Non-binary Distributed Arithmetic Coding

by

Ziyang Wang

Thesis submitted to the

Faculty of Graduate and Postdoctoral Studies

In partial fulfillment of the requirements

For the Masc degree in

Electrical and Computer Engineering

Department of Electrical and Computer Engineering

Faculty of Engineering

University of Ottawa

Abstract

Distributed source coding (DSC) is a fundamental concept in information theory. It refers to distributed compression of correlated but geographically separated sources. With the development of wireless sensor networks, DSC has attracted great research interest in the recent years [26].

Although many channel code based DSC schemes have been developed (e.g., those based on turbo codes [11]and LDPC codes [20]), this thesis focuses on the arithmetic coding based approaches, namely, Distributed Arithmetic Coding (DAC) due to its simplicity in encoding [8]. To date, most of the DAC approaches that have been proposed deal with binary sources and can not handle non-binary cases. Little research has been done to extend DAC for non-binary sources. This work aims at developing efficient DAC techniques for the compression of non-binary sources.

The key idea of DAC is representing the source symbols by overlapping intervals, as opposed to the case of conventional arithmetic coding where the intervals representing the symbols do not overlap. However the design of the overlapping intervals has been completely of heuristic nature to date. As such, the first part of this work is a thorough study of various interval-overlapping rules in binary DAC so as to understand how these rules impact the performance of DAC. The insight acquired in this study is used in the second part of this work, where two DAC algorithms are proposed to compress non-binary non-uniform sources. The first algorithm applies a designed overlap structure in DAC process, while the second converts a non-binary sequence into a binary sequence by Huffman Coding and encoding the result in binary DAC. Simulation studies are performed to demonstrate the efficiencies of the two proposed algorithms in a variety of source parameter settings.

Acknowledgements

I would like to thank all the little people who made this possible.

My deepest gratitude goes first and foremost to Professor Yongyi Mao, my supervisor, for his constant encouragement and guidance. He has walked me through all the stages of the writing of this thesis. Without him, I am still a student who is content to the knowledge learning in the course and have no opportunity to get access to the beautiful world of mathematics, and this thesis could not have reached its present form without his help.

Second, I would like to express my gratitude to Professor Iluju Kringa, my cosupervisor, for his guidance on Database topic. He gives me an opportunity to apply the knowledge I learn into a real-life project, showing me how to combine research with applications, and also give me funding to support my study.

Third, I feel grateful to my classmates who often discuss some knowledge with me. In the discussion with them, I can always figure out some concepts that used to be confused to me.

Last but not the least, my thanks goes to my family who support me and give their love and considerations to me all these years, and I also want to thank all the people who helped me and gave me warm when I was abroad alone.

Dedication

This is dedicated to the one I love.

My parents, who support me and love me all these years.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Wireless Sensor Network(WSN) is a network where data streams are transmitted among thousands of sensors, and a server is set up to collect these data streams. Different from other networks, the nodes in a WSN have limited data processing ability and storage capacity. In order to deal with the large amount of data that flows in WSNs, energy-efficient and processing-efficient compression algorithms are critically demanded.

The data compression problem in WSNs is of distributed nature, since the sensory data streams collected by the sensors are correlated and the sensors hardly have the resource or capability to communicate with each other and perform joint compression (namely, compression based on both their own data stream and other sensors' data streams). Such a problem is known as a Distributed Source Coding (DSC) problem, and was first studied by Slepian and Wolf in 1973 [21]. In their seminal work, Slepian and Wolf proved that it is possible for DSC to achieve same compression rates as joint compression. This result was later extended from lossless compression to lossy compression by the proposal of Wyner-Ziv bound [25]. Although the results of Slepian and Wolf and of Wyner and Ziv are mostly of theoretical nature, the development of WSNs has motivated a surge of research interest in the design of practical DSC techniques.

In 2003, Distributed Source Coding Using Syndromes(DISCUS) is proposed in [18]. It allows channel codes such as turbo codes and LDPC codes to be used in DSC. There is

a large body of research in DSC literature based on DISCUS and various channel codes. In particular, non-binary turbo codes [17] and LDPC codes [27] have also been proposed in DSC for non-binary sources. In 2009, Arithmetic Coding is applied in the DSC scheme giving rise to a completely different approach [8]. Distributed Arithmetic Coding(DAC) [8] achieves a good performance on binary source input, especially for short length sequences.

Different from DSC based on channel codes, DAC provides a different approach in DSC. It takes advantage of the statistic prior knowledge of the source, which makes it easy to incorporate with context-based statistical models. As for performance, DAC outperforms turbo codes and LDPC codes on short binary input sequences. The encoding complexity is of great interest in DSC. DAC has a linear complexity encoder, which suits for DSC application. However, DAC can only process binary sources which limits its applications. DAC is still in experimental stage, and it has a lot of potentials. In [4], the research shows that performance of DAC can be improved with a better prior knowledge of the source. To date, the performance of DAC with non-binary source has not been studied yet. In this thesis, we aim to conduct a thorough study on DAC and find out how to improve the performance with different designs. In addition, we proposed two algorithms for DAC to deal with non-binary input in high efficiency and evaluate the performance of these two algorithms through simulations on different input sequences.

The contributions in this work are divided into the following parts:

1. Conduct a thorough study on DAC and compare how different designs affect DAC on the performance.

2. Propose two algorithms regarding to non-binary DAC and evaluate these algorithms through simulations.

3. Conference paper accepted in CWIT 2015.

In this thesis, chapter 2 presents a survey on entropy coding, which includes Huffman Coding, Arithmetic Coding and Lempel-Ziv Coding. Chapter 3 will outline some previous research about channel codes applied in DSC scheme as well as Distributed Arithmetic

Coding. The encoding and decoding process of these algorithms are described in detail. Chapter 4 shows an experiment of how different designs affect the performance of DAC. DAC is designed based on the rule to overlap intervals. We aim to investigate different overlap rules in DAC through simulations. In this chapter, four overlap rules are proposed and the performance is simulated in Binary DAC through various correlations and probability distributions. Chapter 5 describes two proposed algorithms designed for non-binary input in DAC. In [8],binary DAC is an extension of binary Arithmetic Coding which allows the overlap of two intervals between zero and one. The first algorithms proposed is an approach to extend non-binary Arithmetic Coding to non-binary DAC by a designed overlap rule. While the second is an approach to convert non-binary sources into binary sequences by Huffman Coding and compress the binary sequences in binary DAC. The performance is evaluated through simulations on 4-ary and 8-ary input sequences.

Chapter 2

Entropy Coding

Data compression is a process to remove the redundancy in the source and represent it with fewer bits. It is one of the most crucial applications of Information Theory [14]. Data compression can be divided into two parts: one is lossless data compression in which the message can be rebuild exactly at the receiver [14], while the other is lossy data compression which allows certain amount of probability of error and rate distortion. Entropy coding plays an important role in lossless data compression.

2.1 Entropy

Entropy is a concept that is used in many research areas, but the definition mentioned in this thesis is the concept in Information Theory. In Information theory, Shannon gives a lower bound on the lossless compression rate of a random variable, which is the entropy [19] defined as

$$H(X) := -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i) \tag{2.1}$$

$p(\mathbf{x})$ is the distribution of the random variable X. In [19], Shannon proved if a random variable X is compressed into a rate lower than its entropy, it cannot be recovered at the receiver losslessly by any algorithm.

The entropy measures the uncertainty carried by a random variable, which quantifies the expected value of the information contained in the random variable. The larger the entropy is, the larger uncertainty it contains. In the formulation it is clear that $H(X)$ gets the largest value when all $p(x_i)$ are equal. It is also not difficult to figure out this property by definition. For example, when tossing a coin, the result which can be head or tail is taken as a random variable. The entropy of this variable is the average number of binary questions that are needed to ask in order to get the right result. If the coin is unbias in which situation the probability of head and tail are both 0.5, the entropy equals to 1. We only need one binary question to figure out the right output in this situation. However, in the case that the coin is bias, we will know which result is more likely to be and we will ask a less average number binary questions to find the correct answer.

Shannon proves that the entropy of a random variable is the lower bound that the rate can achieve in lossless compression. If the rate at the transmitter side is above the entropy, all the information can be recovered at the receiver side losslessly without any side information. Otherwise, the information can not be losslessly recovered by any algorithms without any side information. Therefore, when designing a lossless compression algorithm in WSNs, the entropy plays an important role.

2.2 Source Coding

Coding theory is one of the crucial part in Information Theory [19]. It can be divided into source coding and channel coding. The source coding is related to data compression problem while the channel coding is designed to protect the message from the channel noise.

Source coding is a process to compress the input signal into a smaller binary sequence. It is a process to remove the redundancy in the input and use the most concise bits to represent it. Source coding can be divided into two parts: the first is lossless source coding which allows the information to be rebuilt precisely at the receiver; the second is lossy source coding which admits some probability of error and rate distortion.

Plenty of algorithms are developed on the source coding scheme. Entropy is a crucial limit for the rate in the lossless source coding while the probability of error and rate distortion are of great interest in the lossy source coding. The lossless source coding which can achieve capacity is also called entropy coding. In this section, Huffman coding, Arithmetic coding and Lempel-Ziv coding are introduced.

2.2.1 Huffman Coding

Huffman coding is an algorithm developed in 1952 [13]. It is a process to map a set of symbols into a Huffman code which is an optimal prefix code (If there is no codeword that is a prefix of other codeword in a codebook, this code is called prefix code [13]).

The basic rule for all compression algorithms is to assign the smaller length codewords to the symbols with higher probabilities and the larger length codewords to the symbols which are rarely to be seen. In Huffman coding, a codebook is derived through setting up a binary tree to identify representations for symbols in an alphabet A.

Encoding Process

A discrete random variable X takes values from an alphabet $A = \{a_1, a_2....a_n\}$ with a distribution $P = \{p_1, p_2...p_n\}$. An i.i.d X sequence is compressed in Huffman coding, the technique is conducted in the following steps:

- The symbols in alphabet A are sorted by their weight (probability) so that the symbol with the largest probability is put in the top while the symbol with the smallest

probability is put in the bottom.

- All the symbols in A are taken as leaf nodes and the last two symbols are combined to the next level.
- Resort the symbol weight and keep combining the last two symbols until only two symbols are left.
- When the tree is set up, from the first level to the leaf nodes, assign each path a bit 0 or 1. It is clear all the symbols in A are located in a node of the tree, we use the paths from the first level to the node of a symbol as the representation for a symbol.

It can be proved the code is an optimal prefix code in such a tree structure mapping, and the average length of the codeword $\mathbf{c}$ which is calculated by:

$$L = \sum_{i=1}^{q} p_i L_i \tag{2.2}$$

is approaching the entropy of the input random variable.

Decoding Process

The decoding process is generally to rebuild every symbol by looking up the codebook derived in the encoding stage. No segments are required in the codeword since Huffman coding is a prefix code. There is no ambiguity in decoding process, and the original sequence can be rebuilt exactly.

Example

The example below shows the detailed process of Huffman coding. The distribution for a discrete random variable X is shown in Table 2.1.

Symbols	A	B	C	D	E	F	G	H	I
p	0.36	0.15	0.13	0.11	0.09	0.07	0.05	0.03	0.01

Table 2.1: Distribution of source input X

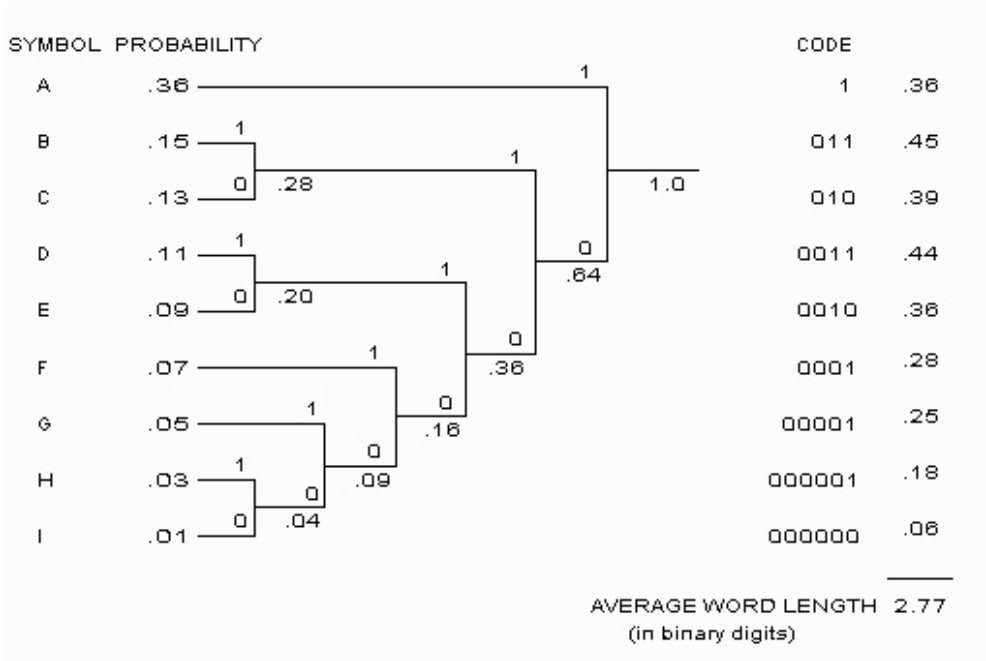

Figure 2.1: Encoding process of Huffman Coding

The Huffman coding encoding process is in the Fig 2.1.

The codebook for the input X in Huffman coding can be generally derived by the encoding tree in Fig 2.1. It is as shown in Table 2.2.

The average length of this Huffman code is

$$L = \sum_{i=1}^{q} p_i L_i = 2.77$$

and the entropy of X can be calculated by

$$H(X) = -\sum_{i=1}^{q} p(x_i) \log_2 p(x_i) = 2.6896$$

The average length is about one bit away from the entropy. Huffman coding has the ability to achieve capacity in certain situations.

Symbol	Codeword
A	1
B	011
C	010
D	0011
E	0010
F	0001
G	00001
H	000001
I	000000

Table 2.2: Codebook in Huffman Coding

Although Huffman coding is an optimal code which can achieve the entropy of the random variable, it also has its limitations. First, the distribution of the random variable is needed, and also Huffman Coding can only achieve capacity for certain designed distributions of input sequences. Compared to Huffman Coding, Lempel-Ziv Coding is superior because it does not need a distribution first. Arithmetic Coding has a larger capacity for the symbols input.

2.2.2 Arithmetic Coding

Arithmetic coding is another entropy coding technique which employs the idea of mapping a symbol sequence into a real number between zero and one based on the probability distribution of the source [24], and the real number is converted into a binary sequence as the codeword. It is proved that if the length of the binary sequence equals to $\lceil -\log_2 p(\mathbf{x}) \rceil$, the receiver is able to rebuild the message exactly without any side information, $p(\mathbf{x})$ is

the probability of the symbol sequence.

Encoding Process

Strictly speaking, Arithmetic coding encoding process can be divided into two stages: the partitioning stage and the encoding stage. In the partitioning stage, the interval $[0, 1]$ is divided into several subintervals based on the probability distribution, and the symbols are assigned to intervals uniquely. In the encoding stage, the input data is encoded symbol by symbol in the model and the final interval is converted into a certain length binary sequence.

For example, a discrete random variable X takes values from a q-ary alphabet $A = \{a_1, a_2...a_q\}$ with a probability distribution $p(x)$. In the partitioning stage, the interval [0,1] is divided into q subintervals based on the probability distribution function. Symbols in alphabet A are assigned to intervals characterized by low and high defined as:

$$l_i = \sum_{k=1}^{i-1} p(a_k) \tag{2.3}$$

$$h_i = \sum_{k=1}^{i} p(a_k) \tag{2.4}$$

for i from 1 to q, every symbol is assigned to a unique interval. All the intervals fill the interval from 0 to 1. The structure is as shown in Fig 2.2.

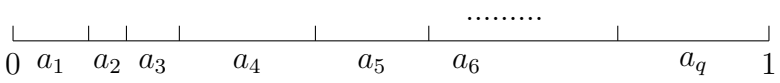

Figure 2.2: The model for Arithmetic coding

The encoding process of Arithmetic coding is to choose the interval recursively until all

the symbols in the input sequence are encoded. In detail, a finite length N sequence of X characterized by $p(x)$ is encoded in Arithmetic coding. $x_1, x_2...x_N$ are encoded symbol by symbol in the algorithm described below:

- Initialize the interval I_0 with $I_0 = [0, 1]$.
- Choose the interval that is assigned to x_1, which is I_1 characterized by $l(x_1)$ and $h(x_1)$.
- Divide the interval I_1 into q subintervals with the same structure as that in the model, and choose the interval that is assigned to x_2 which results in a interval I_2 characterized by $l(x_1, x_2)$ and $h(x_1, x_2)$ as

$$l(x_1, x_2) = l(x_1) + l(x_2)p(x_1) \tag{2.5}$$

$$h(x_1, x_2) = l(x_1) + h(x_2)p(x_1) \tag{2.6}$$

- Conduct step 3 until all the symbols are encoded. The final interval I_N is characterized by $h(x_1, x_2...x_N)$ and $h(x_1, x_2...x_N)$ which can be calculated in a recursive function :

$$l(x_1, x_2...x_N) = l(x_1, x_2...x_{N-1}) + l(x_N)p(x_1, x_2...x_{N-1}) \tag{2.7}$$

$$h(x_1, x_2...x_N) = l(x_1, x_2...x_{N-1}) + h(x_N)p(x_1, x_2...x_{N-1}) \tag{2.8}$$

- Convert the final interval I_N into a length L binary sequence, L is defined as

$$L := \lceil -\log_2 p(x_1, x_2...x_N) \rceil \tag{2.9}$$

The length L binary sequence is transmitted to the receiver as the codeword. It is proved that the input sequence can be rebuild exactly by the receiver in this set up.

