdot \lambda^{-s(u|\underline{u})} \geq \tilde{P}(u|\underline{u}) = \frac{P(u|\underline{u}) + \delta}{1 + c \cdot \delta}$$

and so:

$$\lambda^{-S(\underline{u}^n)} \geq \left[\frac{1}{(1+c \cdot \delta) \cdot \Gamma \cdot \Delta^{c - \frac{1}{n}\sum_{i=1}^{n} u_i}} \right]^n \cdot [\delta^n + P(\underline{u}^n)]$$

This, with (3-42) - (3-44) results in, c.f. (3-47):

$$r(n) \leq \log_d(1+c \cdot \delta) + \log_d \Gamma + (c - \bar{u}) \cdot \log_d \Delta + O(\frac{1}{n})$$

Thus, clipping costs us circa $c \cdot \epsilon \cdot d^{-l}$ bits per symbol in redundancy. Note that the clipping method described here is not optimal, but it suffices to illustrate, with a rather simple proof, the exponential dependency of the redundancy on the clipping parameter l.

Now we study the relation between l and the source probabilities for

the Q-method Designs 2 and 6. Here the stepsums T are required to satisfy (3-69). Since we do not use the stepvalues s in the addition, there is no corresponding limit on $s(u|\underline{u})$. From (3-50), (3-67), and the definitions of the Designs 2 and 6 we find the condition:

$$Q(u|\underline{u}) \geq \epsilon \cdot d^{-l},$$

$$\epsilon = \epsilon_2 \triangleq \lambda\left(\frac{\lambda\beta}{\alpha}\right)^{c-u}, \text{ (Design 2), or } \epsilon = \epsilon_6 \triangleq \left(\frac{\alpha}{\lambda\beta}\right)^{2} \cdot \left(\frac{\lambda^2\beta^3}{\alpha^2}\right)^{c-u}, \text{ (Design 6).}$$

The largest bound occurs for $u = 1$ and the worst-case situation is completed if $Q(1|\underline{u}) = P_{min}$. We obtain the condition:

$$P_{min} \geq \epsilon \cdot d^{-l}, \quad \epsilon = \epsilon_2 \text{ or } \epsilon = \epsilon_6.$$

Again we can use the probability-clipping technique, however, an interesting result follows if we reorder the probabilities, such that $P(0|\underline{u})$ is maximal. Then we have:

$$Q(1|\underline{u}) = P(0|\underline{u}) \geq \frac{1}{c}, \quad Q(u|\underline{u}) \geq \frac{1}{c}, \; u > 1.$$

The Q-method works for all (ordered) probabilities if

$$\frac{1}{c} \geq \epsilon \cdot d^{-l}, \; \epsilon = \epsilon_2 \text{ or } \epsilon = \epsilon_6.$$

This implies that we can fix l and use the encoder and decoder for all sources. This seems a perfect solution, but still it increases the redundancy. From the bounds (3-51) and (3-68) and the fact that $\bar{u}$ is not maximal if the probabilities are ordered as required above, we find that there can

be an increase in code rate. This actually happens as we see from section 3.11, and this seems acceptable when we realize that the Designs 2 and 6 adjust the stepvalues for the first symbols most.

3.13. Discussion and conclusion.

In this section we compare Rissanen's code [9] and Pasco's code [10] with those described in chapter 3. First from Formula (3-19) we find the upperbound to the redundancy of Rissanen's code:

$$r < \epsilon + 2^{-q} + O(\tfrac{1}{n}) \qquad (3\text{-}71)$$

Now comparing Rissanen's condition (3-18) with the global P-test we can 'equate' $2^{-\epsilon}$ with $\frac{\alpha}{\beta}$ or $\epsilon \cong \log_2 \frac{\beta}{\alpha}$. Also Rissanen's table $e(x)$ has 2^q r digits precise entries. Our table $\tilde{A}$ contains N k-digit numbers so we are led to relating N with 2^q. From this we see that (3-71) and the Design 1 upperbound (3-49) actually are the same bounds.

The same holds for Pasco's code. Recall (3-16) and observe that

$$-\log_d(1 - 2^{1-k}) > \log_d(1 + 2^{1-k}).$$

However, the difference is small. Also if $\tilde{A}[i]$ is defined not as in Example 2 but as the k digits precise truncated approximation of λ^{-i}, i.e. $\tilde{A}[i] = d^{-k} \cdot \lfloor d^k \lambda^{-i} \rfloor$, then $\alpha = 1 - 2^{1-k}$, $\beta = 1$ and so $\log_d \frac{\beta}{\alpha} = -\log_d(1 - 2^{1-k})$. Here we want to remark that Pasco truncates the augends in (3-14). In his code this is the only possible choice to guarantee decodability. The difference between his and our approach is that Pasco uses the given probabilities directly while we adapt them according to the selected table. If we further

consider the influence of truncation in Pasco's work, see (3-17), and we ignore the influence of P_{min}, we can set $q \cong -\log \delta$. Now q is the precision used in the multiplication, because $\tilde{P}(u|\underline{u})$ was defined to be q digits precise. This leads to the approximate equality of our table length N and 2^q, so $\delta \cong \frac{1}{N}$. This shows that Pasco's bound and the global P-bound are essentially the same.

For a complete comparison we need to consider the complexity too. In Rissanen's code the table e contains $r \cdot 2^q$ digits, whereas our table $\tilde{A}$ needs $N \cdot k$ digits. However, in [9] no clear relation between ϵ, q and r has been given, complicating the comparison. Rissanen indicates that for binary codes $r \cong q + 2$, somewhat comparable with our optimal choice $k \cong 1 + \log_d(C-1)$, see section 3.8. We conclude that roughly $r \cong k$.

In Pasco's case, the comparison is even more difficult since he uses actual multiplications instead of a table. We stated earlier that a multiplication is more difficult that an addition, however, we must consider the size of the table $\tilde{A}$ in this respect. A table-lookup implementation of a multiplier would require, for a k digit $\times$ q digit input, k digit result table, some $2^{q+k} \cdot k$ digits. As above $2^q \cong N$, resulting in about $N \cdot k \cdot 2^k$ digits for a lookup table as compared to $N \cdot k$ digits for $\tilde{A}$. However, there are more ways to implement the multiplication so a reasonable comparison cannot be made. Although these considerations are somewhat superficial they indicate that Pasco's code and Rissanen's code are similar in performance to our global P-design 1 code.

The coding schemes described here offer a selection from more complex designs with a very low redundancy to the fast but less efficient Designs 5 to 8. The decodability criteria we introduced here allow us the design of efficient codes and give a clear view of the performance we can expect.

If we turn to the results tabulated in section 3.11 and also consider

the computational complexity of the different methods we can conclude that these codes are a practical solution to many data compression problems since they are adaptable to many sources, even sources with memory; in fact, the coding unit only needs the current symbol and its probability vector. Also, if we consider the trade-off between speed and redundancy we can choose between the fast code like Design 6 or its local variation, and an efficient algorithm like Design 1, or its very efficient but very complex local counterpart, Design 3.

Finally, we remark that Langdon and Rissanen introduced a very simple binary arithmetic code, i.e. $c = 2$ and $d = 2$, in [32]. This code needs no table and approximates the probabilities $P(l)$ and $P(h)$, $P(l) \leq P(h)$, by a skew number q or, $P(l) = 2^{-q}$, $P(h) = 1 - 2^{-q}$. So this is also a code that is easily adaptable to different binary sources. The worst case redundancy of this code is about 0.035 bit/symbol, corresponding to a table of ca. 300 binary digits for Design 1 or ca. 1000 binary digits for Design 5.

4. FINAL REMARKS.

In this thesis we described two source coding algorithms. Both compute the codewords from the source data in stead of using a predesigned codebook. A strong similarity exists between these schemes. The VF scheme of chapter 2 might be called the combinatorial or exact coding scheme and the arithmetic code the probabilistic or inexact one. Actually, the arithmetic code developed from the combinatorial scheme as a result of some observations similar to those stated in section 2.8. We will briefly sketch this "derivation".

First, recall that the set cardinalities $M_s(m)$ of the segment sets $\mathcal{M}_s(m)$, (2-4), can be overestimated without losing the decodability property of the code, as long as for all states s the following holds:

$$\tilde{M}_s(m) \geq \sum_{u:P(u|s)>0} \tilde{M}_{T(u,s)}(m - V(u|s)), \quad 1 \leq m \leq n, \tag{4-1}$$

$$\tilde{M}_s(m) \geq 1, \qquad m \leq 0.$$

Note that $\tilde{M}_s(m)$ need not be an integer and also observe the similarity between (4-1) and the decodability criteria for the arithmetic codes, Lemmas 3-1 and 3-2. So, as already said in section 2.8, (2-26), $\tilde{M}_s(m)$ can be a finite precision floating point number.

Another reduction in the storage complexity follows if the $\tilde{M}_s(m)$ tables are circular, i.e., if an integer m_0 and a number K exist such that:

$$\tilde{M}_s(m + m_0) = K \cdot \tilde{M}_s(m). \tag{4-2}$$

Note that if $\tilde{M}_s(m)$ is a finite precision d-ary number then K must be a

power of d. Another reduction in the storage requirement results if all the tables contain the same values, i.e. if $\tilde{M}_s(m) = \tilde{M}(m)$ for all states s. We expand a little on the latter. Define:

$$\tilde{M}(m) = \max\{ \sum_{u:P(u|s)>0} \tilde{M}(m - V(u|s)) \mid s \in \mathcal{S}\} \qquad (4\text{-}3)$$

It is not difficult to see that the Theorems 2-1 and 2-2 carry over to similar theorems for the table as defined in (4-3). So the resulting code still achieves the source entropy asymptotically. However, compared to the original code the rate of convergence is slower.

Example: Again we use the source of Figure 2-1 and the stepvalues as tabulated in section 2.8. We obtain:

$u\|s$	$V^{(1)}$	$V^{(2)}$	$V^{(3)}$	$\tilde{P}^{(1)}$	$\tilde{P}^{(2)}$	$\tilde{P}^{(3)}$
$0\|a$	1	3	9	0.57735	0.60910	0.67780
$1\|a$	3	10	37	0.19245	0.19156	0.20213
$2\|a$	4	14	53	0.11111	0.09891	0.10125
$0\|b$	2	7	28	0.33333	0.31450	0.29822
$1\|b$	2	7	28	0.33333	0.31450	0.29822
$2\|b$	2	6	21	0.33333	0.37101	0.40356
$0\|c$	0	0	0	1	1	1

$$\lambda_{max}^{(1)} = 1.73205 \quad \lambda_{max}^{(2)} = 1.17969 \quad \lambda_{max}^{(3)} = 1.04416$$
$$D_{\infty}^{(1)} = 0.09021 \quad D_{\infty}^{(2)} = 0.07040 \quad D_{\infty}^{(3)} = 0.01248$$

In this table $\lambda_{max}^{(i)}$ denotes the exponential growth of $\tilde{M}(m)$ as given by (4-3) with the steps $V^{(i)}$, $i = 1, 2, 3$. $D_{\infty}^{(i)}$ is the resulting pseudodivergence or asymptotic redundancy. Compare this with the table of section 2.8.