Decoding Process

In order to recover the original message, the decoder in the Arithmetic coding needs to simulate the behavior of the encoder. The codeword is converted into a real number Z between 0 and 1. The interval from 0 to 1 is partitioned into q subintervals which is the same as that in the encoding process. The symbol is decoded as the one with the interval that covers the real number Z. The partitioning process is conducted recursively so that all symbols from x_1 to x_N are decoded. The algorithm is described below:

- The binary input sequence $c_1c_2...c_k$ is converted into a real number Z and $Z \in [0, 1]$.
- Initialize the interval $[0, 1]$ the same as that in the encoding process, and find out which interval Z belongs to. The symbol that is assigned to the interval is decoded as x_1.
- Divide the interval of x_1 into several subintervals the same as that is in encoding process, and find out which interval Z belongs to. The symbol that is assigned to the interval is decoded as x_2.
- The process is conducted recursively until all N symbols are decoded. It is proved that the result is exactly the same as encoding sequence in this decoding process [12].

A termination policy is applied for a various length sequence in Arithmetic Coding in practical implementation. A termination symbol is often added to the alphabet. In the partitioning stage, the termination symbol is assigned to a small interval, and it is encoded after all symbols in the sequence are encoded in the encoding process. In decoding process, the decoding process stops directly whenever Z is falling into the termination interval. The current symbol sequence is the result of the decoding process.

Example

A simple example illustrating the process of Arithmetic coding is shown in Fig 2.3. A discrete random variable X is taking values from an Alphabet $A = \{0, 1, 2, 3\}$ with a distribution listed in Table 2.3.

X	0	1	2	3
p	0.6	0.2	0.1	0.1

Table 2.3: Distribution of the input sequence X

The input sequence 023 is encoded in Arithmetic coding, and the encoding process is shown in Fig 2.3. The final interval after the encoding process is [0.534,0.54], and it is converted into a length $L = -\lceil \log_2 p_0 p_2 p_3 \rceil = 8$ bits binary sequence. The codeword is 10001001.

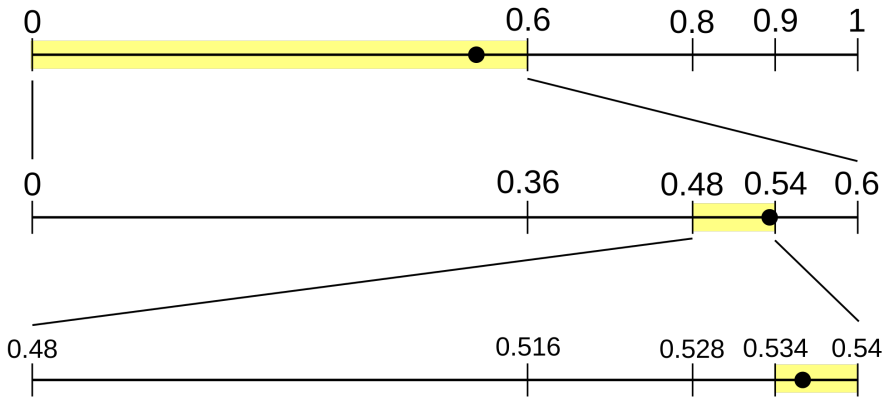

Figure 2.3: Example of Arithmetic coding

In the decoding process, the codeword 10001001 is converted into a real number Z at the receiver and $Z = 0.5352$. In the first step, Z is located in the interval [0,0.6]. In the second step, Z is located in the interval [0.48,0.54]. In the third step, Z is located in the interval [0.534,0.54]. The input sequence is recovered by the receiver.

Arithmetic coding is a mature optimal entropy coding. It has a larger capacity to handle some sources which the rate of Huffman coding can not achieve entropy, and it is superior in short sequence because it can achieve entropy at any length of a sequence when compared with Lempel-Ziv coding.

2.2.3 Lempel-Ziv Coding

Lempel-Ziv coding is developed by Jacob Ziv and Abraham Lempel in 1977 [29] [30]. It is a universal algorithm for lossless sequential data compression. It is universal because it can handle any source input and it does not need the prior probabilistic property to start. It can achieve capacity when the sequence is long enough in certain situation.

The process for Lempel-Ziv coding is to set up a dictionary in which the symbol in a finite alphabet and its codeword are solely mapped. The dictionary is empty at the beginning of the process and it is constantly updated during the encoding process by assigning a codeword to a new symbol. Keep The rate in this process can achieve capacity when the dictionary is complete. Theoretically, the process of setting up the dictionary is to learn the distribution of the source, and when the dictionary is complete, it means the distribution is fully learned so that the most frequently appeared sequences are assigned with the shortest codewords and the sequences rarely to be seen are assigned with the longest codewords.

Lempel-Ziv coding is a widely used compression algorithm because it is simple to implement and its performance is among the best for long sequential data compression. In wireless sensor networks, Lempel-Ziv coding is also a good choice for data compression. However, the efficient is really low for small and medium length sequences.

Chapter 3

Distritbuted Source Coding

Distributed Source Coding (DSC) is an important concept in Information Theory. It is potentially designed for Wireless Sensor Networks(WSNs) [26]. In DSC, multiple correlated sources that do not communicate with each other are encoded separately, and the information is collected by a joint decoder to recover the original messages. Algorithms in the DSC scheme are in favor of those with low complexity in encoding process and high efficiency. In 1973, the proposition of the Slepian-Wolf Coding gives a theoretic bound on lossless data compression in the DSC scheme [21]. However, no effective algorithms can be applied into practical techniques at that time. The design of Distributed Source Coding Using Syndromes (DISCUS) enables the practical work regarding to DSC scheme in 2003 [18], and a lot of channel codes are applied into WSNs to compress image and video streams such as Turbo codes and LDPC codes. Distributed Arithmetic Coding(DAC) is proposed in 2009 [8], and it shows great performance on short length sequences.

3.1 The Slepian-Wolf Theorem

David Slepian and Jack Wolf proved a theoretically bound in the DSC scheme in 1973, and it is known as the Slepian-Wolf bound [22] [3] [21]. The Slepian-Wolf Theorem states that two correlated sources which do not communicate with each other can be compressed into a rate the same as when there is communication between them, and the message can be recovered losslessly at the receiver theoretically [21]. It gives a lower bound in DSC

regarding to lossless data compression. Theoretically, sources can be compressed into this bound with arbitrary small error in certain conditions.

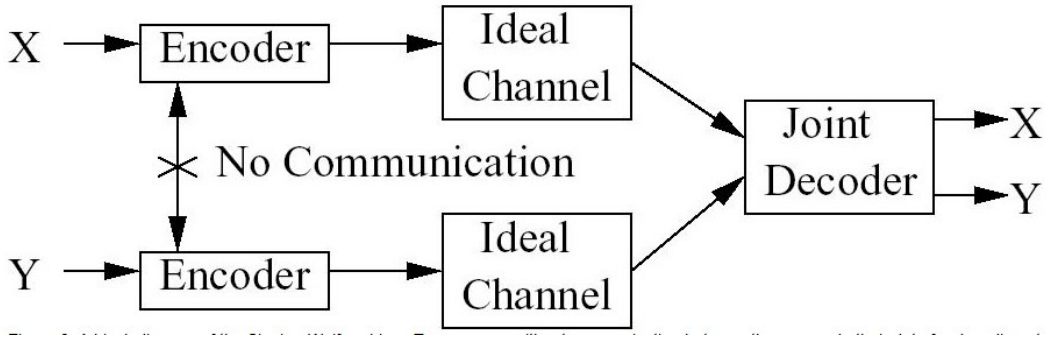

Figure 3.1: Distributed source coding

In Fig 3.1, X and Y are two correlated random variables. If they are encoded and decoded separately, the lowest rate that X and Y can achieve in lossless data compression are $R_X = H(X)$ and $R_Y = H(Y)$ so that the total rate of the two sources is $R = H(X) + H(Y)$. However, it is possible for the total rate R to achieve the a smaller rate that equals to their joint entropy $H(X, Y)$ when there is communication between the two sources [21], and it shows that

$$H(X, Y) \leq H(X) + H(Y) \tag{3.1}$$

The Slepian-Wof Bound is shown in Fig 3.2, which satisfies the property

$$R_X \geq H(X|Y) \tag{3.2}$$

$$R_Y \geq H(Y|X) \tag{3.3}$$

$$R_X + R_Y \geq H(X, Y) \tag{3.4}$$

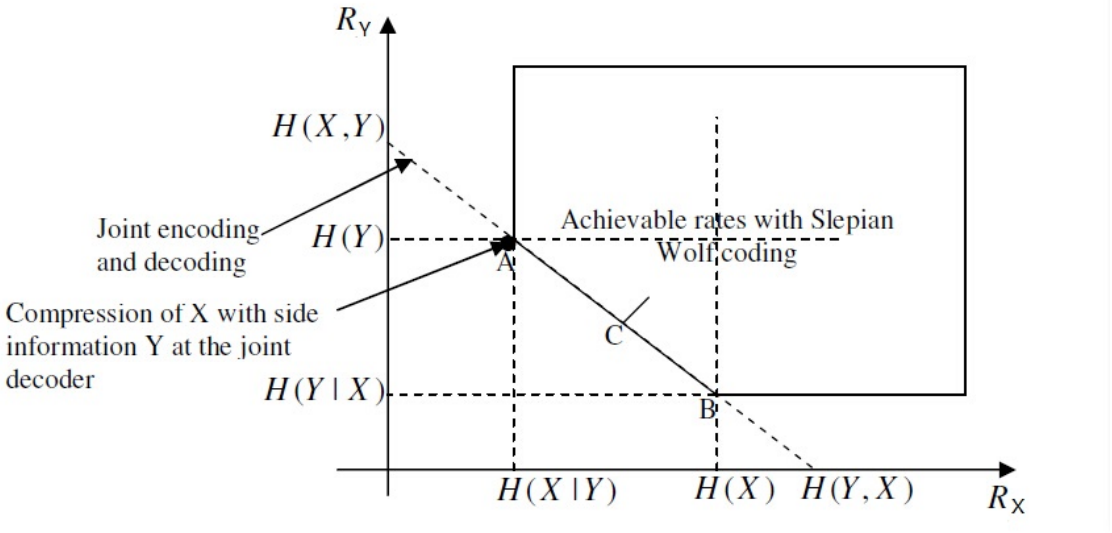

Figure 3.2: Splepian-Wolf Bound

In asymmetric case, random variable Y is encoded into its entropy $H(Y)$, and it is decoded without any ambiguousness at the decoder. While random variable X is compressed into a conditional entropy $H(X|Y)$. The decoder is not able to rebuild X without ambiguity since the rate R_X is lower than its entropy$H(X)$. The decoder employs the message from random variable Y as the side information to decode X based on the correlation between X and Y.

In symmetric case, X and Y are encoded into a rate lower than their entropy while the sum of R_X and R_Y is in the Slepian-Wolf region. The joint decoder record all possible configurations for X and Y and choose a most reliable configuration based on the correlation between them. It is proved that an arbitrary small error probability can be achieved if the input sequence is long enough in certain conditions.

3.2 Distributed Source Codes

Distributed Source Coding Using Syndromes(DISCUS) is proposed in [18] in 2003, and it is a breakthrough in the DSC scheme. A lot of researches ignited by DISCUS have offered algorithms with channel codes being applied into the DSC scheme. The performances of Turbo codes and LDPC codes in DSC are fully investigated, and they have been already applied into practical compression work of distributed video and image streams.

3.2.1 Distributed Source Coding Using Syndromes(DISCUS)

Distributed Source Coding Using Syndromes(DISCUS) is proposed in 2003 [18]. The idea is to partition the vector space of input source to minimize the probability of decode error at the receiver.

In DISCUS, the vector space of a length n input sequence is partitioned by a (n, k) channel code. The encoder transmits the index of syndrome of the codeword instead of the codeword itself. The decoder can rebuild the codeword in the given coset with side information. The compression rate is

$$R = \frac{n-k}{n} \tag{3.5}$$

. The structure of DISCUS is shown in Fig 3.3.

The most crucial challenge in DISCUS is how to design a (n, k) channel code to minimize the probability of error at the decoder with high efficiency. Several powerful channel codes are investigated. Turbo codes and LDPC codes which have low complexity decoding process and high efficiency are of great interest by researchers .

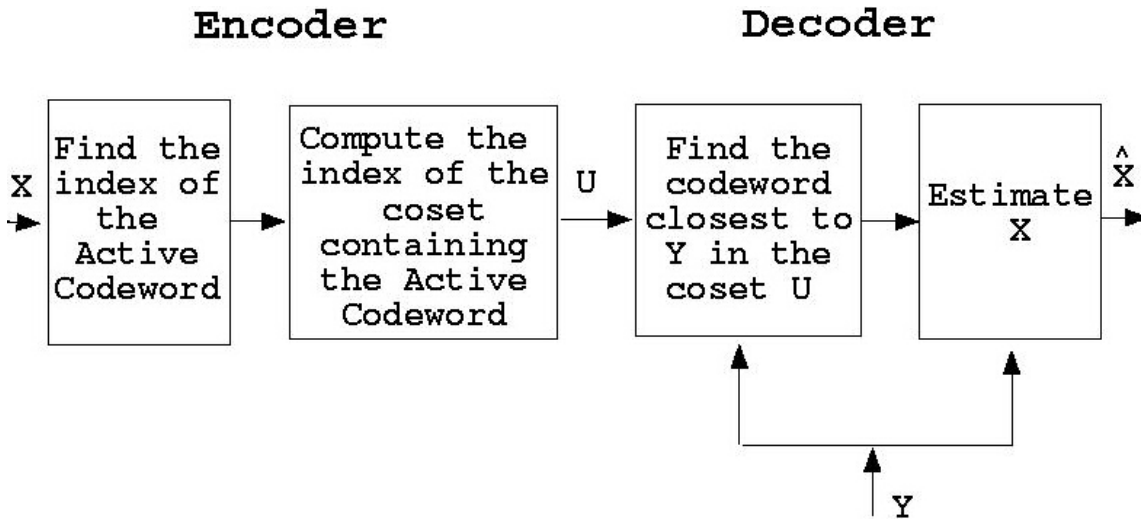

Figure 3.3: Design of DISCUS

3.2.2 Distributed Source Coding Using Turbo Code

Turbo code is a powerful error correction code in channel coding [11]. Its high performance draws a lot research attention in developing related algorithms using turbo codes.

The turbo code is also applied in the DSC scheme [1] [2] [5], and a punctured turbo codes is used to encode the source. Two turbo decoders decode the source with exchange in information to achieve a small probability of error.

It has already been applied in practical applications. Some video streams transmitted in WSNs are using this technique [6] [10].

3.2.3 Distributed Source Coding Using LDPC Code

Low-density-parity-check(LDPC) code is a linear error correction code which has a beautiful structure in mathematics [20]. Given the parity check matrix, the LDPC code can be generated through factor graph [15].

LDPC code has been applied to the DSC scheme. In encoding process, a length n input source is quantized into a codeword of a designed LDPC code, and the encoder transmits the syndrome that this codeword belongs to instead of the index of the codeword. The rate R achieves $\frac{n-k}{n}$. The decoder recovers the sequence by choosing the most reliable

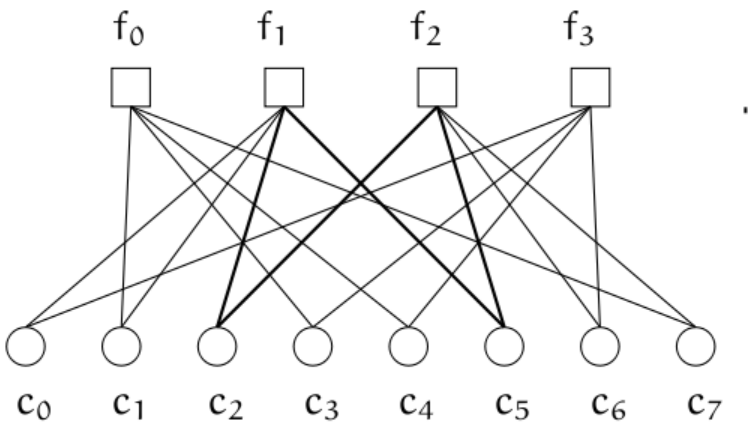

Figure 3.4: Factor graph

codeword in this syndrome with side information Y, which is a correlated sequence of the source input. The decoding process is conducted in BP algorithm [15].

The high performances of LDPC codes and turbo codes are inspiring researchers to put more attention on this subject, and non-binary LDPC code and turbo code are also simulated for non-binary input in DSC [23] [16] [17].

3.3 Distritbuted Arithmetic Coding

Distributed Arithmetic Coding was first fully developed by Marco Grangetto and Enrico Magli in 2009 [8]. It is an extension of Arithmetic Coding in the DSC scheme.

As mentioned above, the rate for Arithmetic Coding is calculated in

$$R = \sum_{i=1}^{q} p_i L_i = -\sum_{i=1}^{q} p_i \lceil \log p_i \rceil \tag{3.6}$$

As p_i is a intrinsic property of the input so that it can not be changed. By intuition, the only method to lower the rate is to reduce L_i. L_i is in a form

$$L_i = -\log p_i \tag{3.7}$$

It is possible to reduce L_i by enlarging the size of interval that is assigned to the symbol p_i. Marco Grangetto proposed a way to enlarge the intervals and make them overlap with other intervals, and this algorithm can achieve a lower rate.

3.3.1 Binary Distributed Arithmetic Coding

A discrete random variable X takes values from a binary alphabet $A = \{0, 1\}$ with a distribution $p(X = 0) = p_0, p(X = 1) = p_1$. Y is another random variable taking values from a binary alphabet A with correlation with X as $p(X = Y) = 1 - \epsilon$. X and Y are compressed by Distributed Arithmetic Coding, and the rate constraint for X and Y are

$$R_X + R_Y \geq H(X, Y) \tag{3.8}$$

In the thesis, we assume Y is encoded and decoded in the rate $R_Y = H(Y)$, while X is encoded and decoded in Distributed Arithmetic Coding. The rate of X is changing from $H(X|Y)$ to $H(X)$.

The partitioning stage

The structure of the the partitioning stage and the encoding stage is shown in Fig 3.5.

In traditional binary Arithmetic Coding, the interval from 0 to 1 is divided into two subintervals I_0 and I_1 with size $I_0 = p_0$ and $I_1 = p_1$. In Distributed Arithmetic Coding, the two intervals I_0 and I_1 are enlarged to size $\tilde{I}_0 = q_0$ and $\tilde{I}_1 = q_1$ in order to achieve a lower rate. The enlarged intervals $\tilde{I}_0$ and $\tilde{I}_1$ overlap with each other in the structure in Fig

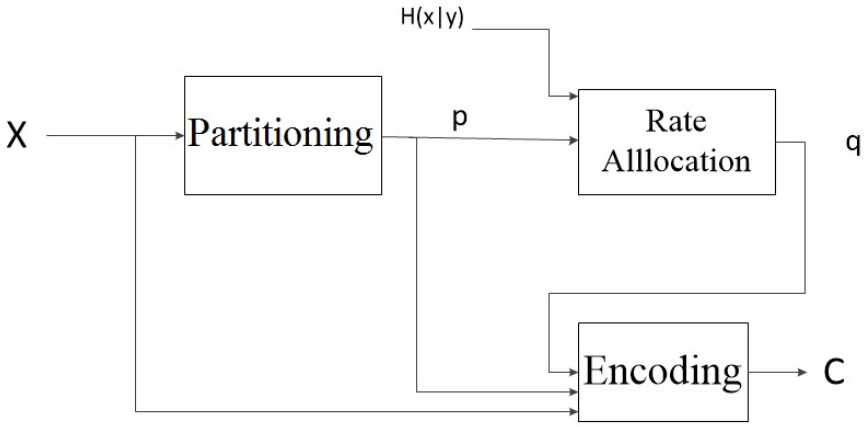

Figure 3.5: Structure of the partitioning stage and the encoding stage

3.6.

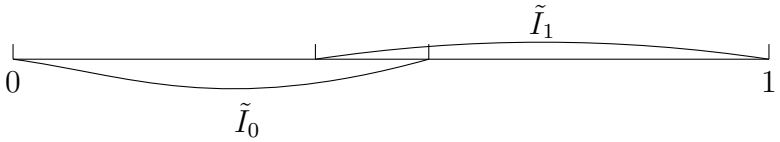

Figure 3.6: Structure of overlapped intervals

A certain ambiguity is added to the algorithm because of the existence of the overlapped region. The side information is a necessary component in the decoding process. The modified interval size q_0 and q_1 are functions of p_0 and p_1 and they are constrained by the rate R_X.

DAC encoder

The encoding process of DAC is similar to that of Arithmetic Coding. It is conducted by choosing the interval and partition it symbol by symbol, and the final interval is converted into a binary sequence with certain length after all the symbols are encoded.

A length N sequence of binary discrete random variable X characterized by probability

distribution $p(X = 0) = p_0$ is the input at the DAC encoder. The encoding process is described below:

- Initialize the interval from 0 to 1 I_0 by dividing it into two overlapped subintervals $\tilde{I}_0$ and $\tilde{I}_1$ in Fig 3.3.
- Choose the first interval $\tilde{I}(x_1)$ characterized by $l(x_1)$ and $h(x_1)$ which can be calculated by

$$l(x_1) = \sum_{b<x_1} q(b) \tag{3.9}$$

$$h(x_1) = l(x_1) + q(x_1) \tag{3.10}$$

  then divide this interval into two subintervals which has the same structure.

- Choose the second interval $\tilde{I}(x_1, x_2)$ characterized by $l(x_1, x_2)$ and $h(x_1, x_2)$ which can be calculated by

$$l(x_1, x_2) = l(x_1) + l(x_2)q(x_1) \tag{3.11}$$

$$h(x_1, x_2) = l(x_1) + h(x_2)q(x_1) \tag{3.12}$$

  then divide this interval into two subintervals with the structure.

- Keep conducting this procedure until all N symbols are encoded and the iterative function is in a formulation :

$$l(x_1, x_2...x_n) = l(x_1, x_2...x_{n-1}) + l(x_n)q(x_1, x_2...x_{n-1}) \tag{3.13}$$

$$h(x_1, x_2...x_n) = l(x_1, x_2...x_{n-1}) + h(x_n)q(x_1, x_2...x_{n-1}) \tag{3.14}$$

- Choose a point in the final interval $\tilde{I}(x_1, x_2...x_N)$ and convert it to a certain length binary sequence $\mathbf{c}$. The length of sequence $\mathbf{c}$ is computed by

$$L = -\log q(x_1, x_2...x_N) = \sum_{i=1}^{N} \log q(x_i) \tag{3.15}$$

  $\mathbf{c}$ is transmitted to the receiver.

In the encoding process, the compression rate R is calculated by

$$R = \sum p(x_i)L(x_i) = -\sum_{i=1}^{q} p(x_i) \log_2 q(x_i) \tag{3.16}$$

DAC decoder

The decoding process of DAC is to create a decoding tree to track all the possible sequences which are able to be encoded to the codeword received. A most reliable sequence is selected by applying the side information.

Two correlated binary discrete random variables X and Y are encoded separately at two different encoders. X is encoded in DAC at a rate $R_X \in [H(X|Y), H(X)]$ while Y is encoded in a lossless source coding at a rate $R_Y = H(Y)$. Y is observed at the decoder and the objective is to rebuild the original X sequence with side information Y.

The decoding process is described below:

- The codedword of X $\mathbf{c}$ is converted into a real number Z so that $Z \in [0, 1]$.
- Initialize the interval from 0 to 1 to the structure in Fig 3.4. The decoding process is conducted symbol by symbol.
- Find out the interval that Z is located in and force the decoding process to decode the symbol. The decoding tree splits two paths to record the two symbols when Z is located in the overlap region.

- All the possible sequences are listed in the decoding tree when all symbols are decoded. The side information Y is employed to recognize the most reliable path.

The pseudo code of the decoding process is listed in Algorithm 1:

- T_i denotes the ith level of the decoding tree. n_i denotes a node in the ith level of the tree and x_i denotes the decode symbol of n_i.
- $x_{i+1} = Test(T_i)$. It computes the possible symbols for the ith level of the tree. The function finds out the intervals that cover the real number of the codeword and give all the possible x_i.
- $n_{i+1} = decode(x_{i+1})$. It stores the information of the decoding symbol sequences in a node of the decoding tree. The function updates the nodes in the ith level of the tree in the decoder.

A metrics is employed to figure out the true path after all symbols are decoded. The metrics is provided by the side information Y. The optimal sequence $(\hat{x}_1, \hat{x}_2 ... \hat{x}_n)$ satisfies the equation

$$\begin{aligned} (\hat{x}_1, \hat{x}_2 ... \hat{x}_n) &= \arg\max_{(x_1, x_2 ... x_n)} p(x_1, x_2 ... x_n | \mathbf{c}, y_1, y_2 ... y_n) \\ &= \arg\max_{(x_1, x_2 ... x_n)} \prod_{i=1}^{n} p(x_i|y_i) p(\mathbf{c}|x_1, x_2 ... x_n) \end{aligned} \tag{3.17}$$

$\mathbf{c}$ is the codeword received and $p(x_i|y_i)$ is the weight of each x_i by

$$p(x_i|y_i) = \begin{cases} 1 - \epsilon & x_i = y_i \\ \epsilon & x_i \neq y_i \end{cases} \tag{3.18}$$

The function $p(\mathbf{c}|x_1, x_2 ... x_n)$ is defined as

Algorithm 1 DAC decoding process [8]

Initialize the root node T_0

while $i < N$ **do**

for All nodes in T_i **do**

$x_{i+1} = Test(T_i)$

if x_{i+1} in overlap region **then**

for $x_{i+1} = (0, 1)$ **do**

$n_{i+1} = decode(x_{i+1})$

$T_{i+1} = T_{i+1} + n_{i+1}$

end for

else

$n_{i+1} = decode(x_{i+1})$

$T_{i+1} = T_{i+1} + n_{i+1}$

end if

end for

Sort the nodes in T_{i+1} by side information

Keep M most reliable nodes in every T_{i+1}

end while

$$p(\mathbf{c}|x_1, x_2...x_n) = \begin{cases} 1 & (\mathbf{c} = ENC(x_1, x_2...x_n)) \\ 0 & (\text{otherwise}) \end{cases} \tag{3.19}$$

It denotes all the possible sequences that can be encoded into the codeword $\mathbf{c}$.

The weight of each path of the tree is calcuted by

$$w(x_1, x_2...x_N) = \prod_{i=1}^{N} p(x_i|y_i) \tag{3.20}$$

The sequence with the largest weight is selected as the decoding result.

M algorithm and Termination Policy

The complexity is one of the most concerning problems for the tree structure since the number of leaf nodes grows exponentially in each level of the tree. Therefore, the complexity of the DAC deocding process explodes when the compression rate is low or the length of the input sequence is large.

A smart way to deal with the situation is to sacrifice some accuracy in order to reduce the complexity. The maximum number of the leaf nodes is set as M. If the number of the leaf nodes is larger than M, the tree is pruned by throwing away the least likely paths. M algorithm may loss some accuracy because the true path may be abandoned in the early stage. In the simulation, M is set as 2048, which provides a balance between accuracy and complexity.

It is found that the last T symbols are more likely to be decoded incorrectly. Based on the Hidden Markov Chain structre of DAC decoding process, the symbols behind are capable of correcting the incorrect symbols decoded earlier. Therefore, the ambiguity in last T symbols are larger and they are more likely to be decoded incorrectly. A termination policy is proposed in [8]. In the simulation, the last T symbols are encoded in traditional Arithmetic Coding without overlaps, and the other symbols are encoded in a lower rate so that the average rate of the sequence remains the same.

Performance Evaluation

A sequence of binary discrete random variable X characterized by $p(x = 0) = 0.5$ is encoded and decoded in binary DAC. The rate R_X is in Slepian-Wolf region that $H(X|Y) \leq R_X \leq H(X)$. The side information Y is the result of X going through a binary symmetric channel with crossover probability ϵ.

The simulation is conducted for various sequence lengths N and different correlations between source input and side information. The sequence length N is set as $N = 200$ and

$N = 1000$. The correlation parameter $\epsilon = 0.05$ for strongly correlated sources while $\epsilon = 0.2$ for weakly correlated sources. M algorithm and termination policy are applied, $M = 2048$ and $T = 15$. The simulation is conducted myself as a reproduce of the result in [8]. The simulation result is shown in Fig 3.7 and Fig 3.8. The label for X axis is the rate R_X, and the label for Y axis is the probability of symbol error.

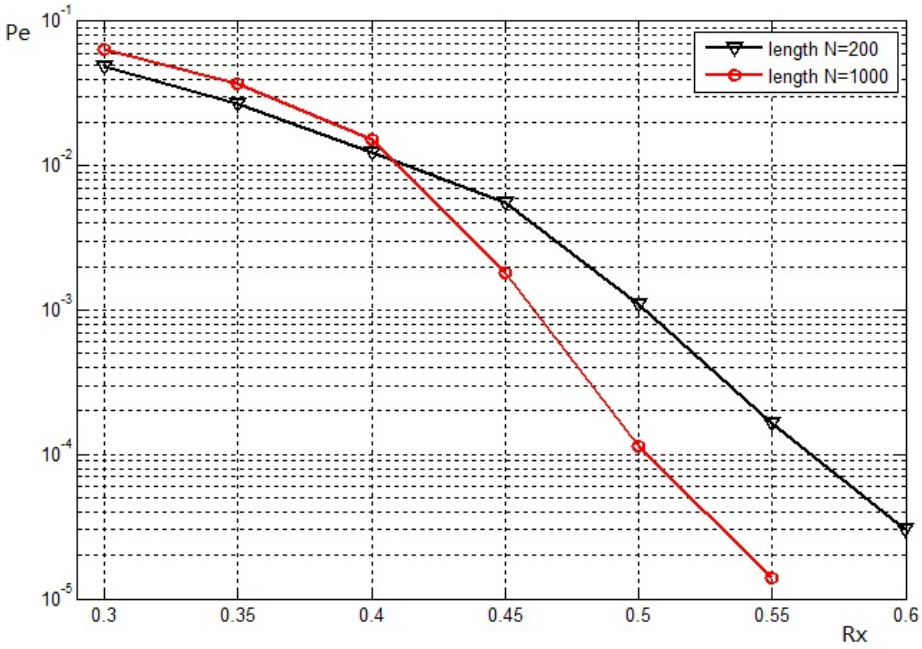

Figure 3.7: Error probability for strongly correlated side infor- mation in $p = 0.5$. $(H(X|Y) = 0.)$

DAC shows a great performance for strongly correlated sources which can achieve a 10^{-5} error probability when rate is around 0.6. The performance of length $N = 1000$ is better than that of length $N = 200$. In [8], DAC outperforms LDPC codes and Turbo codes in performance on small block length sequences.

DAC . The performance of length $N = 1000$ is better than that of length $N = 200$.In [8], DAC outperforms LDPC code and Turbo Code in performance on short length sequences.

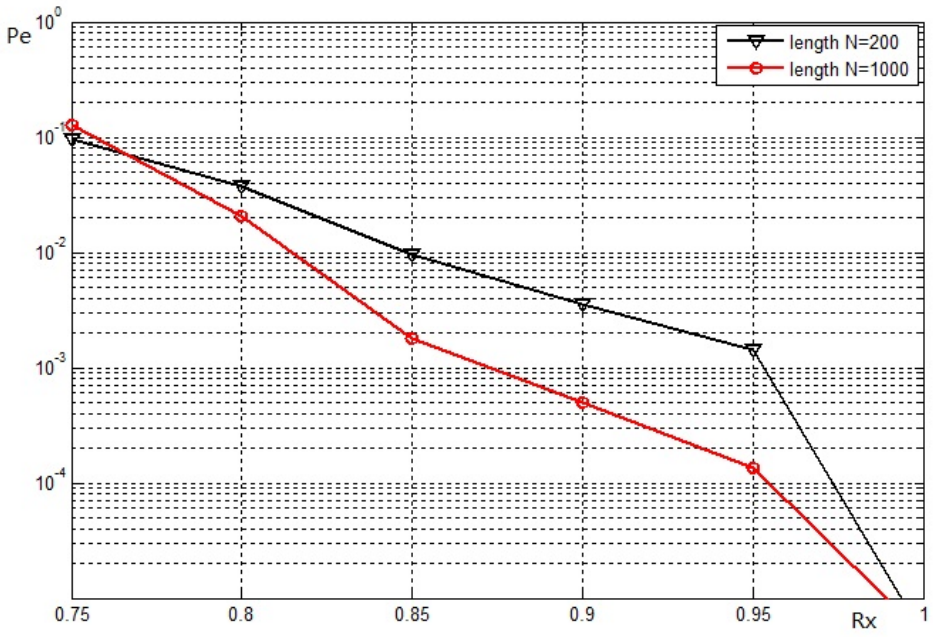

Figure 3.8: Error probability for weakly correlated side infor- mation in $p = 0.5$. $(H(X|Y) = 0.722)$

3.3.2 Block Distributed Arithmetic Coding

Another impressive work on DAC is Block Distributed Arithmetic Coding (Block DAC). It is proposed by in 2013 [28]. It improves the performance and applies DAC into a process that is similar to DISCUS. A brief overview on this algorithm is described in this section.

Problem Set Up

Discrete random variable X takes value from a binary alphabet $A = \{0, 1\}$ with a distribution $p(x = 0) = p_0, p(x = 1) = p_1$. Y is a random variable that is correlated with X by $p(x = y) = 1 - \epsilon$. X and Y are input to Block DAC, and the rate constraint for X and Y is

$$R_X + R_Y \geq H(X, Y) \tag{3.21}$$

In this section, we assume Y is transmitted in the rate $R_Y = H(Y)$, while X is encoded and decoded in Block DAC.