The flexibility of the code using this table has increased, because we can use any set of stepvalues $\tilde{V}(u)$, $u \in \mathcal{U}$, as long as the resulting exponential growth parameter $\tilde{\lambda}$ does not exceed the actual growth λ_{max} of the table, or equivalently:

$$\sum_{u:P(u)>0} \lambda_{max}^{-\tilde{V}(u)} \leq 1 \qquad (4\text{-}4)$$

Compare (4-4) with the Kraft inequality e.t.c.!

Combining the ideas of (4-1) - (4-3) with Elias' algorithm resulted in the arithmetic codes of chapter 3.

ACKNOWLEDGEMENTS

I wish to thank Prof. J.P.M. Schalkwijk for introducing me to the subject of Information Theory, for conveying his enthusiasm in it, and for his valuable advice during this research.

For their support in these years and especially for relieving me of all other tasks during the writing of this thesis I thank all my colleagues at the Information and Communication Theory group.

Above that I wish to mention and thank Frans Willems, with whom I cooperated these years and whose contributions to this work are too numerous to mention.

APPENDICES.

Appendix I.

A segment source with a reducible state set.

In this appendix we give a Markov source and a collection of segment sets, resulting from the selection rules given in (2-4), such that the corresponding segment source contains a reducible state set. In Figure I-1 a binary-alphabet Markov source is depicted. This source is characterized by:

the source letter alphabet $\mathcal{U} = \{ 0, 1\}$,

the state set $\mathcal{S} = \{ a, b, c, d\}$,

the letter probabilities

$P(0|a) = 0.68$, $P(0|b) = 0.50$,

$P(0|c) = 0.32$, $P(0|d) = 0.50$,

and the next state function T as is shown in Figure I-1.

The stationary probability vector is

$q(a) = 0.2164$, $q(b) = 0.2490$,

$q(c) = 0.3182$, $q(d) = 0.2164$,

and the source entropy is

$$H_\infty(P,T) = 0.9489 \text{ bit / source letter.}$$

We assign the following stepvalues:

$V(0|a) = 1, \quad V(1|a) = 3, \quad V(0|b) = 2, \quad V(1|b) = 2,$

$V(0|c) = 3, \quad V(1|c) = 1, \quad V(0|d) = 2, \quad V(1|d) = 2.$

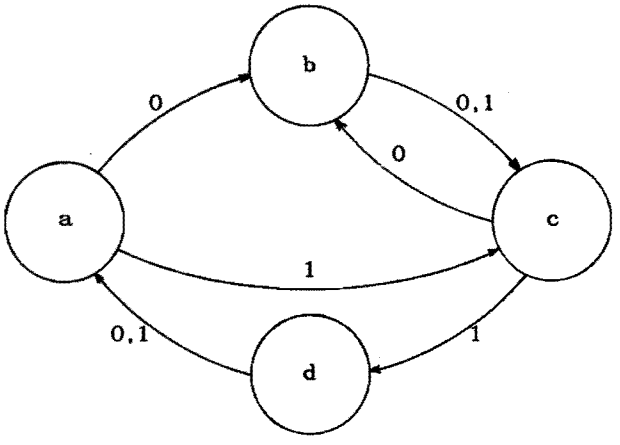

Figure I-1.

It is easily checked that the segment source resulting from this source together with the segment sets $\mathcal{M}_s(5)$, s = a, b, c, d, as defined by (2-4), contains a reducible state set. The states a and b form the irreducible subset $\mathcal{S}_1$, state c makes the irreducible subset $\mathcal{S}_2$, and state d is a transient state, from which the segment source always enters subset $\mathcal{S}_2$. Thus we obtain

$$\Pr\{\mathcal{S}_1\} = q(a) + q(b) = 0.4654,$$

$$\Pr\{\mathcal{S}_2\} = q(c) + q(d) = 0.5346,$$

$$q_1(a) = 0.68, \quad q_1(b) = 0.32,$$

$$q_2(c) = 1.$$

Also

$M_{max} = 9,$

$E_{\underline{U}|s}\{L(\underline{U})\} = 3.36$ letter/segment for s = a,

$= 2.68$ letter/segment for s = b,

$= 3.1424$ letter/segment for s = c,

$= 2.68$ letter/segment for s = d.

Therefore, $EL = 3.1424$ letter/segment and the code rate $R = 1.0088$ bit/source letter. So, this particular code performs worse than not coding the source sequence, however, by analising the asymptotic behaviour we find that for sets $M_s(m)$, with $m > 40$, the coding scheme actually compresses the source sequence, and asymptotically achieves the rate $R_\infty = 0,9491$.

Appendix II.

The proof of Lemma 2-1.

Call the actual state in which the Markov source starts s_{start} and assume that the source enters the irreducible subset $\mathcal{S}_r$. Then with (conditional) probability one the source generates a sequence of segments $\underline{u}_1$, $\underline{u}_2$, $\underline{u}_3$, ... such that

$$\lim_{M\to\infty} \frac{1}{M} \sum_{m=1,M} L_t((\underline{u})_m, T((\underline{u})^{m-1}, s_{start})) = \sum_{s\in\mathcal{S}_r} q_r(s) E_{\underline{U}|s}\{L_t(\underline{U},s)\}. \qquad \text{(II-1)}$$

Alternatively

$$\lim_{M\to\infty} \frac{1}{M} \sum_{m=1,M} L_t((\underline{u})_m, T((\underline{u})^{m-1}, s_{start}))$$

$$= \lim_{M\to\infty} \frac{1}{M} \sum_{m=1,M} L((\underline{u})_m, T((\underline{u})^{m-1}, s_{start})) \cdot \frac{\sum_{m=1,M} L_t((\underline{u})_m, T((\underline{u})^{m-1}, s_{start}))}{\sum_{m=1,M} L((\underline{u})_m, T((\underline{u})^{m-1}, s_{start}))}$$

$$= \lim_{M\to\infty} \frac{\sum_{m=1,M} L_t((\underline{u})_m, T((\underline{u})^{m-1}, s_{start}))}{\sum_{m=1,M} L((\underline{u})_m, T((\underline{u})^{m-1}, s_{start}))} \cdot \lim_{M\to\infty} \frac{1}{M} \sum_{m=1,M} L((\underline{u})_m, T((\underline{u})^{m-1}, s_{start}))$$

$$= q(t) \cdot \sum_{s\in\mathcal{S}_r} q_r(s) E_{\underline{U}|s}\{L(\underline{U}, s)\}. \qquad \text{(II-2)}$$

where the last equality again holds with (conditional) probability one.

From (II-1) and (II-2) it follows that

$$\sum_{s\in\mathcal{S}_r} q_r(s) E_{\underline{U}|s}\{L_t(\underline{U}, s)\} = q(t) \cdot \sum_{s\in\mathcal{S}_r} q_r(s) E_{\underline{U}|s}\{L(\underline{U}, s)\}. \qquad \text{(II-3)}$$

Combining (II-3) with (2-10) and (2-11) proves the lemma.

Appendix III.

The proof of Lemma 2-3.

The matrix $A(\lambda)$ as defined in (2-18) is irreducible (since the Markov

source contains one irreducible state set $\mathcal{S}$) and nonnegative. Therefore, Frobenius' theorem (see Gantmacher [16, Chapter III, p.65]) applies. This theorem states the following:

i. an irreducible nonnegative matrix A always has a positive characteristic number r,

ii. the moduli of all the other characteristic numbers are at most r, and

iii. a characteristic vector with positive coordinates corresponds to the "dominant" characteristic number r.

Now let $\rho_s \triangleq \Sigma_t A_{st}$ for $s \in \mathcal{S}$ and $\rho_{min} \triangleq \min\{\rho_s | s \in \mathcal{S}\}$. Then for the characteristic number r of A we have that (see [16, p.76]):

$$r \geq \rho_{min}.$$

For $\lambda = 1$ we obtain that $r(1) \geq \rho_{min}(1) \geq 1$. For $\lambda \to \infty$, since the stepsizes are chosen such that no zero-circuits exist, we know that

$$A^{|\mathcal{S}|}(\lambda \to \infty) = \underline{0} \qquad \text{(III-1)}$$

where $|\mathcal{S}|$ is the number of states of the source and $\underline{0}$ is the all-zero matrix. Note that $A_{st}^{|\mathcal{S}|}(\lambda \to \infty)$ equals the number of paths, whose stepvalues sum to zero, from state s to state t. The consequence of (III-1) is that $r(\lambda \to \infty) = 0$. Since $r(\lambda)$ is continuous in λ and since $r(1) \geq 1$ and $r(\lambda \to \infty) = 0$ a largest value of λ must exist for which $r(\lambda) = 1$. Call this value $\tilde{\lambda}$. We will now prove that $\tilde{\lambda} = \lambda_{max}$. It is obvious that $\tilde{\lambda} \leq \lambda_{max}$. Suppose that $\tilde{\lambda} < \lambda_{max}$. Since $r(\lambda)$ is continuous in λ, it is clear that $r(\lambda_{max}) < 1$. Since $A(\lambda_{max})$ has a characteristic number equal to 1, then because of the

Frobenius theorem part ii, $r(\lambda_{max}) \geq 1$. This contradicts the assumption that $\tilde{\lambda} < \lambda_{max}$. Therefore, $\tilde{\lambda} = \lambda_{max}$. Frobenius part iii finally yields the positiveness of the coordinates of $\underline{e}$.

Appendix IV.

The proof of Lemma 2-4.

From (2-6) and Lemma 2-3 it follows for $s \in \mathcal{S}$ and $1 - V_{max} \leq n \leq 0$ that

$$(\lambda_{max})^n e_s \leq M_s(n) \leq \frac{1}{e_{min}} (\lambda_{max})^{n + V_{max} - 1} e_s. \qquad \text{(IV-1)}$$

Note that this is only possible because all components of $\underline{e}$ are positive.

We will now prove the lemma by induction. The hypothesis is that (IV-1) holds for $s \in \mathcal{S}$ and $1 - V_{max} \leq n \leq m - 1$ where $m \geq 1$. By (2-6) for $n = m$ and for each $s \in \mathcal{S}$ we have that

$$\begin{aligned} M_s(n) &\triangleq \sum_{u:P(u|s)>0} M_{T(u,s)}(n - V(u|s)) \\ &\geq \sum_{u:P(u|s)>0} \lambda_{max}^{n - V(u|s)} e_{T(u,s)} \\ &= \lambda_{max}^{n} \sum_{t \in \mathcal{S}} \sum_{\substack{u:P(u|s)>0 \\ T(u,s)=t}} \lambda_{max}^{-V(u|s)} e_t \\ &\overset{(*)}{=} \lambda_{max}^{n} \sum_{t \in \mathcal{S}} A_{st}(\lambda_{max}) e_t = \lambda_{max}^{n} e_s, \end{aligned}$$

and, analogously,

$$M_s(n) \leq \sum_{u:P(u|s)>0} \frac{1}{e_{min}} \lambda_{max}^{n - V(u|s) + V_{max} - 1} e_{T(u,s)}$$

$$= \frac{1}{e_{min}} \lambda_{max}^{n + V_{max} - 1} \sum_{t \in \mathcal{S}} A_{st}(\lambda_{max}) e_t$$

$$\overset{(*)}{=} \frac{1}{e_{min}} \lambda_{max}^{n + V_{max} - 1} e_s .$$

The crucial steps (*) follow from the fact that $\underline{e}$ is a characteristic vector of $A(\lambda_{max})$ with characteristic number 1.