Overlap Design

The overlap rule in Block DAC is designed according to the rule that Overlapped intervals have a large hamming distance. It is conducted by finding a set of Grey Code in the most interior interval from 0 to 2^{n-k}. The steps are as follows:

- Leave a small interval d at the end of the interval [0,1]. Since a (n, k) code has 2^{n-k} syndromes computed by

$$s_i = c_i H^T \tag{3.22}$$

  H is the parity check matrix of Code $\mathbf{C}$, divide the interval $[0, 1-d]$ into 2^{n-k} subintervals, all subintervals are same in the same.

- Label the 2^{n-k} intervals from left to right with Grey Code form of 0 to 2^{n-k} so that the neighboured intervals differ in 1 bit.

- As the code $\mathbf{C}$ has property that these 2^{n-k} sequences are uniquely mapped to 2^{n-k} syndromes, we can label the interval with the syndrome the codeword belongs to. After modeling, the interval between 0 and 1 are divided into $2^{n-k}+1$ subintervals which are assigned to 2^{n-k} syndromes and the last interval with size d for future operation.

As each syndrome maps a coset which includes 2^k components. All codewords has a syndrome with all zero sequence. Since in last step, the interval $[0, 1-d]$ are divided into 2^{n-k} subintervals labeled with syndromes, then put the coset regarded to the syndrome on the interval and from the second component to the 2^k component, shift the interval with to the right with distance d_k, and d_k is calculated by

$$d_k = \frac{d(k-1)}{2^k - 1} \tag{3.23}$$

d is the last interval at the design step, and k is the label of the sequence in the coset. In this design, the intervals are overlapped with each other in a way that ensures all overlapped intervals has a larger hamming distance. As a (n, k) code $\mathbf{C}$ has a minimal hamming distance d_{min} that guarantees in a coset the minimal hamming distance is d_{min}, however, we assign the label of interval based on Grey Code property that ensures neighboured intervals differ in 1 bit, so that all overlapped intervals has a hamming distance larger than $d_{min} - 1$. Based on Coding Theory, even without any side information, the code itself is capable of correcting t errors, and t is calculated by

$$t = \lfloor \frac{d_{min} - 2}{2} \rfloor \tag{3.24}$$

Encoding Process

The Encoding process of the Block Distributed Arithmetic Coding is the same as Distributed Arithmetic Coding, it is a DAC encoding process conducted in Block level. The steps are as follows:

- Initialize the low and high as $l = 0, h = 1$
- When a block comes, find the interval the block is mapped and update l and h by the rule:

$$l(x_1, x_2...x_n) = l(x_1, x_2...x_{n-1}) + l(x_n) * \tilde{p}(x_1, x_2...x_{n-1}) \tag{3.25}$$

$$h(x_1, x_2...x_n) = l(x_1, x_2...x_{n-1}) + h(x_n) * \tilde{p}(x_1, x_2...x_{n-1}) \tag{3.26}$$

- After all blocks are encode, choose a real number between l and h and converted into a binary sequence with length $\lceil -\log_2 \tilde{p}(x_1, x_2...x_n) \rceil$

Decoding Process

The Decoding Process of Block Distributed Arithmetic Coding is also similar Distributed Arithmetic Coding, it is a DAC decoding process conducted in Block level.

Algorithm 2 Block DAC decoding process

```
Initialize the root node T_0
while i < N do
  for All nodes in T_i do
    D_{i+1} = Test(T_i)
    if D_{i+1} in overlap region then
      for x_{i+1} = (0, 1) do
        n_{i+1} = decode(D_{i+1})
        T_{i+1} = T_{i+1} + n_{i+1}
      end for
    else
      n_{i+1} = decode(D_{i+1})
      T_{i+1} = T_{i+1} + n_{i+1}
    end if
  end for
  Sort the nodes in T_{i+1} by side information
  Keep M most reliable nodes in every T_{i+1}
end while
```

The decoding tree is set up to find out the true path based on the Metrics with is the correlation between the source and side information in the form that:

$$\begin{aligned}(\hat{x}_1, \hat{x}_2 ... \hat{x}_n) &= \arg\max_{(x_1, x_2 ... x_n)} p(x_1, x_2 ... x_n | (y_1, y_2 ... y_n)) \\ &= \arg\max_{(x_1, x_2 ... x_n)} \prod_{i=1}^{n} P(x_i | y_i)\end{aligned} \tag{3.27}$$

Performance

Compared to Distributed Arithmetic Coding, the structure of Block Distributed Arithmetic Coding is well designed, so that the performance of Block DAC is superior to DAC, since the code itself is capable of correcting errors which the code of DAC is not able to.

In the simulation, a sequence of binary discrete random variable X characterized by $p(X = 0) = 0.5$ is encoded and decoded in binary Distributed Arithmetic Coding. The rate R is controlled in the range $H(X|Y) \leq R \leq H(X)$. The side information Y is the result of X going through a binary symmetric channel with crossover probability ϵ.

In the simulation , Y is compressed in a rate that equals to its entropy, while$X$ is encoded in Block Distritbuted Arithmetic Coding. The codewords are transmitted to a joint decoder. A (8,4) extended Hamming Code is used in Block DAC. M algorithm and termination policy are applied with $M = 2048$, $T = 15$. The length of sequence N is 400, the performance is shown in Fig 3.9, X axis shows the total rate of source X and side information Y, while Y axis shows the probability of error at decoder.

In Fig 3.9, Block DAC outperforms DAC in performance of uniform distributed binary source.

Intuitively, Block DAC is superior to DAC since the codeword itself is able to correct errors. The simulation results prove this hypothesis. The proposition of Block DAC can be considered as a DISCUS using in DAC scheme. It partitions a input sequence into blocks with channel codes. The concept of Hamming distance is brought in to achieve a lower probability of error. However, it can only process equal-probable binary input, which limits its application. The Block Distributed Arithmetic Coding can also be considered as a Uniform Distributed Non-binary DAC when every interval is assigned with a symbol in a non-binary alphabet. However, it can only process uniform distributed source. In addition,

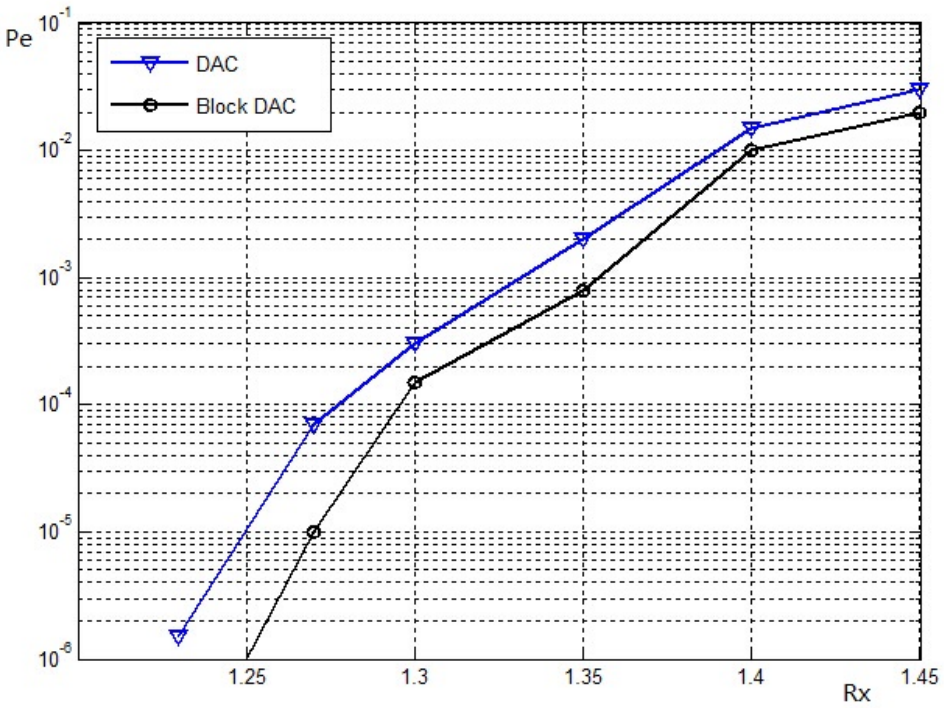

Figure 3.9: Simulation result for Block DAC [28]

the advantage of Block DAC in the structure that overlapped intervals have large hamming distance losses its function.

Chapter 4

Overlap Rule Design in Non-uniform Binary Distributed Arithmetic Coding

Distributed Arithmetic Coding(DAC) is a new data compression algorithm in Wireless Sensor Networks(WSNs) which has a simple encoding process and a low probability of error [7] [9]. It is an effective coding algorithm in DSC, especially for short length sequences. It is an extension of arithmetic coding with overlaps between intervals. Overlap rule design plays an important role in the DAC process. Hence, we aim to investigate how much performance can be affected by different overlap rules and whether the optimal performance can be achieved through a certain design . In this chapter, research illustrating several proposed overlap rules is conducted in order to assess the impact of different designs on performance.

4.1 Problem Set-Up

We begin with a quick overview of the DAC process. In binary DAC, a sequence of symbols are mapped into two overlapped intervals and the process is conducted recursively until all

the symbols are encoded, then the encoded real number is converted into a binary sequence as the codeword.

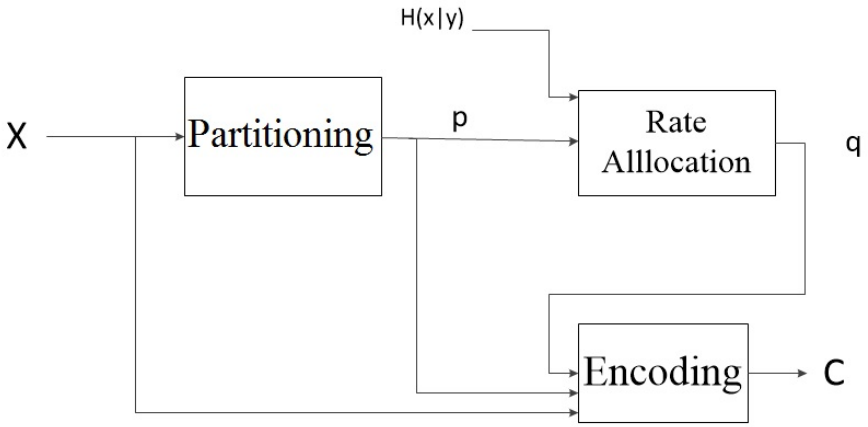

Figure 4.1: Encoding process of DAC

The encoding process in DAC is shown in Fig 4.1. A length N discrete input symbol sequence X takes value in alphabet {0,1} in a distribution $p(x_i = 0) = p_0$ for $i = 1, 2...N$. In the modeling stage of DAC, the interval $[0, 1]$ is divided into two subintervals: the first one is $I_0 = [0, p_0]$ and the subsequent one is $I_1 = [p_0, 1]$. Then I_0 and I_1 are modified into length q_0 and q_1, which is a function that involves the distribution of X and the rate as

$$q_i = F(p_i, R), i \in \{0, 1\} \tag{4.1}$$

the modified intervals $\tilde{I}_0 = [0, q_0]$ and $\tilde{I}_1 = [1 - q_1, 1]$ overlap each other in Fig 4.2.

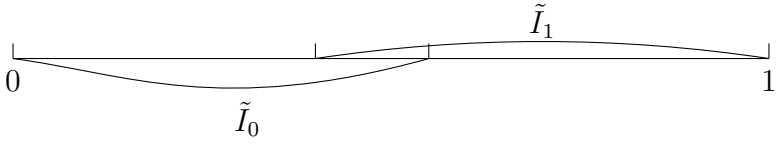

Figure 4.2: Overlap structure in encoding process

In the encoding process, each symbol in X is mapped into interval $\tilde{I}_0$ and $\tilde{I}_1$ recursively and once all N symbols are encoded, the real number, denoted as Z, is encoded into a length $\lceil -\log_2 \prod_{i=1}^{N} q_i \rceil$ binary sequence.

In the encoding stage, we can distinguish that the overlapped intervals q_0 and q_1 are a function that involves distribution $p(x)$ and the rate R with a constraint

$$R = -\sum_{i=1}^{N} p_i \log_2 q_i \tag{4.2}$$

In the formulation, if the distribution of the random variable X is a uniform distribution, $q_0 = q_1 = q$, the size of overlapped intervals are the same so that different overlap rules will result in a similar set up. However, in non-uniform distributions, intuitively, different overlap rules will offer different $\tilde{I}_0$ and $\tilde{I}_1$ configuration, and it is likely to have an impact on the probability of error.

4.2 Proposed Overlap Rule Design

The size of enlarged intervals q_0 and q_1 are a function of the distribution $p(x)$ and the compression rate r. For a fixed rate, a lot of configurations of (q_0, q_1) can be selected. The objective is to find a pair of (q_0, q_1) to minimize the probability of error that

$$(\hat{q}_0, \hat{q}_1) = \arg\min_{q_0, q_1} p(e) \tag{4.3}$$

In this section, four overlap functions are proposed to achieve optimal performance.

4.2.1 Rule 1

The first overlap rule is proposed by Marco Grangetto in [8]. Whereby, a parameter α is applied in the function that

$$q_i = \alpha p_i, i = 0, 1 \tag{4.4}$$

Each interval is enlarged by the same parameter α, and the rate R can be calculated by

$$\begin{aligned} R &= -\sum_{i=1}^{N} p_i \log_2 q_i \\ &= -\sum_{i=1}^{N} \log_2 \alpha p_i \\ &= H(x) - log_2 \alpha \end{aligned} \tag{4.5}$$

Once the rate is selected, parameter α can be calculated and the size of enlarged intervals q_0 and q_1 are generated.

In this design, the size of the overlap $\alpha p_0 + \alpha p_1 - 1 = \alpha - 1$ is the smallest. In addition,it minimizes the divergence between the original distribution $p(x)$ and the modified interval sizes $q(x)$ that $\frac{p_0}{p_1} = \frac{q_0}{q_1}$. These properties are conductive to excellent performance. However, it involves neither an optimization problem nor probability of error on decoder's part. The performance is simply not guaranteed. In addition, when a source input is compressed at a low rate, the interval size may be larger than 1 since there is no bound on the modified interval size.

4.2.2 Rule 2

The second overlap rule is also proposed by Marco Grangetto in [8]. The intention is to place a limit on the overlapped interval sizes q_0 and q_1 to guarantee the sizes of q_0 and q_1 are smaller than 1. In this formulation, the function is in a form that

$$q_i = p_i^k \tag{4.6}$$

in which k is a positive real number between 0 and 1. In this set up, the rate R has the form

$$\begin{aligned} R &= -\sum_{i=1}^{N} p_i \log_2 q_i \\ &= -\sum_{i=1}^{N} p_i \log_2 p_i^k \\ &= kH(x) \end{aligned} \tag{4.7}$$

Once the rate is selected so that k can be calculated and the sizes of enlarged intervals q_0 and q_1 are generated.

This is also a fairly intuitive set up for a q function in favor of its brevity in computation and the property that interval sizes are bounded in [0,1]. However, it does not include any optimization in the probability of decoding error which may lead to a subpar performance.

4.2.3 Rule 3

Another overlap rule is proposed in this thesis for non-binary DAC, which can also be applied in binary cases. The set up is as follows:

- Initialize the interval from 0 to 1 into two subintervals $[0, 1-d]$ and $[1-d, 1]$: d is a parameter in this formulation to control the rate R.
- Divide the interval [0,1-d] into two non-overlapped subintervals based on the distribution of the source X: the first interval is $[0, p_0(1-d)]$ and the second is $[p_0(1-d), 1-d]$.
- Enlarge the two subintervals we have in step two by keeping the lower bound of each interval the same and shifting the upper bounds by d to the right. This design results in two overlapped intervals $[0, p_0(1-d)+d]$ and $[p_0(1-d), 1]$.

In this formulation, the q function is:

$$q_i = p_i(1-d) + d, i = 0, 1 \tag{4.8}$$

and rate R is computed by:

$$\begin{aligned} R &= -\sum_{i=1}^{N} p_i \log_2 q_i \\ &= -\sum_{i=1}^{N} p_i \log_2 [p_i(1-d) + d] \end{aligned} \tag{4.9}$$

This is a proposed overlap rule which can be applied to non-binary DAC: since it does not allow two intervals to overlap completely, the suboptimal performance is guaranteed. In this formulation, the overlap can be controlled through parameter d, and the two overlapping intervals are enlarged by the same size. This is of great benefit to non-binary DAC since this design excutes a concise computation in the decoding process when the number of intervals are large. However, no performance optimization is applied.

4.2.4 Rule 4

Due to the non-linear feature of the DAC process is a non-linear process, it is impractical to seek a standard function of decoding error probability, since we can not estimate which path the decoding tree is following. In this section, we endeavor to set a bound on DAC and optimize this bound by q_0 and q_1 to identify the suboptimal configuration for DAC process.

The decoding error probability is determined by the symbol input X, the side information Y, and the rebuilt real number Z. Although we are unable to identify the probability of error in a formulation containing X, Y and Z, it is possible to offer a lower bound on that probability of error based on Fano's inequality.

Fano's Inequality

Fano's inequality is a concept related to information theory that was first developed by Robert Fano in the 1950s. This theory applies to information lost in a noisy channel: it can be used to give a lower bound in the probability of error in our work. The formulation is:

$$H(X|Y) \leq H(e) + p(e)\log(|\chi| - 1) \tag{4.10}$$

X and Y are input and output messages, and $p(e)$ is the probability that $p(Y \neq X)$, the conditional entropy can be used as a lower bound of the probability of error.

Minimizing lower bound of error probability

In DAC, there are three variables involved: X, Y and Z. X is an input binary discrete random variable, Y is a correlated sequence result from X going through a BSC channel with crossover probability ϵ, and Z is the rebuilt version of the encoded result. Based on Fano's inequality, the lower bound for probability of error is $H(X|Y, Z)$. Additionally, from the factor graph of the decoding process, we can observe that the decoding process is a kind of Hidden Markov Model, in which we have the formulation:

$$\begin{aligned} H(x_n|y_1, y_2...y_n, x_1, x_2...x_{n-1}, Z) &= H(x_{n-1}|y_1, y_2...y_{n-1}, x_1, x_2...x_{n-2}, Z) = ... \\ &= H(x_1|y_1, Z) \end{aligned} \tag{4.11}$$

It is an approaximation we make and our goal is to minimize the conditional entropy $H(x_1|y_1, Z)$.

Let $x_1, x_2...x_N$ be an i.i.d. Bernoulli sequence parametrized by distribution $p := (p_0, p_1)$. The sequence passes through a binary symmetric channel with crossover probability ϵ and

generates output $y_1, y_2 ... y_N$. The size of intervals in DAC are determined by parameters q_0 and q_1 so that the distribution of Z is parametrized by q_0 and q_1. $H(X|Y,Z)$ becomes a function with random variables q_0 and q_1, and the objective is to find the optimal configuration of q_0 and q_1 to minimize the function $H(X|Y,Z)$ which is

$$(\hat{q}_0, \hat{q}_1) = \arg\min_{q_0,q_1} H(X|Y, Z(X)) \tag{4.12}$$

by definition

$$H(X|Y, Z(X)) = -\sum_{x,y,z} p(x_i, y_i, z(x_i)) \log p(x_i|y_i, z(x_i)) \tag{4.13}$$

$z(x_i)$ is denoted by z_i below. Since X, Y and Z forms a Markov Chain in which the given X, Y and Z are independent, $p(x_i, y_i, z_i)$ is in the form:

$$p(x_i, y_i, z_i) = p(y_i, z_i|x_i)p(x_i) = p(y_i|x_i)p(z_i|x_i)p(x_i) \tag{4.14}$$

and the other term $p(x_i|y_i, z_i)$ can be formulated to

$$p(x_i|y_i, z_i) = \frac{p(x_i, y_i, z_i)}{p(y_i, z_i)} = \frac{p(x_i, y_i, z_i)}{\sum_x p(x_i, y_i, z_i)} \tag{4.15}$$

then the conditional entropy function

$$H(X|Y,Z) = -\sum_{x,y,z} p(x_i, y_i, z_i) \log \frac{p(x_i, y_i, z_i)}{p(y_i, z_i)} \tag{4.16}$$

Discrete random variable X and Y take values from $\{0, 1\}$, and Z is considered as a discrete random variable taking values in {Left-interval,Mid-interval,Right-interval}. The value of Z depends on which interval the rebuilt real number locates. Given X, the distribution of Y can be denoted as

$$p(Y|X) = \begin{cases} \epsilon & Y \neq X \\ 1-\epsilon & Y = X \end{cases} \tag{4.17}$$

Approximation

The exact distribution of the random variable Z is unknown, hence, we make a hypothesize that Z is a random point uniformly distributed in the encoding interval. It may not be accurate but it is impractical to determine the true distribution of Z [4], so a uniform distribution is considered reasonable to reflect the true distribution. Therefore, the distribution of Z given that X is denoted by

$$p(Z|X=0) = \begin{cases} \frac{1-q_1}{q_0} & Z = L \\ \frac{q_0+q_1-1}{q_0} & Z = M \\ 0 & Z = R \end{cases} \tag{4.18}$$

$$p(Z|X=1) = \begin{cases} 0 & Z = L \\ \frac{q_0+q_1-1}{q_1} & Z = M \\ \frac{1-q_0}{q_1} & Z = R \end{cases} \tag{4.19}$$

Formulation Deriving

Since all the components in the formulation can be calculated, here we list all the components for $p(X, Y, Z)$ as

$$
\begin{aligned}
p(0,0,L) &= p(Y=0|X=0)p(Z=L|X=0)P(X=0) &= p_0(1-\epsilon)\tfrac{1-q_1}{q_0}\\
p(0,0,M) &= p(Y=0|X=0)p(Z=M|X=0)P(X=0) &= p_0(1-\epsilon)\tfrac{q_0+q_1-1}{q_0}\\
p(0,1,L) &= p(Y=1|X=0)p(Z=L|X=0)P(X=0) &= p_0\epsilon\tfrac{1-q_1}{q_0}\\
p(0,1,M) &= p(Y=1|X=0)p(Z=M|X=0)P(X=0) &= p_0\epsilon\tfrac{q_0+q_1-1}{q_0}\\
p(1,0,M) &= p(Y=0|X=1)p(Z=M|X=1)P(X=1) &= p_1\epsilon\tfrac{q_0+q_1-1}{q_1}\\
p(1,0,R) &= p(Y=0|X=1)p(Z=R|X=1)P(X=1) &= p_1\epsilon\tfrac{1-q_0}{q_1}\\
p(1,1,M &= p(Y=1|X=1)p(Z=M|X=1)P(X=1) &= p_1(1-\epsilon)\tfrac{q_0+q_1-1}{q_1}\\
p(1,1,R) &= p(Y=1|X=1)p(Z=R|X=1)P(X=1) &= p_1(1-\epsilon)\tfrac{1-q_0}{q_1}
\end{aligned}
$$

then we calculate the marginal distribution for Y and Z by the sum over X this joint distribution, results in :

$$
\begin{aligned}
p(Y=0,Z=L) &= p_0(1-\epsilon)\tfrac{1-q_1}{q_0}\\
p(Y=1,Z=l) &= p_0\epsilon\tfrac{1-q_1}{q_0}\\
p(Y=0,Z=M) &= p_0(1-\epsilon)\tfrac{q_0+q_1-1}{q_0} + p_1\epsilon\tfrac{q_0+q_1-1}{q_1}\\
p(Y=1,Z=M) &= p_0\epsilon\tfrac{q_0+q_1-1}{q_0} + p_1(1-\epsilon)\tfrac{q_0+q_1-1}{q_1}\\
p(Y=0,Z=R) &= p_1\epsilon\tfrac{1-q_0}{q_1}\\
p(Y=1,Z=R) &= p_1(1-\epsilon)\tfrac{1-q_0}{q_1}
\end{aligned}
$$

and from the calculation we can see that for some configurations of Y and Z, we have $p(y,z) = p(x,y,z)$ such that $\log\frac{p(x,y,z)}{p(y,z)} = 0$, then the conditional entropy becomes

$$
\begin{aligned}
H(X|Y,Z) = -\textstyle\sum_{x,y,z} p(x_i,y_i,z_i)\log\frac{p(x_i,y_i,z_i)}{p(y_i,z_i)}\\
= -p(X=0,Y=0,Z=M)\log\tfrac{p(X=0,Y=0,Z=M)}{p(Y=0,Z=M)} - p(X=0,Y=1,Z=M)\log\tfrac{p(X=0,Y=1,Z=M)}{p(Y=1,Z=M)}\\
-p(X=1,Y=0,Z=M)\log\tfrac{p(X=1,Y=0,Z=M)}{p(Y=0,Z=M)} - p(X=1,Y=1,Z=M)\log\tfrac{p(X=1,Y=1,Z=M)}{p(Y=1,Z=M)}
\end{aligned}
\tag{4.20}
$$

After transforming

$$
\begin{aligned}
H(X|Y,Z) = -(q_0+q_1-1)[\tfrac{p_0(1-\epsilon)}{q_0}\log\frac{\frac{p_0(1-\epsilon)}{q_0}}{\frac{p_0(1-\epsilon)}{q_0}+\frac{p_1\epsilon}{q_1}} + \tfrac{p_1\epsilon}{q_1}\log\frac{\frac{p_1\epsilon}{q_1}}{\frac{p_0(1-\epsilon)}{q_0}+\frac{p_1\epsilon}{q_1}}\\
+\tfrac{p_1(1-\epsilon)}{q_1}\log\frac{\frac{p_1(1-\epsilon)}{q_1}}{\frac{p_1(1-\epsilon)}{q_1}+\frac{p_0\epsilon}{q_0}} + \tfrac{p_0\epsilon}{q_0}\log\frac{\frac{p_0\epsilon}{q_0}}{\frac{p_0\epsilon}{q_0}+\frac{p_1(1-\epsilon)}{q_1}}]
\end{aligned}
\tag{4.21}
$$

Then if the distribution of the input source random variable X is fixed, and the correlation between X and Y are fixed, it means that q_0, q_1 are the only random variables in $H(X|Y,Z)$. Consequently, the solution is clear in order to achieve the optimal q_0, q_1 by

$$(\hat{q}_0, \hat{q}_1) = \arg\min_{q_0,q_1} H(X|Y, Z(X)) \tag{4.22}$$

The problem is optimized by identifying all possible configurations of q_0 and q_1. The optimal configuration is applied in the DAC encoding and decoding process.

This design offers a function to minimize a theoretical bound of the probability of error. With support of theorem, the performance cannot be worse than the three rules introduced above. However, the computing requirements in this design are huge. Furthermore, several approximations whose impact on performance are unknown have been applied.

4.3 Performance Evaluation

In this section performance evaluation of four overlap rules is compared through simulations on source input with various correlations and distributions. The DAC process is the same as proposed in [8], while the interval sizes are generated through different functions. The result is shown with various rates in the X axis and the probability of error which is the probability that a symbol in an input sequence is decoded incorrectly in the Y axis.

In the simulation, the distribution in the test is a $p_0 = 0.6, p_0 = 0.8,$ and $p_0 = 0.9$. In each distribution, the simulation is conducted in two sequence length and correlations between source input and side information. Specifically, the length N is simulated with $N = 200$ and $N = 1000$, the parameter in the BSC channel is selected $\epsilon = 0.05$ as a high correlation and $\epsilon = 0.15$ reflects low correlation. In DAC, M=2048 and T=15.

4.3.1 Performance Evaluation For Short Length Strongly Correlated Source Sequences

The simulation result of $N = 200, p_0 = 0.6, and \epsilon = 0.05$ is shown in Fig 4.3.

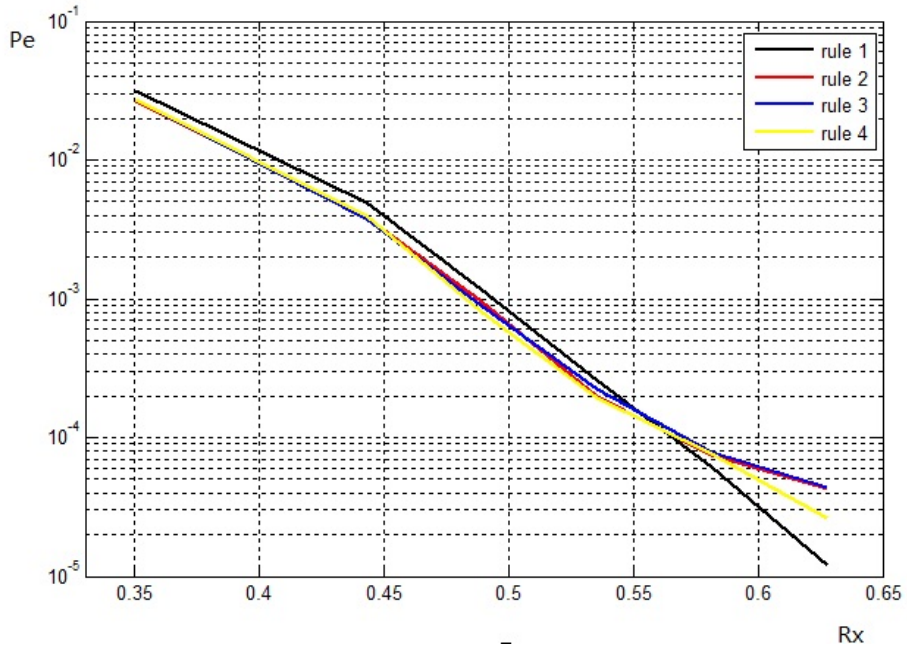

Figure 4.3: Error probability for 200 length sequences with strongly correlated side information in $p = 0.6$. $(H(X|Y) = 0.28)$

In this set-up, entropy $H(X) = 0.97$ and the conditional entropy $H(X|Y) = 0.28$ In Fig 4.3, the four overlap rule designs seems to have similar performance with rule 1 displaying superior results when the rate is towards the entropy.

The simulation result of $N = 200, p_0 = 0.8, and \epsilon = 0.05$ is shown in Fig 4.4.

In this set-up, entropy $H(X) = 0.72$ and the conditional entropy $H(X|Y) = 0.28$. In Fig 4.4, rule 1 is not able to compress the source into a rate lower than 0.45 since the size of interval q_0 is larger than 1. The difference among these four overlap rules increases as the distribution becomes further from uniform distribution. However, although we can discern that rule 2, rule 3 and rule 4 are superior to rule 1 in this arrangement, we cannot verify which overlap rule is most favourable in every circumstance.

The simulation result of $N = 200, p_0 = 0.9, and \epsilon = 0.05$ is shown in Fig 4.5.

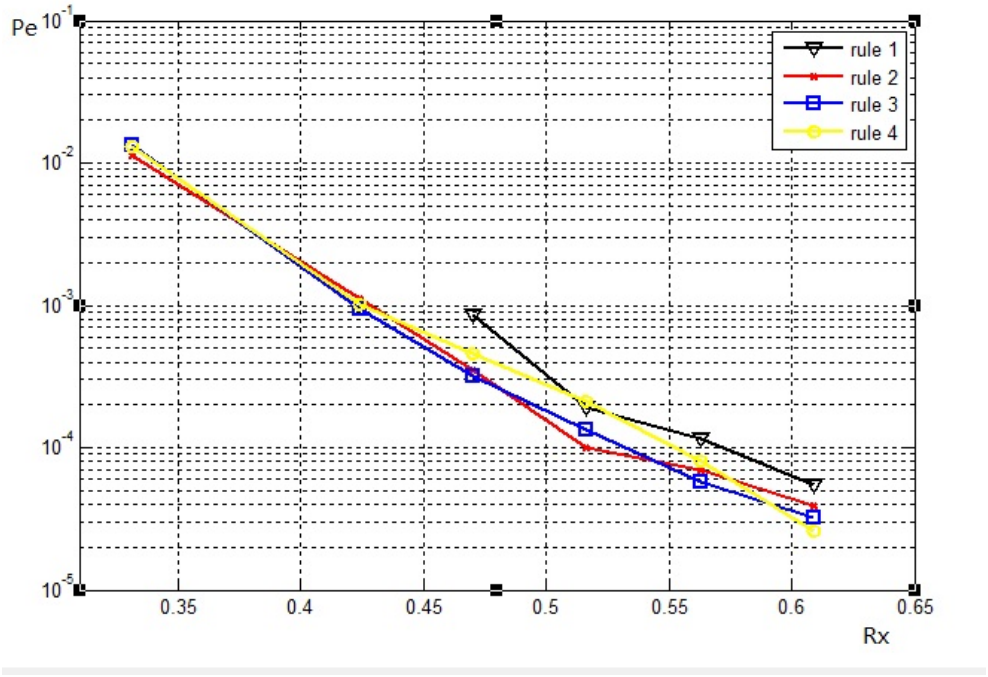

Figure 4.4: Error probability for 200 length sequences with strongly correlated side information in $p = 0.8$. $(H(X|Y) = 0.28)$

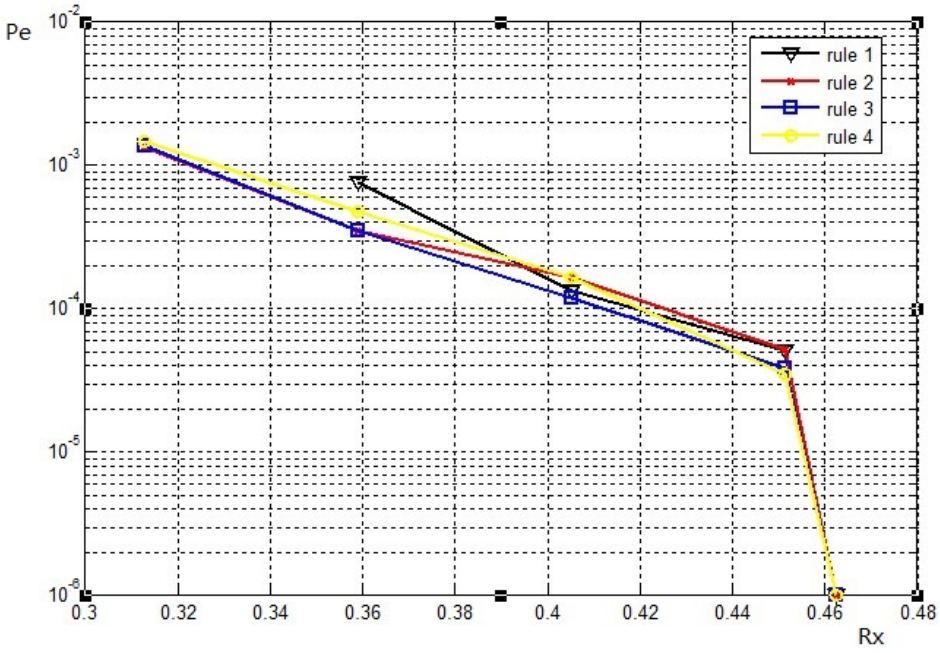

Figure 4.5: Error probability for 200 length sequences with strongly correlated side information in $p = 0.9$. $(H(X|Y) = 0.28)$

In this set-up, entropy $H(X) = 0.46$ and the conditional entropy $H(X|Y) = 0.28$. In Fig 4.5, rule 1 can not be compressed into a rate lower than 0.36 since the size of interval q_0 is larger than 1. The difference among the four overlap design rules is not readily apparent due to the size of enlarged intervals q_0 and q_1 in the four overlap rules being fairly close.