Appendix V.

The proof of Lemma 2-5.

Let A be the matrix $A(\lambda_{max})$ and Z be the diagonal matrix with diagonal values e_s, $s \in \mathcal{S}$. Now

$$B \triangleq Z^{-1}AZ$$

is a stochastic matrix (see [16, chapter III, p.101]).

Then for $n = 1, 2, 3, \ldots$ we have for the n^{th} power of A that

$$A^{(n)} = (ZBZ^{-1})^{(n)} = ZB^{(n)}Z^{-1} \qquad (V\text{-}1)$$

where we used the fact that $A = ZBZ^{-1}$.

We now proceed with some notation. First observe that the state transition pseudoprobability $A(t|s) = A_{st}(\lambda_{max})$. Then denote the following for $n = 1, 2, 3, \ldots$

- the stationary probabilities $q(s)$ by q,
- the conditional probabilities $W^n(t_1, t_2, \ldots, t_n|s)$ by $W_1 \times W_2 \times \ldots \times W_n$ (where W is the state transition probability matrix)
- the joint probabilities $W^n(t_1, t_2, \ldots, t_n|s)q(s)$ by $W_1 \times W_2 \times \ldots \times W_n \times q$, and
- the conditional pseudoprobabilities $A^n(t_1, t_2, \ldots, t_n|s)$ by $A_1 \times A_2 \times \ldots \times A_n$.

Furthermore, note that the (marginal) conditional probabilities $W_n(t_n|s)$ equal $W^{(n)}$, the n^{th} power of W. Analogously, the (marginal) conditional pseudoprobabilities $A_n(t_n|s)$ equals $A^{(n)}$.

First we have that

$$\begin{aligned} D_n &\triangleq D(W_1 \times W_2 \times \ldots \times W_n \| A_1 \times A_2 \times \ldots \times A_n | q) \\ &= D(W_1 \| A_1 | q) + D(W_2 \| A_2 | W_1 \times q) \\ &\qquad + \ldots + D(W_n \| A_n | W_1 \times W_2 \times \ldots \times W_{n-1} \times q) \\ &= nD(W_1 \| A_1 | q). \end{aligned} \tag{V-2}$$

Furthermore, by the log-sum inequality, [23, p.48]

$$D_n \geq D(W^{(n)} \| A^{(n)} | q). \tag{V-3}$$

Combining (V-1), (V-2), and (V-3) we find that

$$\begin{aligned} &D(W_1 \| A_1 | q) \\ &\quad \geq \frac{1}{n} D(W^{(n)} \| A^{(n)} | q) = \frac{1}{n} D(W^{(n)} \| ZB^{(n)}Z^{-1} | q) \\ &\quad = \frac{1}{n} D(W^{(n)} \times q \| ZB^{(n)}Z^{-1} \times q) \\ &\quad \geq \frac{-1}{n} \cdot \log_2 \Big(\sum_{t,s} ZB^{(n)}Z^{-1}(t|s)q(s) \Big) \end{aligned}$$

where the last step again follows from the log-sum inequality and the fact that $W^{(n)}$ is a stochastic matrix resulting in $\Sigma_{t,s} W^{(n)}(t|s)q(s) = 1$. Now observe that since $B^{(n)}$ is a stochastic matrix,

$$\sum_{t,s} ZB^{(n)}Z^{-1}(t|s)q(s) \leq \frac{1}{e_{min}}.$$

Therefore, for all $n = 1, 2, 3, \ldots$,

$$D(W_1 \| A_1 | q) \geq \frac{-1}{n} \log_2 \frac{1}{e_{min}}$$

yields

$$D(W_1 \| A_1 | q) \geq 0.$$

Finally, if we extrapolate the notation somewhat,

$$\begin{aligned} D_\infty(P, \tilde{P}, T) &= D(P_1 \| \tilde{P}_1 | q) \\ &\geq D(W_1 \| A_1 | q) \\ &\geq 0. \end{aligned}$$

Here the first inequality "surprisingly" follows from the log-sum inequality.

Appendix VI.

The proof of Theorem 2-2.

First we define

$$U_{min} \triangleq \min\{-\log_2 P(u|s) \,|\, u\in\mathcal{U},\ s\in\mathcal{S},\ 0<P(u|s)<1\},$$
$$U_{max} \triangleq \max\{-\log_2 P(u|s) \,|\, u\in\mathcal{U},\ s\in\mathcal{S},\ 0<P(u|s)\},$$

then

$$V_{\gamma,min} \triangleq \min\{V_\gamma(u|s) \,|\, u\in\mathcal{U},\ s\in\mathcal{S},\ 0<P(u|s)<1\} \geq \gamma U_{min}, \qquad \text{(VI-1)}$$
$$V_{\gamma,max} \triangleq \max\{V_\gamma(u|s) \,|\, u\in\mathcal{U},\ s\in\mathcal{S},\ 0<P(u|s)\} < \gamma U_{max} + 1. \qquad \text{(VI-2)}$$

Now we have for $\underline{u} \in \mathcal{M}_s(n)$ that

$$\begin{aligned} -\log_2 P(\underline{u}|s) &= \frac{1}{\gamma} \sum_{k=1,L(\underline{u})} -\gamma \log_2 P(u_k | T(\underline{u}^{k-1}, s)) \\ &\leq \frac{1}{\gamma} \sum_{k=1,L(\underline{u})} V(u_k | T(\underline{u}^{k-1}, s)) \\ &\leq \frac{1}{\gamma} (n + V_{\gamma,max} - 1) \\ &< \frac{1}{\gamma} (n + \gamma U_{max}) = \frac{n}{\gamma}(1 + \frac{\gamma U_{max}}{n}) \end{aligned} \qquad \text{(VI-3)}$$

and

$$\begin{aligned} -\log_2 P(\underline{u}|s) &> \frac{1}{\gamma}(\sum_{k=1,L(\underline{u})} V(u_k | T(\underline{u}^{k-1}, s)) - L(\underline{u})) \\ &> \frac{1}{\gamma}(n - (\frac{n}{V_{\gamma,min}} + 1)|\mathcal{S}|) \\ &\geq \frac{1}{\gamma}(n - (\frac{n}{\gamma U_{min}} + 1)|\mathcal{S}|) \\ &= \frac{n}{\gamma}(1 - \frac{|\mathcal{S}|}{\gamma U_{min}} - \frac{|\mathcal{S}|}{n}). \end{aligned} \qquad \text{(VI-4)}$$

To obtain (VI-3) and (VI-4) we used the definition of $V_\gamma(u|s)$, the definition of $\mathcal{M}_s(n)$ (see (2-4)), and (VI-1) and (VI-2).

It follows from (2-19) and (2-20) that

$$\alpha_\gamma = \lim_{n\to\infty} \frac{\log_2 M_{max}(n)}{n}.$$

Therefore, with (VI-3) and (VI-4) we find that

$$\frac{1}{\gamma}\left(1 - \frac{|\varphi|}{\gamma U_{min}}\right) < \alpha_\gamma < \frac{1}{\gamma}.$$

This proves that

$$\lim_{\gamma\to\infty} \gamma\cdot\alpha_\gamma = 1. \tag{VI-5}$$

Note that

$$\lim_{\gamma\to\infty} \alpha_\gamma = 0. \tag{VI-6}$$

Now for u, s for which $P(u|s) > 0$ we obtain for the pseudoprobabilities

$$\begin{aligned}
\tilde{P}_\gamma(u|s) &\triangleq 2^{-\alpha_\gamma V_\gamma(u|s)} \\
&= 2^{-\alpha_\gamma\left[-\gamma\log_2 P(u|s)\right]} \\
&\le 2^{-\alpha_\gamma(-\gamma\log_2 P(u|s))} \\
&= 2^{\gamma\alpha_\gamma\log_2 P(u|s)} \\
&= P(u|s)^{\gamma\alpha_\gamma}
\end{aligned} \tag{VI-7}$$

and

$$\tilde{P}_\gamma(u|s) \geq 2^{-\alpha_\gamma(-\gamma\log_2 P(u|s) + 1)}$$

$$= 2^{-\alpha_\gamma} P(u|s)^{\gamma\alpha_\gamma}. \qquad \text{(VI-8)}$$

Hence from (VI-7) and (VI-8) using (VI-5) and (VI-6) we can conclude for u, s such that $P(u|s) > 0$ that

$$\lim_{\gamma\to\infty} \tilde{P}_\gamma(u|s) = P(u|s).$$

This proves the theorem.

Appendix VII.

The proof of the test lemma.

Let $\mathcal{M}^{int}$ denote the set of proper prefixes of a set $\mathcal{M}$, where $\mathcal{M}$ is a proper and complete set of strings over a finite alphabet $\mathcal{U}$. So,

$$\mathcal{M}^{int} \triangleq \{\underline{u} \,|\, \exists \underline{v} : L(\underline{v}) > 0 \quad \text{and} \quad \underline{uv} \in \mathcal{M}\}.$$

First we show that if $\mathcal{M}$ is maximal for a probability vector P, then

$$\forall \underline{u} \in \mathcal{M},\ \underline{v} \in \mathcal{M}^{int} : P(\underline{u}) \leq P(\underline{v}). \qquad \text{(VII-1)}$$

Proof: Assume (VII-1) does not hold, i.e., there is a $\underline{u} \in \mathcal{M}$ and a $v \in \mathcal{M}^{int}$ with $P(\underline{u}) > P(\underline{v})$. Without loss of generality, we may take $\underline{v}$ to be a maximal proper prefix, i.e., for all $u \in \mathcal{U}$, $\underline{v}u \in \mathcal{M}$. Now consider the segment set $\mathcal{M}'$ with

$$\mathcal{M}' = (\mathcal{M} - \{\underline{v}u | u \in \mathcal{U}\}) \cup \{\underline{u}u | u \in \mathcal{U}\}$$

so $|\mathcal{M}| = |\mathcal{M}'|$. Now use the well known average length lemma, e.g. Massey [15], which states that

$$\mathbf{EL} \triangleq \sum_{\underline{u} \in \mathcal{M}} P(\underline{u})L(\underline{u}) = \sum_{\underline{v} \in \mathcal{M}^{int}} P(\underline{v}). \tag{VII-2}$$

Then we obtain, denoting the average length of $\mathcal{M}'$ by $\mathbf{EL}'$,

$$\mathbf{EL}' - \mathbf{EL} = P(\underline{u}) - P(\underline{v}) > 0,$$

which contradicts the assumption that $\mathcal{M}$ is maximal. So (VII-1) holds.