4.3.2 Performance Evaluation For Short Length Weakly Correlated Source Sequences

The simulation result of $N = 200, p_0 = 0.6, and \epsilon = 0.15$ is shown in Fig 4.6.

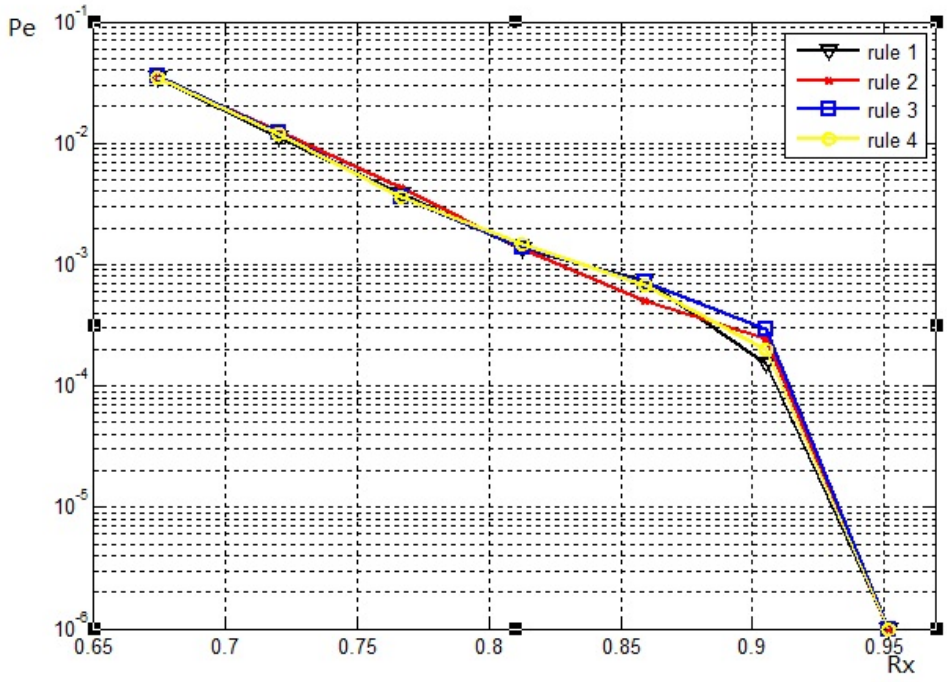

Figure 4.6: Error probability for 200 length sequences with weakly correlated side information in $p = 0.6$. $(H(X|Y) = 0.61)$

In this set-up, entropy $H(X) = 0.97$ and the conditional entropy $H(X|Y) = 0.61$. In Fig 4.6, the variation between the four overlap design rules is not obvious due to the size of enlarged intervals q_0 and q_1 in the four overlap rules being relatively close.

The simulation result of $N = 200, p_0 = 0.8, and \epsilon = 0.15$ is shown in Fig 4.7.

In this configuration, entropy $H(X) = 0.72$ and the conditional entropy $H(Y|X) =$

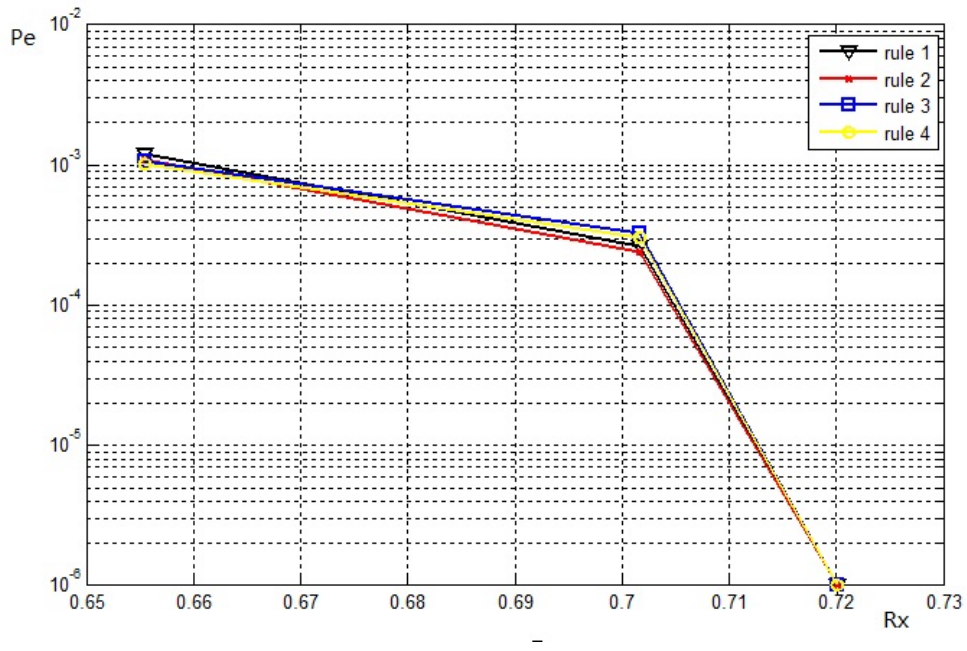

Figure 4.7: Error probability for 200 length sequences with weakly correlated side information in $p = 0.8$. $(H(X|Y) = 0.61)$

0.61. In Fig 4.7, the contrast between the four overlap design rules is not clear because the size of enlarged intervals q_0 and q_1 in the four overlap rules are fairly close.

4.3.3 Performance Evaluation For Long Length Strongly Correlated Source Sequences

The simulation result of $N = 1000, p_0 = 0.6, and \epsilon = 0.05$ is shown in Fig 4.8.

In Fig 4.8 the variation in performance is larger than that in $N = 200$ simulations, and the p_e in rule 2 drops quickly when the compression rate is around 0.5. The p_e for rule 3 and rule 4 is superior to the other two when the compression rate is around 0.6.

The simulation result of $N = 1000, p_0 = 0.8, and \epsilon = 0.05$ is shown in Fig 4.9.

In Fig 4.9 the performance of rule 4 exceeds that of the other overlap rules. The input in rule 1 cannot be compressed into a rate lower than 0.45. When the rate is slightly above the conditional entropy $H(X|Y)$, all three overlap rules have a nearly the same performance. However, when the rate increases towards the entropy $H(X)$, the p_e in rule

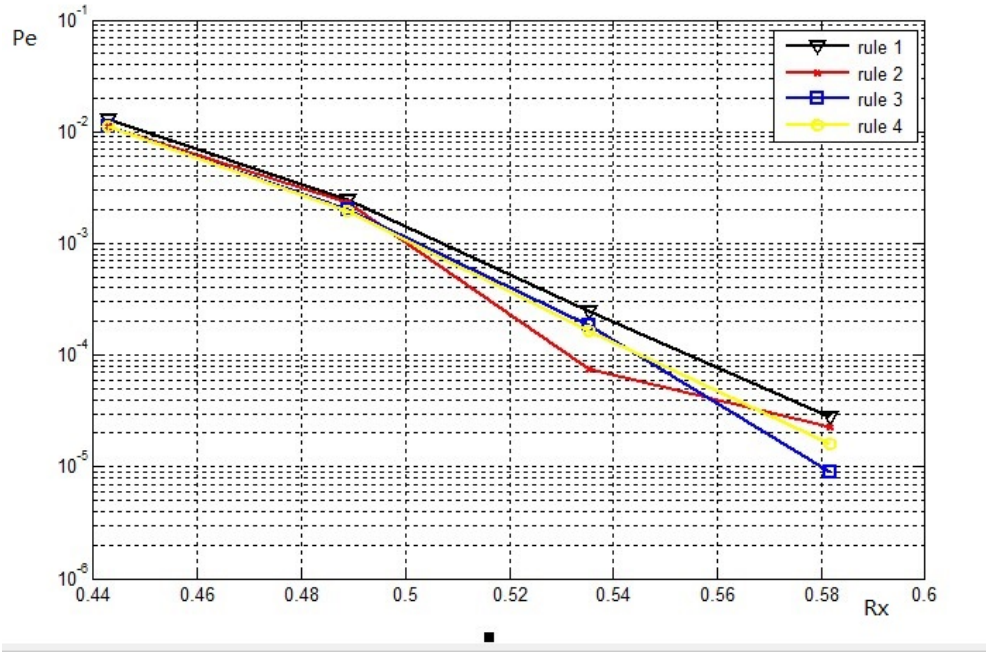

Figure 4.8: Error probability for 1000 length sequences with strongly correlated side information in $p = 0.6$. $(H(X|Y) = 0.28)$

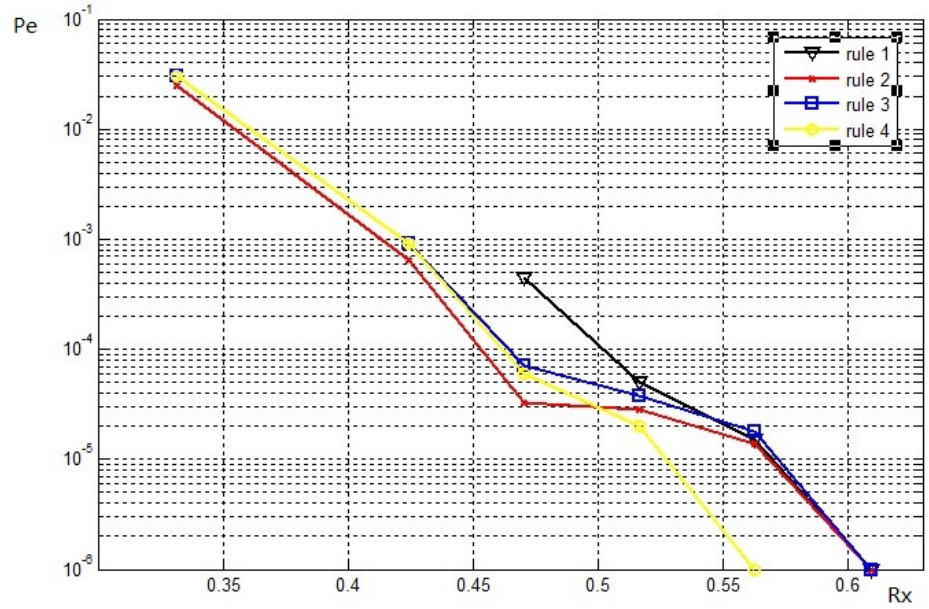

Figure 4.9: Error probability for 1000 length sequences with strongly correlated side information in $p = 0.8$. $(H(X|Y) = 0.28)$

4 is dropping sharply to 0. In this simulation, the performance of rule 4 is superior to that of the other three overlap rules.

The simulation result of $N = 1000, p_0 = 0.9, and \epsilon = 0.05$ is shown in Fig 4.10.

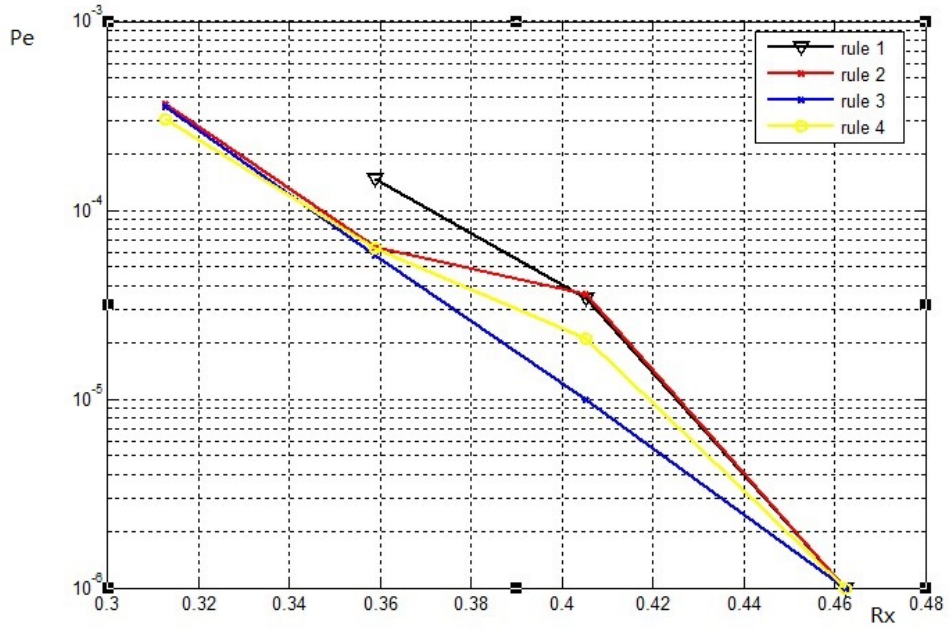

Figure 4.10: Error probability for 1000 length sequences with strongly correlated side information in $p = 0.9$. $(H(X|Y) = 0.28)$

In Fig 4.10 rule 3 greatly outperforms the other three overlap rules. Initially, the performance of all rules are nearly identical, then the rate changes towards the entropy and rule 3 exhibits a sharper drop than its counterparts with all three achieving a 0 error when the rate arriving at entropy $H(X)$.

4.3.4 Performance Evaluation For Long Length Weakly Correlated Source Sequences

The simulation result of $N = 1000, p_0 = 0.6, and \epsilon = 0.15$ is shown in Fig 4.11.

In Fig 4.11 rule 4 has a smaller p_e when the compression rate is around 0.85. The difference is not large, and the performance of four overlap rules are comparable.

The simulation result of $N = 1000, p_0 = 0.8, and \epsilon = 0.15$ is shown in Fig 4.12.

In Fig 4.12 the variation among the four overlap rules is relatively small. Rule 2 has a smaller p_e when the compression rate is around 0.7.

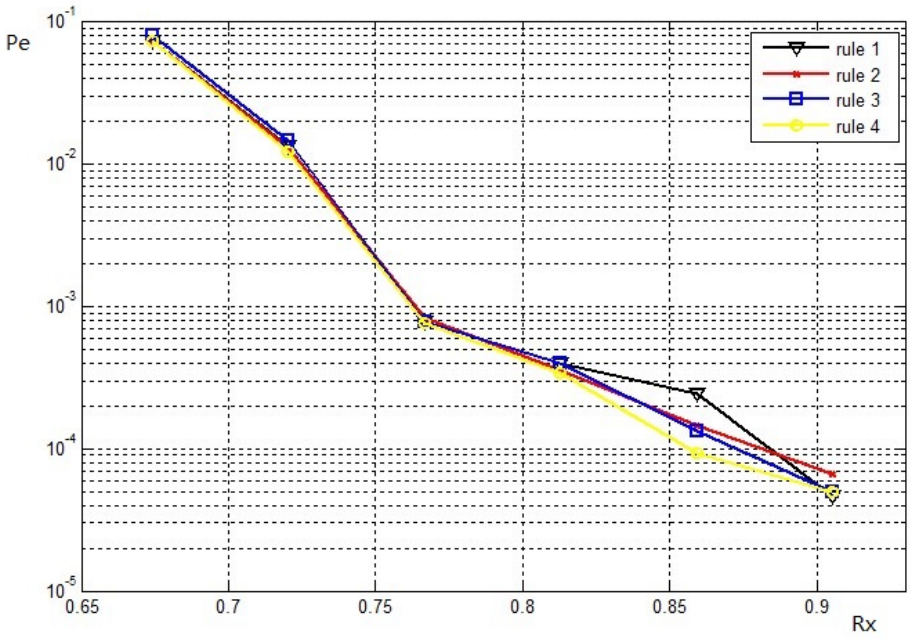

Figure 4.11: Error probability for 1000 length sequences with weakly correlated side information in $p = 0.6$. $(H(X|Y) = 0.61)$

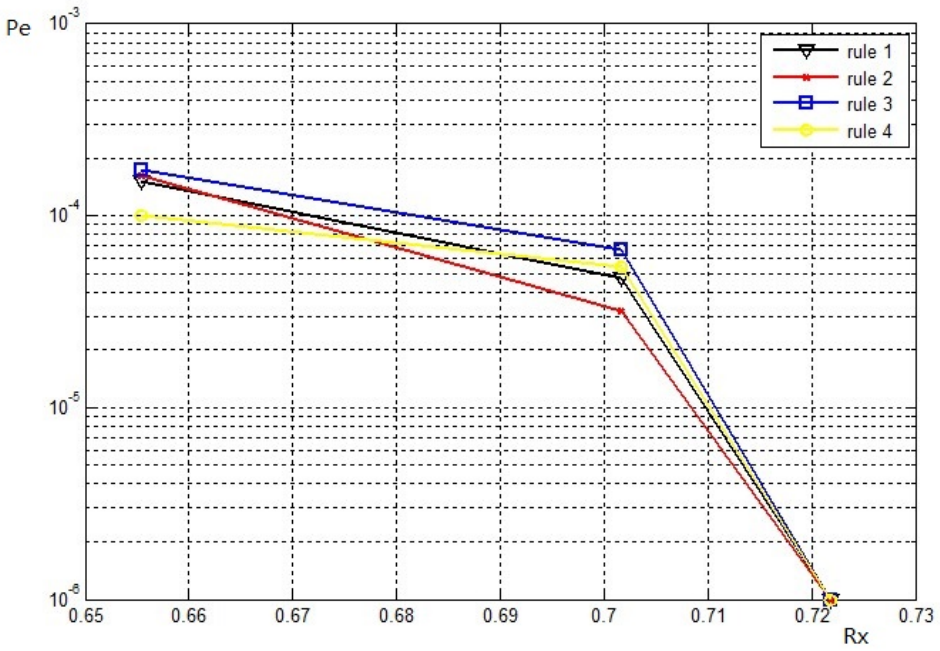

Figure 4.12: Error probability for 1000 length sequences with weakly correlated side information in $p = 0.8$. $(H(X|Y) = 0.61)$

4.4 Discussion and Conclusion

Through the design and performance evaluation, it is apparent that although the four overlap rule designs have varying function for the size of enlarged intervals q_0 and q_1, the impact on performance is not subtantial.

Of the four designed overlap rules, rule 1 and rule 2 are proposed based on their brevity in computation and simple function forms. Rule 3 is proposed based on its ability to reduce the decoding complexity in non-binary DAC. Rule 4 is proposed to optimize the lower bound of the decoding error. Presumably, rule 4 should outperform ts counterparts in all situations. However, the four overlap rules result in the same performance. After fully investigating the issue, the reasons are likely to be :

1. The DAC process is a suboptimal process that's intuitive design does not include any optimization problems. Although the performance is impressive for short sequences, the designs are all suboptimal.

2. In binary DAC, although four overlap functions result in different configurations of (q_0, q_1), these configurations differ in 10^{-3} or 10^{-2}, which is too small to have an impact on performance.

3. The error bound offered by Fano's inequality is too loose. In addition, the approximations we make are not accurate enough. The true distribution of Z is far from uniform distribution. In [4], an investigation of codeword distribution of equiprobable sources in binary DAC is proven. We simulate the distribution of codewords in binary DAC by counting the real number of codewords. The distribution is shown in Fig 4.13 and Fig 4.14 for equiprobable and non-equiprobable source sequences. The X axis shows the real number of the codewords, while the Y axis depicts the number of codewords that are able to convert to this real number. The shape displayed in the figure is far from uniform distribution. The approximation of distribution of Z may produce a huge loss in performance.

Although preliminary results show that the overlap rule design is not a key factor in

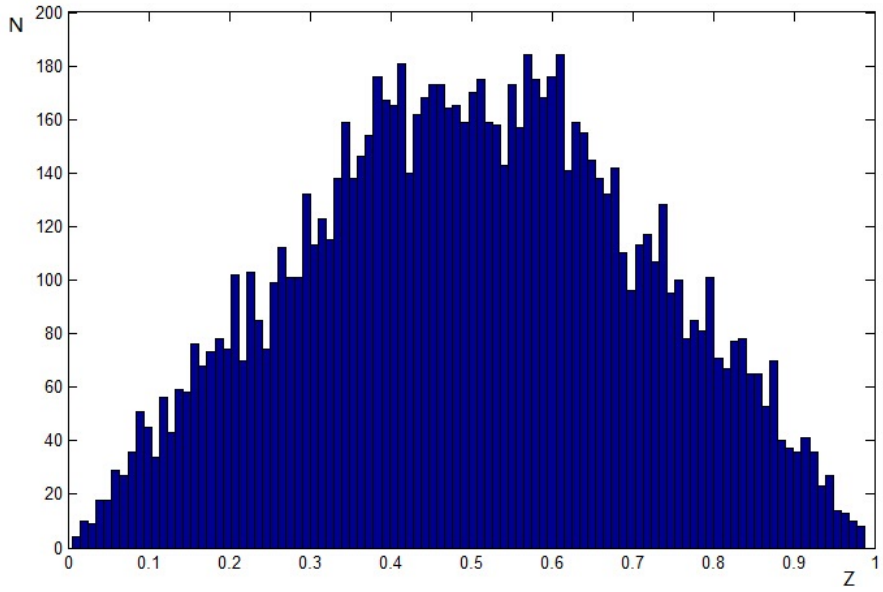

Figure 4.13: Distribution of real number of codewords for equiprobable source sequences

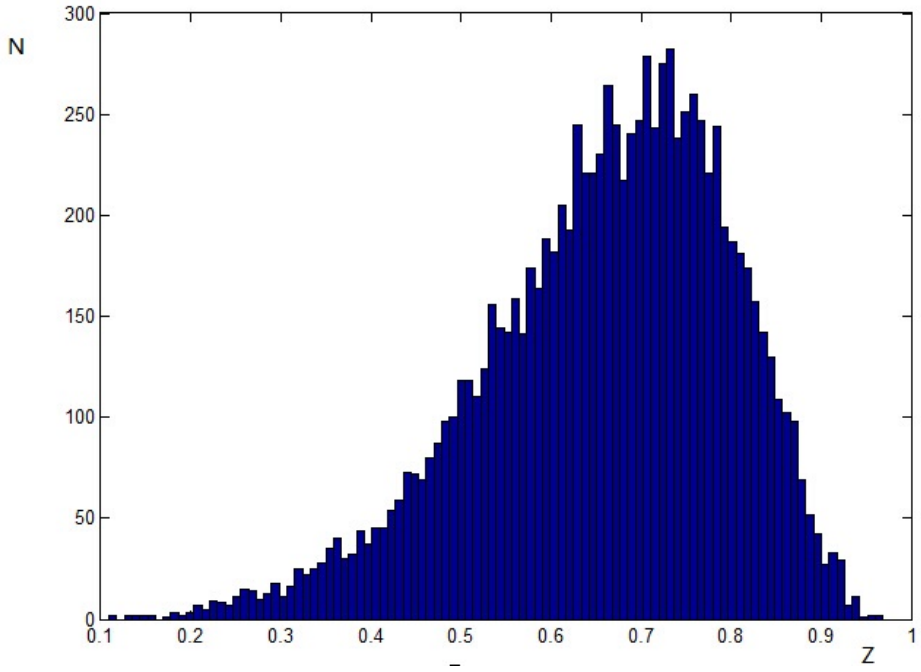

Figure 4.14: Distribution of real number of codewords for non-equiprobable source sequences

binary DAC, it likely plays an important role when the alphabet of source input increases. However, little research has been conducted on non-bianry DAC schemes as of now. We have placed emphasis on the proposition of non-binary DAC and leave the analysis of

overlap factors in non-binary DAC for future work.

Chapter 5

Non-binary Distributed Arithmetic Coding

DAC outperforms Turbo codes and LDPC codes in the performance of small and medium length binary source sequences. However, the fact that DAC cannot process non-binary input limits its applications.

There are plenty of source sequences taking values in a non-binary alphabet in practical applications in Wireless Sensor Networks(WSNs). In addition, the correlated non-binary sequences lose their intrinsic correlations when they are converted into a binary sequence by quantization resulting in the decoder experiencing difficulty using side information. A specific example is described to motivate the research on non-binary DAC.

Weather Telematics has a large wireless sensor network. Vehicles that carry a sensor travel around the country to collect weather data such as air temperature, road temperature, humidity and etc in different areas. The data is sent to the server in TCP every 8 seconds. An efficient data compression algorithm is required to reduce the size of the data streams in order to save cost. The performance of DAC on this project is excellent since the sequence length is small and the correlations among data in adjacent areas are fairly high. However, the data takes value from a large alphabet in order to increase accuracy and DAC cannot process non-binary input. By intuition, DAC is able to outperform any other algorithms if it can process non-binary input in this project. Therefore, a proposed

algorithm on the non-binary DAC scheme is necessary.

In this chapter, two proposed algorithms are presented as follow up work of DAC. They allow non-binary input sequences and achieve high efficiency.

5.1 Multi-Interval Distributed Arithmetic Coding (MI-DAC)

The proposed scheme is an algorithm related to a non-binary DAC problem. Following the binary DAC, the encoder of our proposed scheme is composed of two stages: the partitioning stage and the encoding stage.

In the partitioning stage, the manner in which the intervals are segmented and symbols are assigned to intervals is illustrated. Subsequently, the encoding stage consists of the process whereby an input sequence is encoded into a binary codeword. The entire process of encoder is shown in Fig 5.1:

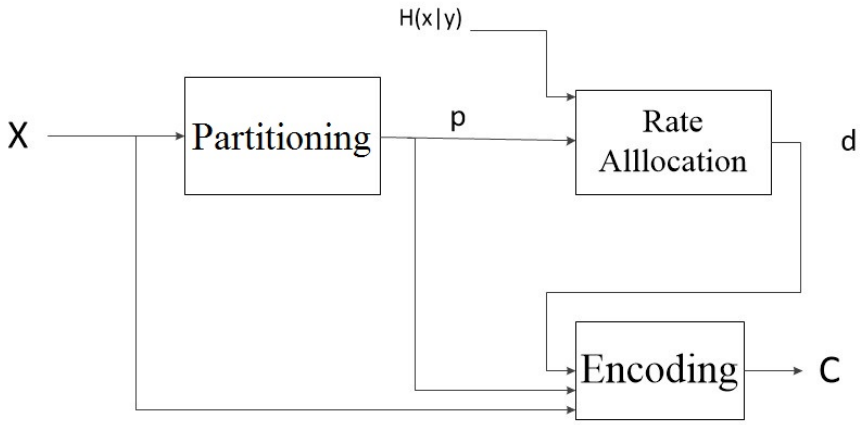

Figure 5.1: The structure of Multi-interval DAC encoder

5.1.1 Partitioning

A discrete random variable X is taking values from a q-ary alphabet A with a distribution $p(x)$. The rule to design overlaps among symbols in the alphabet is shown below:

- Initialize the intervals from 0 to 1 into two subintervals $[0, 1-d]$ and $[1-d, 1]$ with a parameter d, d is controlled by allocated rate of the source X.

- Divide the interval $[0, 1-d]$ into q non-overlapped subintervals based on the distribution of X, and each subinterval I_i can be characterized by l_i and h_i which can be calculated by

$$\begin{aligned} l_i &= \sum_{k=0}^{i-1} p(x_k)(1-d) \\ h_i &= \sum_{k=0}^{i} p(x_k)(1-d) \end{aligned} \tag{5.1}$$

  The structure is shown in Fig 5.2:

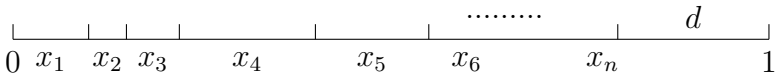

Figure 5.2: Structure of overlap design

- In the final step, d is added to the upper bound of each interval h_i so that sizes of all intervals grow larger and intervals overlap with other intervals. After modifying each interval, all intervals fill the region from 0 to 1, and the interval assigned to symbol x_i is characterized by l_i and h_i which can be calculated by:

$$\begin{aligned} l_i &= \sum_{k=0}^{i-1} p(x_k)(1-d) \\ h_i &= \sum_{k=0}^{i} p(x_k)(1-d) + d \end{aligned} \tag{5.2}$$

  Furthermore, all symbols and intervals are matched in pairs, and the size of each interval q_i can be computed in the formulation :

$$q_i = p(x_i)(1-d) + d \tag{5.3}$$

While the rate R is computed in the form:

$$R = -\sum p(x_i)\log_2 q_i = -\sum p(x_i)\log_2[p(x_i)(1-d)+d] \tag{5.4}$$

Figure 5.3: Structure of Multi-interval DAC

In this partition, the overlap design is quite intuitive. Although it is characteristically concise in its computation and demands less computing requirements in the decoding process, it does not have an optimal formulation in the probability of error. However,the DAC process is not a linear process and all overlap rules proposed are suboptimal. Optimizing the decoding error seems impractical and it is not guaranteed that adding a layer of complexity will necessarily achieve better results. Therefore, the overlap design is not necessarily optimal, and the performance is guaranteed with partially overlapping property.

5.1.2 Encoding Process

The encoding stage is similar to the encoding process of DAC: every symbol x_i in the sequence X is iteratively mapped to an interval. The process is described in detail below:

- Initialize the interval characterized by low and high as $l_0 = 0, h_0 = 1$

- When a symbol is presented, identify the interval the symbol is mapped on and update l and h by the rule:

$$l(x_1, x_2...x_n) = l(x_1, x_2...x_{n-1}) + l(x_n) * q(x_1, x_2...x_{n-1}) \tag{5.5}$$

$$h(x_1, x_2...x_n) = l(x_1, x_2...x_{n-1}) + h(x_n) * q(x_1, x_2...x_{n-1}) \tag{5.6}$$

  in which l_i and h_i are the lower bound and upper bound we designed in partitioning stage and $q(x_1, x_2...x_{i-1})$ is the size of the current interval calculated by

$$q(x_1, x_2...x_{i-1}) = \prod_{k=1}^{i-1} q(x_k) = \prod_{i=k}^{i-1} [p(x_k)(1-d) + d] \tag{5.7}$$

- After all blocks are encoded, choose a real number between l and h and convert it into a binary codeword with length $\lceil -\log_2 q(x_1, x_2...x_n) \rceil$

5.1.3 Decoding Process

The basic rule in the decoding process is similar to that of binary DAC, but the complexity is much higher than that found in binary cases.

The decoding process is conducted by setting up a decoding tree, when an ambiguity occurs, the tree splits into several branches to record all possible paths. In every step of decoding, the codeword is converted into a real number between 0 and 1, denoted as Z. The objective is to find all the intervals that cover Z.

In the partitioning stage, the interval of a symbol is characterized by l_i and h_i, which have formulations

$$\begin{aligned} l_i &= \sum_{k=0}^{i-1} p(x_k)(1-d) \\ h_i &= \sum_{k=0}^{i} p(x_k)(1-d) + d \end{aligned} \tag{5.8}$$

we can have a convenient way to determine which intervals cover Z to compare Z with each l_i and h_i. Denote the smallest interval that covers Z as S_{min} and the largest as S_{max}. It is not difficult to distinguish that intervals from S_{min} to S_{max} all cover Z. S_{min} and S_{max} can be calculted by :

$$\begin{aligned} S_{min} &= \arg\min_i h_i > Z \\ S_{max} &= \arg\max_i l_i < Z \end{aligned}$$

The decoding tree then splits into several branches to record symbols from S_{min} to S_{max} and decode the symbol in every step.

Algorithm 3 Multi-interval DAC decoding process

```
Initialize the root node T_0
while i < N do
  for All nodes in T_i do
    (S_min, S_max) = Test(T_i)
    for x_{i+1} = (S_min, S_max) do
      n_{i+1} = decode(x_{i+1})
      T_{i+1} = T_{i+1} + n_{i+1}
    end for
  end for
  Sort the nodes in T_{i+1} by side information
  Keep M most reliable nodes in every T_{i+1}
end while
```

The weight of each path is defined as the correlation between source X and side information Y using MAP metrics. The objective is to identify the most reliable path which has the largest correlation with side information. This can be interpreted by

$$
\begin{aligned}
(\hat{x}_1, \hat{x}_2 ... \hat{x}_n) &= \arg\max_{(x_1, x_2 ... x_n)} p(x_1, x_2 ... x_n | (y_1, y_2 ... y_n)) \\
&= \arg\max_{(x_1, x_2 ... x_n)} \prod_{i=1}^{n} P(x_i | y_i)
\end{aligned}
\tag{5.9}
$$

In implementation, M-algorithm is applied. M-algorithm is an algorithm to keep M reliable paths while decoding the codeword, and when the number of leaf nodes is larger than M, pruning the tree by deleting less probable paths. Essentially, it is a tradeoff between complexity and the probability of error.

The termination policy is also applied in our proposed algorithm. For a finite length sequence, we find that the final T symbols are more likely to be decoded incorrectly. Therefore, the last T symbols are encoded and decoded in traditional arithmetic coding instead of DAC. As a result, final T symbols are decoded without errors. Although it will have some rate loss, this policy guarantees performance with reliable metrics in the final T symbols and reduces complexity by not expanding the leaf nodes in the final T steps.

5.2 Huffman Coded Distributed Arithmetic Coding (HC-DAC)

Another proposed algorithm for non-binary DAC is called Huffman Coded Distributed Arithmetic Coding. This is based on the fact that DAC has a relatively good performance for short length binary sequences, especially in uniformly distributed Bernoulli case. When dealing with input taking values in a large alphabet, the most intuitive way to compress the source using DAC is to convert the input into a binary sequence in a quantization algorithm and compress the binary sequence in binary DAC. However, after the quantization, the correlation between input source X and side information Y may result in loss, binary DAC cannot be conducted without side information.