Now we prove that if

$$\forall \underline{u} \in \mathcal{M}, \ \underline{v} \in \mathcal{M}^{int} : P(\underline{u}) \leq P(\underline{v}) \tag{VII-3}$$

then $\mathcal{M}$ is maximal.

Proof: Let $\mathcal{M}'$ be a maximal segment set with $|\mathcal{M}'| = |\mathcal{M}|$, and let $\mathcal{M}'^{int}$ be its set of proper prefixes. Define

$$\mathcal{F}_1 \triangleq \mathcal{M}'^{int} \cap \mathcal{M}^{int} \tag{VII-4a}$$

$$\mathcal{F}_2 \triangleq \mathcal{M}^{int} - \mathcal{F}_1 \tag{VII-4b}$$

$$\mathcal{F}_3 \triangleq \mathcal{M}'^{int} - \mathcal{F}_1 \tag{VII-4c}$$

Since $\mathcal{M}'$ is maximal we have $\mathbf{EL} \leq \mathbf{EL}'$. Assume that

$$\mathbf{EL} < \mathbf{EL}', \tag{VII-5}$$

then, using (VII-2) and (VII-4),

$$\sum_{\underline{v}\in\mathcal{F}_2} P(\underline{v}) < \sum_{\underline{w}\in\mathcal{F}_3} P(\underline{w}).$$

And with $|\mathcal{F}_2| = |\mathcal{F}_3|$ we know that a pair $\underline{v} \in \mathcal{F}_2$, $\underline{w} \in \mathcal{F}_3$ exists with

$$P(\underline{v}) < P(\underline{w}) \qquad \text{(VII-6)}$$

Now from (VII-4) we get $\underline{w} \notin \mathcal{M}^{int}$. So $\underline{w}$, or a proper prefix of $\underline{w}$, belongs to $\mathcal{M}$. Also, $\underline{v} \in \mathcal{M}^{int}$, and with (VII-3) we find

$$P(\underline{v}) \geq P(\underline{w}),$$

which contradicts (VII-6) and (VII-5), thus EL = EL'. This proves the final part.

Appendix VIII.

The region of optimality for Tunstall codes.

We demonstrate the following two related results for the algorithm for discrete memoryless sources.

r1) Given a (proper and complete) segment set $\mathcal{M}$, we find the subset of DMS for which $\mathcal{M}$ is maximal.

r2) Given a Tunstall segment set $\mathcal{M}_T$, satisfying an extra constraint stated below, we show that a segment set $\mathcal{M}(n)$, as defined in (2-28), exists such that $\mathcal{M}(n) = \mathcal{M}_T$.

First we introduce some notations.

The <u>type</u> of a string $\underline{u} \in \mathcal{U}^*$ is defined as the $|\mathcal{U}|$ dimensional vector

$$\underline{b}(\underline{u}) \triangleq (b_u(\underline{u}) \,|\, u \in \mathcal{U})$$

with

$$b_u(\underline{u}) \triangleq |\{i \,|\, 1 \leq i \leq L(\underline{u}),\ u_i = u\}|, \qquad u \in \mathcal{U}.$$

Given a set $\mathcal{M}$, denote by $\mathfrak{B}(\mathcal{M})$ the <u>set of types</u> of all segments in $\mathcal{M}$. These sets can easily be visualized as a set of gridpoints in the space $(\mathbb{R}^{\geq 0})^{|\mathcal{U}|}$, i.e., every coordinate x_u, $u \in \mathcal{U}$, is non-negative.

Let $\mathbb{W}$ be a plane in this space. We say that $\mathbb{W}$ <u>agrees with</u> a segment set $\mathcal{M}$ if all points of $\mathfrak{B}(\mathcal{M}^{int})$ lie inside, or on the boundary of the subspace enclosed by $\mathbb{W}$ and the planes $x_u = 0$, $u \in \mathcal{U}$, and the points of $\mathfrak{B}(\mathcal{M})$ lie outside this subspace or on $\mathbb{W}$. We give an example.

Example: Let $\mathcal{U} = \{0, 1\}$ and $\mathbb{W}$: $(-\log_2 0.7)x_0 + (-\log_2 0.3)x_1 = 3.6$. See Figure VIII-1. From this figure we see that $\mathbb{W}$ agrees with the following segment set $\mathcal{M}$:

$$\begin{aligned}\mathcal{M} = \{&0000000,\ 0000001,\ 000001,\ 00001,\ 00010,\ 00011,\ 00100,\\ &00101,\ 0011,\ 01000,\ 01001,\ 0101,\ 011,\ 10000,\\ &10001,\ 1001,\ 101,\ 110,\ 111\} \qquad \text{(VIII-1)}.\end{aligned}$$

So, $\mathfrak{B}(\mathcal{M}) = \{(7,0), (6,1), (5,1), (4,1), (3,2), (2,2), (1,2), (0,3)\}$, and is indicated by the crosses in Figure VIII-1.

Define for a probability vector P on $\mathcal{U}$, the planes $\mathbb{W}_P(\Omega)$ as follows:

$$\mathbb{W}_P(\Omega): \sum_{u \in \mathcal{U}} (-\log_2 P(u)) x_u = \Omega$$

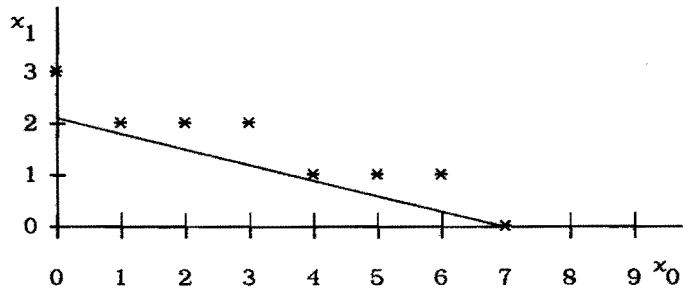

Figure VIII-1.

Now we can reformulate the testlemma.

If a segment set $\mathcal{M}$ is optimal for a probability vector P then $\mathcal{M}$ agrees with $\mathbb{W}_P(\Omega)$ for some $\Omega \in \mathbb{R}$ and vice versa.

In the example of this appendix the plane $\mathbb{W}$ is $\mathbb{W}_P(3.6)$ for $P = \{0.7, 0.3\}$. So $\mathcal{M}$, (VIII-1), is maximal for this P.

Given a set $\mathcal{M}$, define the plane set $\mathcal{W}(\mathcal{M})$ as follows:

$$\mathcal{W}(\mathcal{M}) \triangleq \{\mathbb{W} | \mathbb{W} \text{ agrees with } \mathcal{M}\}$$

Now the first claim (r1) follows from the fact that $\mathcal{W}(\mathcal{M})$ can be described by a finite set of 'extreme' planes, and these define the range of probability vectors for which $\mathcal{M}$ is maximal. We clarify this, continuing our example.

Let $\mathcal{M}$ be defined by (VIII-1) then, see Figure VIII-2, $\mathcal{W}(\mathcal{M})$ is bounded by:

$$\mathbb{W}_1: \quad 2x_0 + 5x_1 = 12 \quad \text{and} \quad \mathbb{W}_2: \quad 2x_0 + 7x_1 = 14.$$

Using the definition of W_P, we see that

$$W_1 = W_{P_1}(\Omega_1), \quad P_1 = (r^2, r^5), \quad \Omega_1 = -12\log_2 r,$$

and r is the solution of: $r^2 + r^5 = 1$, so $r = 0.8087$.

$$W_2 = W_{P_2}(\Omega_2), \quad P_2 = (s^2, s^7), \quad \Omega_2 = -12\log_2 s,$$

and s is the solution of: $s^2 + s^7 = 1$, so $s = 0.8398$.

So $\mathcal{M}$ is optimal if and only if $0.6540 \leq P(0) \leq 0.7053$.

We show that a Tunstall set $\mathcal{M}_T$ can be generated by the algorithm if it satisfies the following (weak) extra condition:

$\mathcal{M}_T$ satisfies the testlemma without equality, i.e.,

$$\forall\, \underline{u} \in \mathcal{M}_T,\ \underline{v} \in \mathcal{M}_T^{int} : P(\underline{u}) < P(\underline{v}). \qquad \text{(VIII-2)}$$

This implies that no point from $\mathcal{P}(\mathcal{M}^{int})$ is on the plane W_P. Now, the algorithm generates sets $\mathcal{M}(n)$ with planes $W(n)$: $\sum_{u \in \mathcal{U}} V(u) x_u = n$.

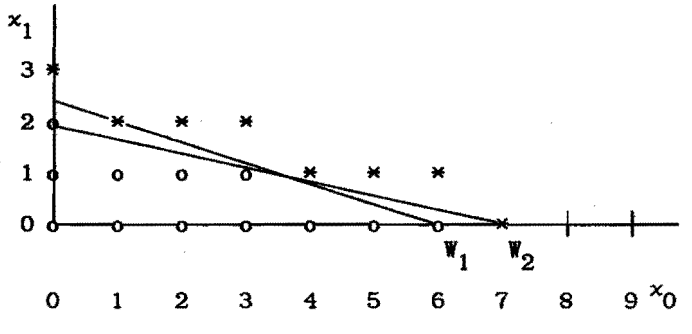

Figure VIII-2.

If $\mathcal{M}_T$ satisfies (VIII-2) and a $W(n)$ exists such that $W(n) \in \mathcal{W}(\mathcal{M}_T)$, then

$\mathcal{M}(n) = \mathcal{M}_T$. Now this is always possible since $\mathcal{H}(\mathcal{M}_T)$ contains planes described with rational $\omega(u)$ and Ω and due to (VIII-2) these planes do not contain points from $\mathcal{B}(\mathcal{M}^{int})$. Thus proving claim (r2).

Appendix IX.

The proof of the global tests.

In this appendix we will give the proof of the global tests (3-35) and (3-36).

Proof of Lemma 3-3: We only have to show that, if

$$\sum_{u\in\mathcal{U}} \lambda^{-s(u|\underline{u}^i)} \leq \frac{\alpha}{\beta}$$

then, for all local positions $S(\underline{u}^i)$, (3-31) holds.

Note: $s(u|\underline{u}^i)$ only depends on $P(u|\underline{u}^i)$, not on the local position, i.e. if the source is memoryless then $s(u|\underline{u}^i) = s(u)$.

From (3-22): $\quad A[S(\underline{u}^i)] \geq \alpha\cdot\lambda^{-S(\underline{u}^i)}$

From (3-22): $\quad \sum_{u\in\mathcal{U}} A[S(\underline{u}^i) + s(u|\underline{u}^i)] \leq \beta\cdot\lambda^{-S(\underline{u}^i)} \sum_{u\in\mathcal{U}} \lambda^{-s(u|\underline{u}^i)}$

Together with (3-35) we obtain

$$\sum_{u\in\mathcal{U}} A[S(\underline{u}^i) + s(u|\underline{u}^i)] \leq \beta\cdot\lambda^{-S(\underline{u}^i)}\cdot\frac{\alpha}{\beta} \leq A[S(\underline{u}^i)]$$

Q.E.D.

The proof of Lemma 3-4 is a bit more involved, since we must also consider the stepsums $T(u|\underline{u}^i)$. We take the following approach. First fix the stepvalues $s(u|\underline{u}^i)$. Then try to select stepsums $T(u|\underline{u}^i)$ such that decodability is guarantied. Now, what bounds on $s(u|\underline{u}^i)$ follow?

$$\text{If} \qquad \lambda^{-T(u+1|\underline{u}^i)} \geq \frac{\beta}{\alpha}\{\lambda^{-T(u|\underline{u}^i)} + \lambda^{-s(u|\underline{u}^i)}\} \qquad \text{(IX-1)}$$

then the local condition (3-32) at $S(\underline{u}^i)$ is satisfied for all $S(\underline{u}^i)$. The proof of this involves only the application of (3-22) as before.

$$\text{If} \qquad T(u+1|\underline{u}^i) = -\lceil \log_\lambda \{\frac{\beta}{\alpha}(\lambda^{-T(u|\underline{u}^i)} + \lambda^{-s(u|\underline{u}^i)})\}\rceil \qquad \text{(IX-2)}$$

then (IX-1) is also true.

This implies that we can find stepsums such that both (3-32) holds and

$$\lambda^{-T(u+1|\underline{u}^i)} < \frac{\lambda\beta}{\alpha}\{\lambda^{-T(u|\underline{u}^i)} + \lambda^{-s(u|\underline{u}^i)}\} \qquad \text{(IX-3)}$$

Observe the encoding formula (3-25) and conclude that for the symbol $u_{i+1} = 0$ we do not have to add anything to $B(\underline{u}^i)$. We may set

$$T(0|\underline{u}^i) = \infty \qquad \text{(IX-4)}$$

Using (IX-3) and (IX-4) we can prove by induction on u that

$$\lambda^{-T(u+1|\underline{u}^i)} < \frac{\lambda\beta}{\alpha}\sum_{v=0}^{u}(\frac{\lambda\beta}{\alpha})^{u-v}\cdot\lambda^{-s(v|\underline{u}^i)} \qquad \text{(IX-5)}$$

Now we know that for given stepvalues we can find, by (IX-2), stepsums,

such that (3-32) is satisfied. If (3-33) also holds then decodability is guarantied. So, suppose that the global Q-test holds, then by (IX-5)

$$1 \geq \sum_{u \in \mathcal{U}} \left(\frac{\lambda\beta}{\alpha}\right)^{c-u} \cdot \lambda^{-s(u|\underline{u}^i)} > \lambda^{-T(\omega+1|\underline{u}^i)}$$

so (3-33) holds and the code is decodable.

Summarizing: If the stepvalues are selected such that (3-36) is satisfied, then by (IX-2) stepsums can be found such that the resulting code is decodable independent of $S(\underline{u}^i)$.

Appendix X.

The decodability of Design 2.

Here we want to show that the global method in Design 2 generates a decodable code. First we show that the selection

$$s(u|\underline{u}^i) = \lceil (c - u) \cdot \log_\lambda \frac{\lambda\beta}{\alpha} - \log_\lambda P(u|\underline{u}^i) \rceil \qquad (3\text{-}38)$$

satisfies the global Q-test.

Proof: By (3-38): $\lambda^{-s(u|\underline{u}^i)} \leq \left(\frac{\alpha}{\lambda\beta}\right)^{c-u} \cdot P(u|\underline{u}^i)$ (X-1)

So

$$\sum_{u \in \mathcal{U}} \left(\frac{\lambda\beta}{\alpha}\right)^{c-u} \cdot \lambda^{-s(u|\underline{u}^i)} \leq \sum_{u \in \mathcal{U}} P(u|\underline{u}^i) = 1$$

Q.E.D.

As shown in Appendix IX we know that there exist stepsums in accordance with the steps found, but the question remains whether (3-39) is a good

choice. We proceed to prove that this is the case:

First from $$T(u|\underline{u}^i) = \lceil (c-u)\log_\lambda \frac{\lambda\beta}{\alpha} - \log_\lambda Q(u|\underline{u}^i) \rceil \qquad (3\text{-}39)$$

we find

$$\lambda^{-1}(\frac{\lambda\beta}{\alpha})^{u-c} Q(u|\underline{u}^i) < \lambda^{-T(u|\underline{u}^i)} \leq (\frac{\lambda\beta}{\alpha})^{u-c} Q(u|\underline{u}^i). \qquad (X\text{-}2)$$

We verify (3-33):

$$\lambda^{-T(\omega+1|\underline{u}^i)} \leq (\frac{\lambda\beta}{\alpha})^{\omega+1-c} Q(\omega+1|\underline{u}^i) = 1$$

So $T(\omega+1|\underline{u}^i) \geq 0$ in accordance with (3-33).

Now it remains to check (3-32). In Appendix IX we stated that if (IX-1) holds then (3-32) holds for all local positions. We proceed:

$$\begin{aligned} \lambda^{-T(u+1|\underline{u}^i)} &\overset{(a)}{>} \lambda^{-1}(\frac{\lambda\beta}{\alpha})^{u+1-c}[Q(u|\underline{u}^i) + P(u|\underline{u}^i)] \\ &= \frac{\beta}{\alpha}\cdot(\frac{\lambda\beta}{\alpha})^{u-c} Q(u|\underline{u}^i) + \frac{\beta}{\alpha}\cdot(\frac{\lambda\beta}{\alpha})^{u-c} P(u|\underline{u}^i) \\ &\overset{(b)}{\geq} \frac{\beta}{\alpha}\cdot\lambda^{-T(u|\underline{u}^i)} + \frac{\beta}{\alpha}\cdot\lambda^{-s(u|\underline{u}^i)} \end{aligned}$$

Here (a) follows from (X-2) and (b) from (X-1) and (X-2). So (IX-1) holds and thus (3-32) and (3-33) hold.

Remark: By using (3-33) and (IX-1) in the proof we eliminated the need to know the local position $S(\underline{u}^i)$ in the table. In this way we again obtain a global result.

Appendix XI.

The decodability of Design 6.

This appendix contains the proof that the global Q-design 6 is decodable. From (3-61) - (3-66) we find that:

$$\left(\frac{\alpha}{\lambda\beta}\right)^2 \left(\frac{\lambda^2\beta^3}{\alpha^2}\right)^{u-c} \cdot P(u|\underline{u}) \le \lambda^{-s(u|\underline{u})} \le \left(\frac{\lambda^2\beta^3}{\alpha^3}\right)^{u-c} \cdot P(u|\underline{u})$$
$$\left(\frac{\alpha}{\lambda\beta}\right)^2 \left(\frac{\lambda^2\beta^3}{\alpha^2}\right)^{u-c} \cdot Q(u|\underline{u}) \le \lambda^{-T(u|\underline{u})} \le \left(\frac{\lambda^2\beta^3}{\alpha^3}\right)^{u-c} \cdot Q(u|\underline{u}) \qquad \text{(XI-1)}$$

We verify (3-33) using (XI-1):

$$\lambda^{-T(\omega+1|\underline{u})} \le Q(\omega+1|\underline{u}) = 1. \qquad \text{(XI-2)}$$

Now, from Appendix IX we know that if (IX-1) is satisfied then (3-32) holds. So:

$$\lambda^{-T(u+1|\underline{u})} \ge \left(\frac{\lambda^2\beta^3}{\alpha^2}\right)^{u-c} \cdot \frac{\beta}{\alpha} \cdot [Q(u|\underline{u}) + P(u|\underline{u})]$$
$$\ge \frac{\beta}{\alpha} \cdot [\lambda^{-T(u|\underline{u})} + \lambda^{-s(u|\underline{u})}],$$

and this, with (XI-2), proves that the local Q-test, Lemma 3-2, holds for all positions $S(\underline{u})$, so the code is decodable.

Appendix XII.

Simulation results.

This appendix contains the results of the simulations done with the eight designs. The binary codes were designed using a table $\tilde{A}$ as given in Example 2, so $\tilde{A}[i] \triangleq d^{-k} \cdot \lceil d^{k} \cdot d^{-i/N} \rceil$, $0 \leq i < N$; k and N are positive integers.

The storage complexity for these codes is $N \cdot k$ binary digits. We designed tables with $N \cdot k \cong 1000$ and $N \cdot k \cong 10000$, and k several values in the range from 10 to 20. We selected values that should be representative for the algorithms. For instance, the minimum of $\log \frac{\lambda\beta}{\alpha}$ under the restriction that $N \cdot k = C$ occurs at $k_{opt} = 1 + \log_d C-1$. For $C = 1000$ this gives $k_{opt} \cong 11$ and if $C = 10000$ then $k_{opt} \cong 14$. Since the local designs generally performed best with rather precise tables, i.e. large k, we also selected values in that range.

In Table 1 we list the global P-method redundancy bounds. The first data column states the lowerbound result, (3-54), the second column gives the upperbound for Design 1, (3-49), and the last column tabulates the upperbound (3-60) for Design 5.

We select the following memoryless sources:

Source B1: $c = 2$; $P(0) = 0.75$, $P(1) = 0.25$; $H(P(U)) = 0.8113$ bits.

$\bar{u} = 0.25$.

Source B2: $c = 2$; $P(0) = 0.999139$, $P(1) = 0.000861$; $H(P(U)) = 0.0100$

$\bar{u} = 0.0009$.

Source B3: $c = 8$; $P(i) = A \cdot (i + 1)^{-0.73}$, $0 \leq i < c$; $H(P(U)) = 2.8012$

$\bar{u} = 2.3623$.

A normalizes the probabilities.

Source B4: $c = 8$; $P(i) = A \cdot (i + 1)^{-6.45}$, $0 \leq i < c$; $H(P(U)) = 0.1016$

$\bar{u} = 0.01354$.

A normalizes the probabilities.

Source B5: $c = 16$; $P(i) = A \cdot (i + 1)^{-1.29}$, $0 \leq i < c$; $H(P(U)) = 3.0023$

$\bar{u} = 2.7438$.

A normalizes the probabilities.

Source B6: $c = 16$; $P(i) = A \cdot (i + 1)^{-6.46}$, $0 \leq i < c$; $H(P(U)) = 0.1010$

$\bar{u} = 0.01345$.

A normalizes the probabilities.

So for each of the three cardinalities, $c = 2, 8, 16$, we select a high- and a low entropy source. Because the Q-design is sensitive to $\bar{u}$ we reorder the probabilities in the six sources such that $\bar{u}$ is maximal. So we define:

Source B1': $c = 2$; $P(0) = 0.25$, $P(1) = 0.75$; $H(P(U)) = 0.8113$ bits.

$\bar{u} = 0.75$.

Source B2': $c = 2$; $P(0) = 0.000861$, $P(1) = 0.999139$; $H(P(U)) = 0.0100$

$\bar{u} = 0.9991$.

Source B3': $c = 8$; $P(i) = A \cdot (c - i)^{-0.73}$, $0 \leq i < c$; $H(P(U)) = 2.8012$

$\bar{u} = 4.6377$.

A normalizes the probabilities.

Source B4': $c = 8$; $P(i) = A \cdot (c - i)^{-6.45}$, $0 \leq i < c$; $H(P(U)) = 0.1016$

$\bar{u} = 6.9865$.

A normalizes the probabilities.

Source B5': $c = 16$; $P(i) = A \cdot (c - i)^{-1.29}$, $0 \leq i < c$; $H(P(U)) = 3.0023$

$\bar{u} = 12.256$.

A normalizes the probabilities.

Source B6': $c = 16$; $P(i) = A \cdot (c - i)^{-6.46}$, $0 \leq i < c$; $H(P(U)) = 0.1010$

$\bar{u} = 14.987$.

A normalizes the probabilities.

In the Tables 2 we present the lowerbound results for global Q-designs, (3-55), for each of the twelve sources. The Tables 3 and 4 give the upperbound, (3-51) resp. (3-68), for the global Q-designs, Design 2 and Design 6. The next tables list the redundancies resulting from the actual implementation of the Designs 1 to 8 for the twelve sources.

For the global designs, (1, 2, 5, and 6), the redundancy is computed by (3-44) using the stepvalues as given by the designs. In the case of a local design, (3, 4, 7, and 8), the stepvalues depend on the local position. In the tables we list the redundancy computed by (3-44) using the worst set of stepvalues. We find this set by an exhaustive search over all N local positions. So these values are upperbounds to the actual redundancies and we may expect that in most cases the actual redundancy is even lower.

N	k	lowerbound	upperbound Design 1	upperbound Design 5
100	10	.00282	.0128	.0284
91	11	.00141	.0124	.0262
83	12	.000704	.0128	.0262
77	13	.000352	.0133	.0270
56	18	.0000110	.0179	.0357
53	19	.0000055	.0189	.0378
769	13	.000352	.00165	.00366
714	14	.000176	.00158	.00333
667	15	.0000881	.00159	.00326
625	16	.0000440	.00164	.00333
556	18	.0000110	.00181	.00363
500	20	.0000028	.00200	.00401

Table 1. Global P-Design upper- and lowerbounds.

N	k	B1	B2	B3	B4	B5	B6
100	10	.0224	.0256	.0722	.102	.170	.205
91	11	.0217	.0248	.0699	.0990	.164	.198
83	12	.0223	.0255	.0719	.102	.169	.204
77	13	.0233	.0267	.0752	.107	.177	.213
56	18	.0313	.0357	.101	.143	.237	.286
53	19	.0330	.0377	.106	.151	.250	.302
769	13	.00289	.00330	.00932	.0132	.0219	.0264
714	14	.00276	.00315	.00889	.0126	.0209	.0252
667	15	.00278	.00317	.00895	.0127	.0210	.0254
625	16	.00288	.00329	.00927	.0131	.0218	.0263
556	18	.00317	.00362	.0102	.0145	.0240	.0289
500	20	.00350	.00400	.0113	.0160	.0265	.0320

Table 2[a]. Global Q-design lowerbounds.

N	k	B1'	B2'	B3'	B4'	B5'	B6'
100	10	.0160	.0128	.0431	.0130	.0480	.0130
91	11	.0155	.0124	.0417	.0126	.0464	.0126
83	12	.0159	.0128	.0429	.0129	.0477	.0129
77	13	.0167	.0134	.0448	.0135	.0499	.0135
56	18	.0223	.0179	.0601	.0181	.0669	.0181
53	19	.0236	.0189	.0635	.0191	.0706	.0191
769	13	.00207	.00165	.00556	.00167	.00619	.00167
714	14	.00197	.00158	.00530	.00160	.00590	.00160
667	15	.00198	.00159	.00534	.00161	.00594	.00161
625	16	.00206	.00165	.00553	.00167	.00616	.00167
556	18	.00226	.00181	.00608	.00183	.00677	.00183
500	20	.00250	.00200	.00673	.00203	.00750	.00203

Table 2[b]. Global Q-design lowerbounds.

N	k	B1	B2	B3	B4	B5	B6
100	10	.0324	.0356	.0822	.112	.180	.215
91	11	.0327	.0358	.0809	.110	.175	.209
83	12	.0344	.0375	.0839	.114	.181	.216
77	13	.0363	.0397	.0882	.120	.190	.226
56	18	.0491	.0536	.119	.161	.255	.304
53	19	.0519	.0566	.125	.170	.269	.321
769	13	.00419	.00460	.0106	.0145	.0232	.0277
714	14	.00416	.00455	.0103	.0140	.0223	.0266
667	15	.00428	.00467	.0104	.0142	.0225	.0269
625	16	.00448	.00489	.0109	.0147	.0234	.0279
556	18	.00497	.00542	.0120	.0163	.0259	.0307
500	20	.00550	.00600	.0133	.0180	.0285	.0340

Table 3[a]. Global Q-design upperbounds for Design 2.

N	k	B1'	B2'	B3'	B4'	B5'	B6'
100	10	.0260	.0228	.0531	.0230	.0580	.0230
91	11	.0265	.0234	.0527	.0236	.0574	.0236
83	12	.0280	.0248	.0549	.0250	.0598	.0250
77	13	.0297	.0263	.0578	.0265	.0629	.0265
56	18	.0402	.0357	.0779	.0360	.0848	.0360
53	19	.0425	.0378	.0823	.0380	.0895	.0380
769	13	.00337	.00295	.00686	.00298	.00749	.00298
714	14	.00337	.00298	.00670	.00300	.00730	.00300
667	15	.00348	.00309	.00684	.00311	.00744	.00311
625	16	.00366	.00325	.00713	.00327	.00776	.00327
556	18	.00406	.00361	.00788	.00363	.00857	.00363
500	20	.00450	.00400	.00873	.00403	.00950	.00403

Table 3[b]. Global Q-design upperbounds for Design 2.

N	k	B1	B2	B3	B4	B5	B6
100	10	.0754	.0825	.186	.253	.403	.480
91	11	.0706	.0772	.173	.234	.372	.444
83	12	.0714	.0779	.173	.235	.373	.444
77	13	.0740	.0807	.179	.243	.385	.459
56	18	.0983	.107	.237	.321	.510	.607
53	19	.104	.113	.251	.339	.538	.641
769	13	.00971	.0106	.0239	.0325	.0518	.0618
714	14	.00898	.00981	.0219	.0297	.0473	.0564
667	15	.00888	.00970	.0216	.0293	.0464	.0553
625	16	.00912	.00995	.0221	.0299	.0475	.0566
556	18	.00997	.0109	.0241	.0326	.0517	.0617
500	20	.0110	.0120	.0266	.0360	.0571	.0681

Table 4[a]. Global Q-design upperbound for Design 6.

N	k	B1'	B2'	B3'	B4'	B5'	B6'
100	10	.0612	.0541	.121	.0545	.132	.0545
91	11	.0575	.0510	.113	.0514	.123	.0513
83	12	.0583	.0517	.114	.0521	.124	.0521
77	13	.0605	.0537	.118	.0541	.128	.0541
56	18	.0804	.0715	.155	.0720	.170	.0720
53	19	.0849	.0755	.165	.0760	.179	.0760
769	13	.00788	.00697	.0156	.00701	.0170	.00701
714	14	.00732	.00649	.0143	.00653	.0156	.00653
667	15	.00725	.00644	.0141	.00648	.0154	.00648
625	16	.00745	.00662	.0145	.00667	.0158	.00666
556	18	.00816	.00725	.0158	.00730	.0172	.00730
500	20	.00902	.00802	.0175	.00807	.0190	.00807

Table 4[b]. Global Q-design upperbound for Design 6.

N	k	B1=B1'	B2=B2'	B3=B3'	B4=B4'	B5=B5'	B6=B6'
100	10	.00622	.00876	.00954	.0121	.00717	.0123
91	11	.00466	.00974	.00675	.00414	.00747	.00427
83	12	.00800	.0108	.00795	.00620	.00611	.00635
77	13	.00366	.0117	.00553	.00817	.00563	.00818
56	18	.0146	.0166	.00959	.0178	.00903	.000190
53	19	.00476	.0176	.00845	.00105	.00716	.00128
769	13	.00114	.00136	.00100	.000368	.00117	.000484
714	14	.00104	.00156	.00103	.000367	.000554	.000491
667	15	.000566	.000257	.000677	.000153	.000903	.000270
625	16	.00112	.000358	.000880	.00134	.00102	.00147
556	18	.000772	.000557	.000871	.000149	.000943	.000266
500	20	.00122	.000757	.00100	.000148	.000271	.000276

Table 5. The measured global P-method redundancy (Design 1).

N	k	B1	B2	B3	B4	B5	B6
100	10	.0312	.0287	.0788	.112	.174	.212
91	11	.0321	.0317	.0750	.103	.172	.202
83	12	.0291	.0349	.0759	.103	.175	.211
77	13	.0361	.0377	.0810	.112	.181	.216
56	18	.0459	.0523	.110	.160	.246	.303
53	19	.0378	.0553	.115	.152	.257	.303
769	13	.00342	.00396	.00972	.0133	.0228	.0265
714	14	.00350	.00436	.00972	.0129	.0214	.0257
667	15	.00319	.00325	.00985	.0136	.0220	.0257
625	16	.00392	.00356	.0104	.0141	.0224	.0270
556	18	.00392	.00415	.0110	.0145	.0248	.0290
500	20	.00472	.00476	.0123	.0161	.0272	.0322

Table 6[a]. The measured global Q-method redundancy (Design 2).

N	k	B1'	B2'	B3'	B4'	B5'	B6'
100	10	.0187	.0188	.0489	.0223	.0533	.0224
91	11	.0184	.0207	.0472	.0153	.0521	.0154
83	12	.0231	.0229	.0485	.0184	.0527	.0186
77	13	.0199	.0247	.0492	.0213	.0560	.0213
56	18	.0369	.0345	.0696	.0359	.0759	.0183
53	19	.0283	.0365	.0719	.0202	.0778	.0204
769	13	.00277	.00266	.00618	.00169	.00699	.00180
714	14	.00280	.00296	.00618	.00179	.00645	.00191
667	15	.00244	.00176	.00634	.00167	.00686	.00179
625	16	.00312	.00196	.00626	.00296	.00706	.00309
556	18	.00302	.00236	.00692	.00197	.00772	.00209
500	20	.00372	.00276	.00773	.00218	.00821	.00230

Table 6[b]. The measured global Q-method redundancy (Design 2).

N	k	B1	B2	B3	B4	B5	B6
100	10	.00372	.00645	.00200	.00156	.00146	.00157
91	11	.00191	.00714	.000665	.00114	.000772	.00116
83	12	.00197	.00804	.000977	.00132	.000525	.00135
77	13	.000410	.00886	.000405	.00163	.000167	.00167
56	18	.00122	.0133	.0000521	.0000164	.000219	.0000173
53	19	.0000426	.0143	.000537	.0000338	.000288	.0000416
769	13	.000490	.000581	.000250	.000268	.000193	.000270
714	14	.000347	.000150	.0000923	.000153	.0000990	.000155
667	15	.000191	.0000883	.0000481	.0000736	.0000638	.0000750
625	16	.000322	.0000724	.0000257	.0000884	.0000294	.0000989
556	18	.000323	.000103	.0000371	.0000100	.00000653	.0000114
500	20	.000222	.000168	.0000743	.00000316	.00000599	.00000481

Table 7[a]. The measured local P-method redundancy bound for Design 3.

N	k	B1'	B2'	B3'	B4'	B5'	B6'
100	10	.00372	.00645	.00256	.0121	.00349	.0123
91	11	.00191	.00714	.00117	.00412	.00185	.00427
83	12	.00197	.00804	.00107	.00618	.00129	.00635
77	13	.000410	.00886	.000588	.00815	.00101	.00818
56	18	.00122	.0133	.000524	.0178	.00189	.000190
53	19	.0000426	.0143	.00127	.00103	.00129	.00128
769	13	.000490	.000582	.000260	.000359	.000243	.000484
714	14	.000347	.000614	.000140	.000357	.000118	.000491
667	15	.000191	.0000883	.000106	.000146	.0000867	.000269
625	16	.000322	.0000724	.0000680	.00133	.0000625	.00146
556	18	.000323	.000103	.0000729	.000141	.0000438	.000265
500	20	.000222	.000168	.000114	.000139	.0000315	.000275

Table 7[b]. The measured local P-method redundancy bound for Design 3.

N	k	B1	B2	B3	B4	B5	B6
100	10	.00372	.0154	.00836	.0796	.0322	.186
91	11	.00191	.0172	.00868	.0774	.0305	.176
83	12	.00197	.0192	.0106	.0807	.0312	.182
77	13	.000410	.0210	.00896	.0877	.0386	.192
56	18	.00122	.0303	.0153	.126	.0553	.269
53	19	.00476	.0322	.0176	.126	.0584	.277
769	13	.000490	.00133	.000828	.00901	.00121	.0220
714	14	.000347	.00149	.000874	.00833	.00123	.0209
667	15	.000191	.000693	.000746	.00893	.00118	.0209
625	16	.000322	.000795	.000927	.00959	.00137	.0224
556	18	.000323	.00104	.000799	.0101	.00127	.0244
500	20	.000222	.00131	.000938	.0113	.00165	.0273

Table 8[a]. The measured local Q-method redundancy bound for Design 4.

N	k	B1'	B2'	B3'	B4'	B5'	B6'
100	10	.00872	.00877	.00745	.00241	.00774	.00253
91	11	.00740	.00975	.00771	.00442	.00842	.00455
83	12	.00197	.0108	.00574	.00650	.00761	.00665
77	13	.00690	.0118	.00669	.00836	.00478	.00851
56	18	.00569	.0166	.00617	.000432	.00828	.000431
53	19	.00948	.0176	.00957	.00154	.00930	.00154
769	13	.000490	.00136	.00103	.000386	.000791	.000503
714	14	.000347	.000159	.000995	.000387	.000762	.000526
667	15	.000941	.000259	.00105	.000174	.000863	.000307
625	16	.000322	.000359	.000903	.00137	.00125	.00151
556	18	.00122	.000558	.000778	.000196	.000703	.000290
500	20	.000222	.000759	.000547	.000198	.00129	.000303

Table 8[b]. The measured local Q-method redundancy bound for Design 4.

N	k	B1=B1'	B2=B2'	B3=B3'	B4=B4'	B5=B5'	B6=B6'
100	10	.0137	.0188	.0195	.0122	.0172	.0123
91	11	.0129	.0207	.0177	.0151	.0185	.0153
83	12	.0170	.0229	.0200	.0182	.0182	.0184
77	13	.0134	.0247	.0185	.0212	.0186	.0212
56	18	.0280	.0345	.0274	.0180	.0269	.0180
53	19	.0189	.0365	.0273	.0199	.0260	.0201
769	13	.00212	.00266	.00208	.00165	.00244	.00178
714	14	.00210	.00156	.00243	.00177	.00182	.00189
667	15	.00169	.00176	.00218	.00165	.00240	.00177
625	16	.00232	.00196	.00248	.00294	.00262	.00307
556	18	.00212	.00236	.00267	.00195	.00271	.00206
500	20	.00272	.00276	.00300	.00215	.00272	.00228

Table 9. The measured redundancy for Design 5.

N	k	B1	B2	B3	B4	B5	B6
100	10	.0562	.0687	.176	.232	.390	.462
91	11	.0514	.0647	.161	.223	.362	.432
83	12	.0592	.0710	.163	.223	.358	.428
77	13	.0589	.0766	.165	.229	.372	.449
56	18	.0905	.106	.229	.303	.500	.589
53	19	.0849	.112	.240	.321	.526	.623
769	13	.00764	.00916	.0220	.0302	.0506	.0602
714	14	.00700	.00716	.0207	.0283	.0455	.0550
667	15	.00694	.00775	.0200	.0271	.0447	.0527
625	16	.00792	.00835	.0205	.0285	.0462	.0558
556	18	.00842	.00955	.0229	.0307	.0504	.0596
500	20	.00792	.0108	.0256	.0341	.0557	.0662

Table 10[a]. The measured redundancy for Design 6.

N	k	B1'	B2'	B3'	B4'	B5'	B6'
100	10	.0412	.0388	.110	.0326	.118	.0327
91	11	.0404	.0427	.101	.0374	.111	.0375
83	12	.0472	.0470	.104	.0427	.112	.0428
77	13	.0459	.0507	.106	.0475	.117	.0475
56	18	.0727	.0702	.148	.0542	.161	.0542
53	19	.0661	.0743	.154	.0582	.167	.0584
769	13	.00569	.00526	.0136	.00431	.0154	.00444
714	14	.00560	.00436	.0130	.00461	.0136	.00473
667	15	.00544	.00476	.0126	.00469	.0141	.00481
625	16	.00632	.00516	.0132	.00619	.0147	.00631
556	18	.00662	.00596	.0148	.00559	.0162	.00571
500	20	.00772	.00676	.0165	.00620	.0177	.00633

Table 10[b]. The measured redundancy for Design 6.

N	k	B1	B2	B3	B4	B5	B6
100	10	.00372	.00645	.00217	.00156	.00138	.00157
91	11	.00191	.00714	.000971	.00114	.000739	.00116
83	12	.00197	.00804	.000999	.00132	.000487	.00135
77	13	.000410	.00886	.000391	.00163	.000380	.00167
56	18	.00122	.0133	.00160	.0000165	.000281	.0000173
53	19	.0000426	.0143	.000490	.0000338	.000247	.0000416
769	13	.000490	.000581	.000194	.000268	.000234	.000270
714	14	.000347	.000150	.000157	.000153	.000116	.000155
667	15	.000191	.0000883	.0000444	.0000736	.0000534	.0000750
625	16	.000322	.0000724	.0000391	.0000884	.0000281	.0000989
556	18	.000323	.000103	.0000671	.0000100	.0000250	.0000114
500	20	.000222	.000168	.000108	.00000317	.0000116	.00000480

Table 11[a]. The local P-method redundancy for Design 7.

N	k	B1'	B2'	B3'	B4'	B5'	B6'
100	10	.00372	.0154	.00304	.0121	.00617	.0123
91	11	.00191	.0172	.00240	.0151	.00649	.0153
83	12	.00197	.0192	.00242	.0182	.00611	.0184
77	13	.000410	.0210	.00204	.0211	.00635	.0212
56	18	.00122	.0303	.00366	.0180	.0110	.0180
53	19	.00476	.0322	.00338	.0199	.0103	.0201
769	13	.000490	.00133	.000262	.00164	.000363	.00178
714	14	.000347	.000614	.000208	.00175	.000189	.00189
667	15	.000191	.000693	.0000837	.00164	.000203	.00177
625	16	.000322	.000795	.000146	.00293	.000222	.00306
556	18	.000323	.00104	.0000460	.00193	.000227	.00206
500	20	.000222	.00131	.0000843	.00213	.000209	.00228

Table 11[b]. The local P-method redundancy for Design 7.

N	k	B1	B2	B3	B4	B5	B6
100	10	.00372	.0345	.0192	.193	.109	.418
91	11	.00466	.0383	.0190	.175	.0952	.384
83	12	.00499	.0424	.0189	.181	.0957	.386
77	13	.00366	.0461	.0216	.184	.101	.405
56	18	.00569	.0651	.0311	.250	.160	.535
53	19	.00476	.0691	.0392	.265	.173	.567
769	13	.000490	.00326	.000939	.0214	.00342	.0511
714	14	.000347	.00252	.00103	.0199	.00302	.0466
667	15	.000191	.00281	.000911	.0190	.00297	.0449
625	16	.000322	.00313	.00110	.0195	.00304	.0466
556	18	.000323	.00377	.000998	.0219	.00350	.0507
500	20	.000222	.00444	.00133	.0246	.00446	.0566

Table 12[a]. The local Q-method redundancy for Design 8.

N	k	B1'	B2'	B3'	B4'	B5'	B6'
100	10	.00372	.00878	.00949	.00267	.0101	.00280
91	11	.00191	.00977	.00884	.00469	.0113	.00470
83	12	.00499	.0108	.0109	.00668	.00836	.00682
77	13	.000410	.0118	.00885	.00868	.0143	.00884
56	18	.0102	.0167	.0131	.000893	.0194	.000886
53	19	.0142	.0177	.0206	.00202	.0159	.00202
769	13	.000816	.00136	.000959	.000419	.00107	.000537
714	14	.000697	.000162	.000951	.000421	.000811	.000546
667	15	.000191	.000261	.000857	.000212	.000785	.000345
625	16	.000722	.000362	.00101	.00141	.00112	.00155
556	18	.000323	.000562	.000728	.000222	.00114	.000336
500	20	.000722	.000762	.000687	.000250	.00123	.000354

Table 12[b]. The local Q-method redundancy for Design 8.

REFERENCES.

[1] R. G. Gallager, *Information Theory and Reliable Communication*. New York: Wiley, 1968.

[2] D. A. Huffman, "A method for the construction of minimum-redundancy codes," in *Proc. IRE*, vol 40, pp 1098-1101, Sept. 1952.

[3] B. P. Tunstall, "Synthesis of noiseless compression codes," Ph.D. dissertation, Georgia Inst. Tech., Atlanta, GA, Sept. 1967.

[4] F. Jelinek and K. S. Schneider, "On variable-length-to-block coding," *IEEE Trans. Inform. Theory*, vol IT-18, pp 765-774, Nov. 1972.

[5] F. Jelinek, *Probabilistic Information Theory*. New York: McGraw-Hill, 1968.

[6] J. Verhoeff, "A new data compression technique," *Ann. Syst. Res.*, vol 6, pp 139-148, 1977.

[7] J. P. M. Schalkwijk, F. Antonio and R. Petry, "An efficient algorithm for data reduction," in *Proc. Hawaii Int. Conf. System Sciences*, pp 498-499, 1972.

[8] J. P. M. Schalkwijk, "On Petry's extension of a source coding algorithm," in *Proc. 2nd Symp. Inform. Theory Benelux*, pp 99-102, 1981.

[9] J. J. Rissanen, "Generalized Kraft inequality and arithmetic coding," *IBM J. Res. Develop.*, vol 20, pp 198-203, May 1976.

[10] R. C. Pasco, "Source coding algorithms for fast data compression," Ph.D. dissertation, Stanford Univ., CA, 1976.

[11] T. J. Lynch, "Sequence time coding for data compression," in *Proc. IEEE* (Corresp.), vol 54, pp 1490-1491, Oct. 1966.

[12] L. D. Davisson, "Comments on 'Sequence time coding for data compression'," in *Proc. IEEE* (Corresp.), vol 54, pp 2010, Dec. 1966.

[13] J. P. M. Schalkwijk, "An algorithm for source coding," *IEEE Trans. Inform. Theory*, vol IT-18, pp 395-399, May 1972.

[14] T. Cover, "Enumerative source coding," *IEEE Trans. Inform. Theory*, vol IT-19, pp 73-76, Jan. 1973.

[15] J. L. Massey, "The entropy of a rooted tree with probabilities," presented at *IEEE Int. Symp. Inform. Theory*, St. Jovite, Canada, Sept. 1983.

[16] F. R. Gantmacher, *Application of the Theory of Matrices*. New York: Interscience, 1959.

[17] C. B. Jones, "An efficient coding system for long source sequences," *IEEE Trans. Inform. Theory*, vol IT-27, pp 280-291, May 1981.

[18] F. Rubin, "Arithmetic stream coding using fixed precision registers," *IEEE Trans. Inform. Theory*, vol IT-25, pp 672-675, Nov. 1979.

[19] M. Guazzo, "A general minimum-redundancy source-coding algorithm," *IEEE Trans. Inform. Theory*, vol IT-26, pp 15-25, Jan. 1980.

[20] J. J. Rissanen and G. G. Langdon, Jr., "Arithmetic coding," *IBM J. Res. Develop.*, vol 23, pp 149-162, Mar. 1979.

[21] F. Jelinek and G. Longo, "Algorithms for source coding," in: *CISM Courses and Lectures*, no 216, Ed: G. Longo, Wien-New York: Springer, pp 293 - 330, 1975.

[22] J. Rissanen and G. G. Langdon, Jr., "Universal modeling and coding," *IEEE Trans. Inform. Theory*, vol IT-27, pp 12-23, Jan. 1981.

[23] I. Csiszár and J. Körner, *Information Theory: Coding Theorems for Discrete Memoryless Systems*. Budapest: Akademiai Kiado, 1981.

[24] J. L. Massey, "An information theoretic approach to algorithms," in *The Impact of Processing Techniques on Communication*, Ed: J. K. Skwirzynski, NATO ASI Series E, Dordrecht, Boston: Nijhoff Publ., 1985.

[25] D. C. van Voorhis, "Constructing codes with ordered codeword lengths," *IEEE Trans. Inform. Theory*, vol IT-21, pp 105 - 106, Jan 1975.

[26] T. C. Hu and A. C. Tucker, "Optimal computer search trees and variable length alphabetic codes," *SIAM J. Appl. Math.*, vol 21, pp 514 - 532, 1971.

[27] A. M. Garsia and M. L. Wachs, "A new algorithm for minimum cost binary trees," *SIAM J. Comput.*, vol 6, pp 622 - 642, Dec 1977.

[28] J. van Leeuwen, "On the construction of Huffman trees," In: *Proc. 3rd Int. Colloq. Automata, Languages, Programming*, Edinburgh, pp 382 - 410, July 1976.

[29] R. G. Gallager, "Variations on a theme by Huffman," *IEEE Trans Inform. Theory*, vol IT-24, pp 668 - 674, Nov 1978.

[30] O. Johnsen, "On the redundancy of binary Huffman codes," *IEEE Trans. Inform. Theory*, vol IT-26, pp 220 - 222, March 1980.

[31] R. M. Capocelli, R. Giancarlo, and I. J. Taneja, "Bounds on the redundancy of Huffman codes," *IEEE Trans. Inform. Theory*, vol IT-32, pp 854 - 857, Nov 1986.

[32] G. G. Langdon, Jr. and J. Rissanen, "Compression of black-white images with arithmetic coding," *IEEE Trans. Comm.*, vol COMM-29, pp 858-867, June 1981.

EFFICIËNTE EN SNELLE DATA KOMPRESSIE KODES VOOR DISKRETE BRONNEN MET GEHEUGEN.

SAMENVATTING

Dit proefschrift behandelt de kompressie van datarijen, gegenereerd door bronnen met geheugen. Het doel van de data kompressie is het minimaliseren van het aantal kanaal symbolen dat nodig is om het oorspronkelijke bericht exact te beschrijven.

In de Informatie Theorie wordt een bron gezien als een stochastisch proces, meestal met diskrete tijdstappen en een eindige uitkomstenruimte. Van de verschillende mogelijke vormen van de foutvrije kodes worden er hier twee behandeld.

De eerste kode is van het variabele- naar vaste lengte type. Gebruik makend van Schalkwijks enumeratieve kodering is een kode ontworpen voor de klasse van Markov bronnen. Het proefschrift beschrijft deze kode en de analyse hiervan. Bewezen wordt de asymptotische optimaliteit van deze kode en de robuustheid ervan. Dit laatste betekent dat de efficiëntie van de kode niet snel verandert ten gevolge van variaties in de bron parameters. Ook wordt bewezen dat voor de subklasse van de geheugenloze bronnen, d.i. processen met identiek verdeelde, onafhankelijke diskrete stochasten, deze kodes overgaan in de optimale Tunstall kodes. Hiermee is tevens een snelle en laag komplexe kodeer methode gegeven voor deze Tunstall kodes.

De tweede hierin beschreven kode is van het stroom type, d.w.z. dat de rij van kode symbolen stapsgewijs opgebouwd wordt als de opvolgende bron symbolen "voorbij stromen". Het fundamentele algoritme is beschreven door Elias. Door zijn flexibiliteit is deze kode toepasbaar voor de gehele klasse van de stationaire, ergodische bronnen. Het nadeel van dit algoritme

is de onmogelijkheid een praktische implementatie te realiseren. Verschillende auteurs hebben praktische implementaties gevonden en de hier beschreven kode is daarop een voortzetting.

Het proefschrift beoogt inzicht te geven in het Elias algoritme en de praktische implementaties hiervan. Hierbij blijkt dat het koderings mechanisme en het ontwerpen van de kode parameters gescheiden plaats kunnen vinden. Er worden twee koderings mechanismen beschreven welke in een of meerdere opzichten afwijken van de voorgaande oplossingen. Het meest in het oog springend verschil is de eliminatie van de vermenigvuldigingen nodig in de andere algoritmen. Ook de kode parameter ontwerp algoritmen wijken af van de vorige algoritmen. Het blijkt mogelijk en zinvol om lokaal, d.w.z. tijdens het koderen van elk volgend symbool, de parameters te optimaliseren. De analyse van deze kodes resulteert in wederzijds afhankelijke komplexiteits- en redundantie grenzen, die een keuze uit de verschillende kode vormen mogelijk maken, afhankelijk van de behoefte. Ook deze kodes blijken een zelfde robuustheid te vertonen als de in het eerste deel beschreven kode.

CURRICULUM VITAE.

Tjalling Jan Tjalkens was born in Arnhem, The Netherlands, on April 4, 1957. He received his M.S. degree in electrical engineering from the Eindhoven University of Technology, Eindhoven, The Netherlands, in 1983.

Currently he is working at the Department of Electrical Engineering, Eindhoven University of Technology. His research interest is in source coding and modelling.

STELLINGEN

bij het proefschrift van Tj.J. Tjalkens
29 september 1987, Technische Universiteit Eindhoven.

I

Met behulp van Ott's algoritme is een asymptotisch korrekte toestandsschatter voor Markovketens te maken.

G. Ott, "Compact encoding of stationary Markov sources", *IEEE Trans. Inform. Theory*, IT-13,(1),1967.

II

De entropie van een bronmodel is in het algemeen geen maat voor de kompressie van een aan de hand van dit model ontworpen kode. Dit geldt echter wel voor het speciale geval van de optimale k^e ($k < m$) orde benadering van een m^e orde Markov bron.

H. Tanaka et al., "Efficient encoding of sources with memory", *IEEE Trans. Inform. Theory*, IT-25,(2),1979.

J. Rissanen, "A universal data compression system", *IEEE Trans. Inform. Theory*, IT-29,(5),1983.

E.N. Gilbert, "Codes based on inaccurate source probabilities", *IEEE Trans. Inform. Theory*, IT-17,(3),1971.

III

Door in Van Voorhis algoritme P enkel die waarden te berekenen en op te slaan die werkelijk nodig zijn in een gegeven kode ontwerp wordt de hoeveelheid werk aanzienlijk gereduceerd.

D.C. van Voorhis, "Constructing codes with bounded codeword lengths", *IEEE Trans. Inform. Theory*, IT-20,(2),1974.

IV

De beste benadering, met een gegeven komplexiteit, van een m^e orde Markov bron is niet de k^e ($k < m$) orde Markov bron.

J. Rissanen, "A universal data compression system", *IEEE Trans. Inform. Theory*, IT-29,(5),1983.

V

Universele kodes waarbij de kodewoorden direkt uit de data gegenereerd worden, zijn superieur aan algoritmen welke symbool- of rijkansen schatten.

J. Ziv, A. Lempel, "Compression of individual sequences via variable rate coding", *IEEE Trans. Inform. Theory*, IT-24,(5),1978.

P. Elias, "Interval and recency rank source coding: two on-line adaptive variable-length schemes", *IEEE Trans. Inform. Theory*, IT-33,(1),1987.

F.M.J. Willems, "Repetition times and universal data compression", *Proc* 7^{th} *Symp. Inform. Theory Benelux*, Noordwijkerhout, The Netherlands, 1986.

T.M. Cover, "Enumerative source encoding", *IEEE Trans. Inform. Theory*, IT-19,(1),1973.

L.D. Davisson, "Universal noiseless coding", *IEEE Trans. Inform. Theory*, IT-19,(6),1973.

VI

Door gebruik te maken van de samengestelde bron in het ontwerp van een Lawrence-achtige universele bronkode is een lagere redundantie bereikbaar dan die welke de originele kode realiseert in het interessante gebied waar de entropie laag is.

J.C. Lawrence, "A new universal coding scheme for the binary memoryless source", *IEEE Trans. Inform. Theory*, IT-23,(4),1977.

Tj.J. Tjalkens, F.M.J. Willems, "Universal variable- to fixed length source coding for binary memoryless sources", *Proc* 8^{th} *Symp. Inform. Theory Benelux*, Deventer, The Netherlands, 1987.