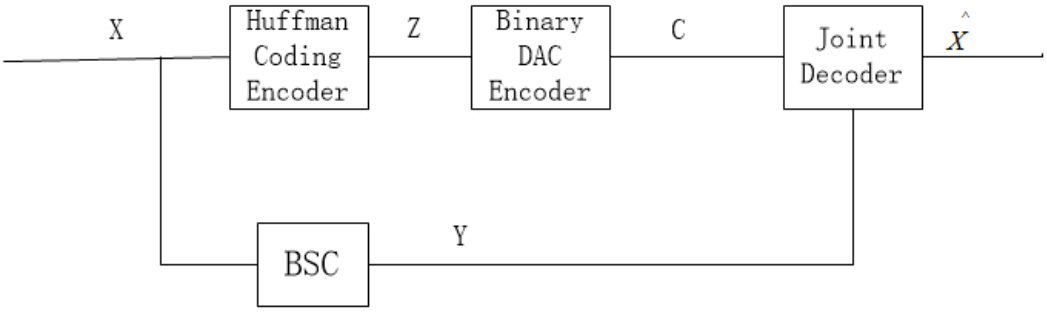

Figure 5.4: Structure of Huffman Coded DAC

5.2.1 Encoding Process

The structure of our proposed algorithm is shown in figure 5.4, an i.i.d discrete source input $X_1, X_2...X_N$ takes values in a non-binary alphabet A in a distribution $P(X)$. The side information Y is the result of X going through a channel with crossover probability ϵ. The encoding process is described below:

- First, source input X is encoded by Huffman coding. As is well known to all, Huffman coding is a lossless entropy coding which can compress an input sequence into a binary sequence with uniform distribution. The non-binary input X is compressed in its entropy $H(X)$, and the output sequence Z is a uniformly distributed binary sequence with an average length $N_z = N_x * H(X)$.

- In the second step, the sequence Z is encoded using binary DAC with binary uniform distribution. The rate for binary DAC is designed as $R = \log_2 q$, q is the enlarged interval size. After DAC encoding, the codeword is $\mathbf{C}$, which is transmitted to the decoder, and the rate for the encoding process equals to $-\log_2 q * H(X)$

5.2.2 Decoding Process

The decoding process is similar to the DAC decoding process. The decoding tree is configured to identify the ambiguity we insert at the encoding stage with side information.

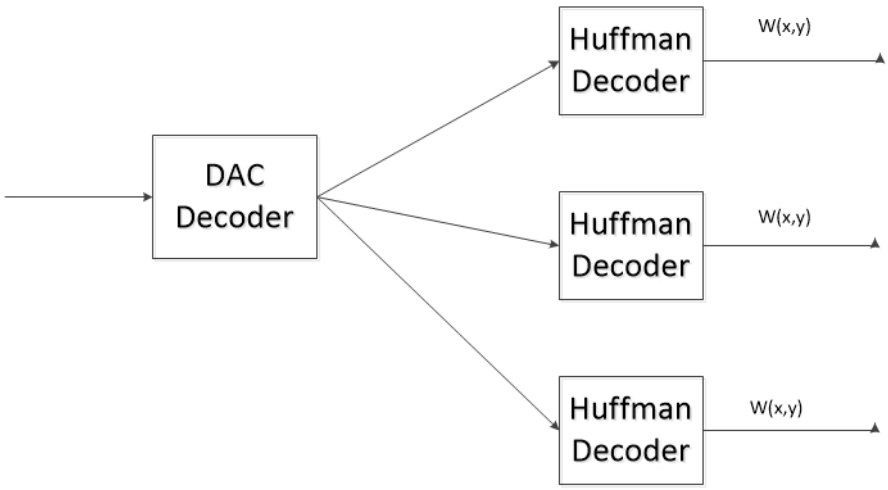

Figure 5.5: Structure of Huffman Coded DAC decoding process

There is an extra step in the Huffman-Coded DAC decoding process, which is to convert binary sequence candidates into original source input X by Huffman decoding, and apply side information Y to compute the weight and recognize the most reliable paths.

The detailed decoding process procedure is described below:

- Initialize the interval overlap design the same as the encoding stage, then read the codeword $c_1c_2...c_k$ and convert it into a real number Z between 0 and 1.
- Determine which interval Z belongs to, if Z is covered by more than 1 interval, the tree splits into two branches to account for each possible symbol. All possible paths are kept in each step.
- To continue decoding, the intervals need to be updated based on the current symbol, and redivided to the subintervals recursively until all symbols are decoded.
- M-algorithm is applied. If the number of leaf nodes is larger than M, the tree is trimmed by remaining the most reliable M paths. It is necessary to convert all paths into an original X sequence and use side information Y to compute the weight of

every path. It must be noted that Huffman coding is varies in length: the decoding result of same length codewords may result in different length X sequence. In order to maintain a fair comparison, we record the length of the shortest symbol sequence L_{min} and compute weight of every path by the length L_{min}.

The decoding metrics are calculated by the result of Huffman decoding $\hat{X}$ and side information Y by

$$\begin{aligned}(\hat{x}_1, \hat{x}_2 ... \hat{x}_n) &= \arg\max_{(x_1, x_2 ... x_n)} p(x_1, x_2 ... x_n | (y_1, y_2 ... y_n)) \\ &= \arg\max_{(x_1, x_2 ... x_n)} \prod_{i=1}^{n} P(x_i | y_i)\end{aligned} \tag{5.10}$$

The termination policy is applied since the final T symbols for DAC are more likely to be decoded incorrectly. We encode and decode the final T symbols DAC using traditional arithmetic coding to ensure accuracy in decoding. Although it will cause small amount of rate loss, it can be considered neglectable in case of long sequences. T is usually set as 15 or 20.

5.3 Performance Evaluation

The simulation is conducted in various correlations and distributions.

As we know, Huffman Coding is optimal for long sequence and some special distribution in short length sequences, otherwise, the result may not be an exactly uniform Bernoulli distribution possibly resulting in some loss of binary uniform DAC. This may have a negtive impact on performance. Therefore, our simulation will consider all aspects through different distributions and different correlations in 4-ary and 8-ary sources. In the simulation, the X axis shows the change in compression rate and the Y axis shows the probability of symbol error, which is the probability that a symbol in an input sequence is decoded incorrectly.

5.3.1 4-ary Source Input Simulation

4-ary lossless source input in Huffman Coding

In this section, discrete random variable X takes values from a 4-ary alphabet A with distribution in Table 5.1.

X	0	1	2	3
p	0.5	0.25	0.125	0.125

Table 5.1: Diadic distribution of input X

In this set-up, entropy $H(X)$ is 1.75. The result of Huffman Coding is an exactly uniform distributed binary sequence

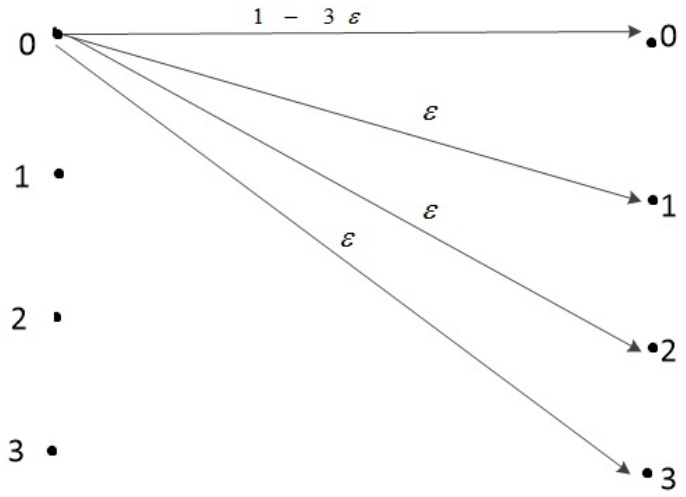

Figure 5.6: Symmetric channel model

The side information Y is the result of source X going through a symmetric channel with a crossover probability ϵ. The model of symmetric channel is shown in Fig 5.6: every symbol x_i has the same probability ϵ to change value into the other 3 values. A different ϵ is selected to control the correlation between source and side information.

The simulation is conducted in two different correlations. In strongly correlated situations, ϵ is set as 0.05, in which $p(x_i = y_i) = 0.85.$, while in weakly correlated situations, ϵ is set as 0.1 so that $p(x_i = y_i) = 0.7$.

The result of a strongly correlated source simulation is shown in Fig 5.7.

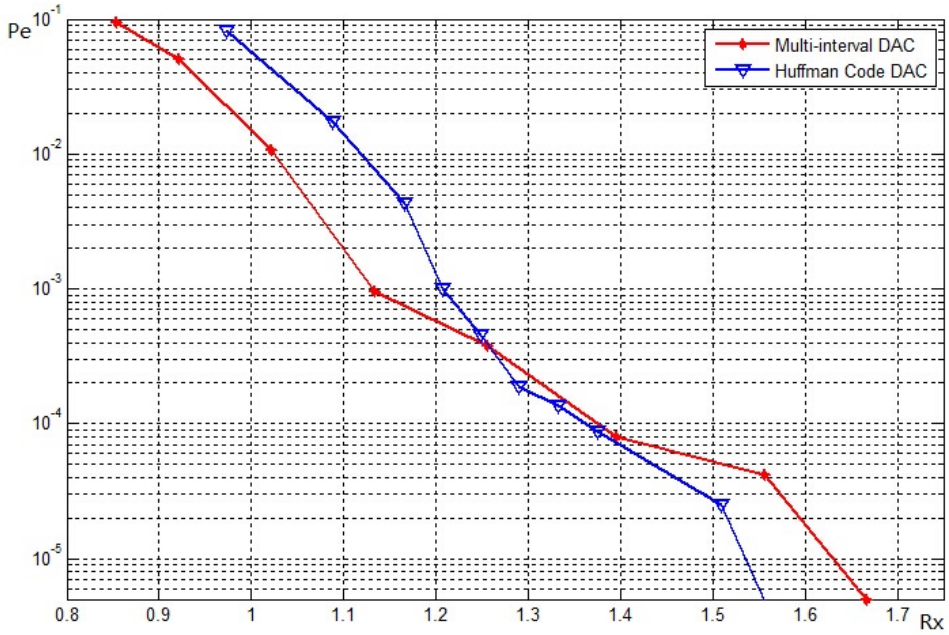

Figure 5.7: Error probability for 4-ary diadic source with strongly correlated side information. $(H(X|Y) = 0.7581)$

In Fig 5.7 Multi-interval DAC outperforms Huffman Coded DAC when the compression rate is low. It is capable of handling certain situations which the Huffman Coded DAC cannot decode correctly. However, the Huffman Coded DAC is falling in a faster rate when the compression rate increases and the performance is better than Multi-interval DAC when the rate approaches the entropy. In the decoding process of Huffman Coded DAC, every candidate path is decoded in Huffman coding to compare the weight of each path. Intuitively it is less likely to be decoded incorrectly when the compression rate is high.

The result of simulations on weakly correlated source sequences is shown in Fig 5.8.

In Fig 5.8 the performance of the two proposed algorithms are more comparable. The trend is the same as that found in the strongly correlated situation: when the rate is low, Multi-interval DAC outperforms Huffman-Coded DAC. The probability of error of Huffman-Coded DAC is falling more quickly and shows a better performance when the rate is closer to the entropy. The difference is smaller than that found in the strongly correlated situation, which indicates that when the correlation decreases, the influence of

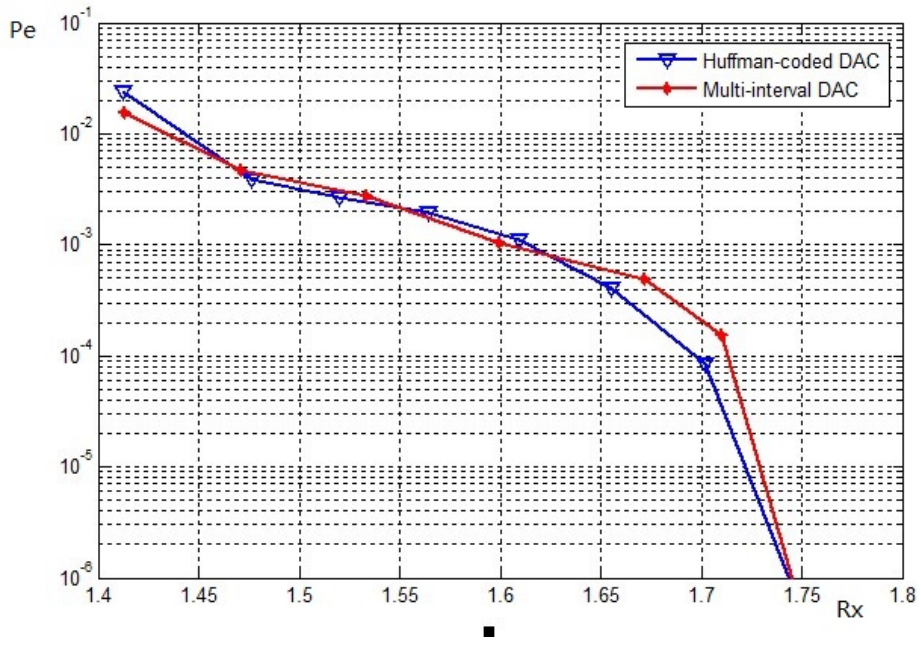

Figure 5.8: Error probability for 4-ary diadic source with weakly correlated side information. ($H(X|Y) = 1.1979$)

the performance of Huffman-Coded DAC is larger than that of Multi-interval DAC.

4-ary non-diadic source input in Huffman Coding

Evidently, the distribution is a key factor for interval design in traditional arithmetic coding so that if there is a small discrepancy between the distribution of input and that in the arithmetic coding model, it may have some rate loss. Therefore, the Huffman Coded DAC is more likely to have a loss in performance since the input in DAC is not an exactly uniform distributed binary source. In this section, we are going to explore how much loss we have in this situation.

In order to facilitate a fair contrast, the simulation conditions are the same as that in a lossless case. The performance of the two proposed algorithms is shown in strongly correlated sequences and weakly correlated sequences. In the simulation, the input source discrete random variable X takes value from a 4-ary alphabet A in a distribution in Table 5.2.

In this arrangement, the entropy $H(X)$ is 1.6855. Y is the result of X going through a

X	0	1	2	3
p	0.5	0.3	0.1	0.1

Table 5.2: Non-diadic distribution of source input X

symmetric channel with crossover probability ϵ, and the source X is encoded and decoded in the two proposed algorithms. The result is shown in Fig 5.9.

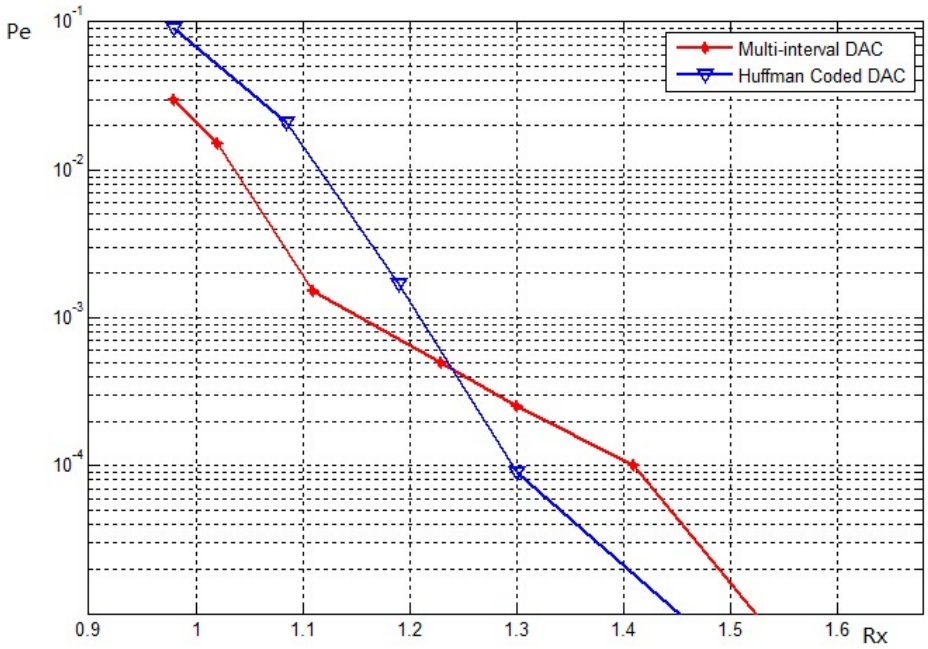

Figure 5.9: Error probability for 4-ary non-diadic source with strongly correlated side information. $(H(X|Y) = 0.7315)$

In Fig 5.9 the performance is similar to that of strongly correlated sequences in lossless data found in Huffman coding source. The performance of Mulit-interval DAC is better than Huffman-Coded DAC when the rate is low, conversely, when the rate is increases, the probability of error of the Huffman-Coded DAC drops more rapidly. When rate is around 1.25 the performance of the two proposed algorithms is nearly identical and when the rate is larger than 1.25 and approaching entropy, the performance of Huffman-Coded DAC is better until it reaches entropy,in which case both algorithms achieve a zero error performance.

In these figures it is evident that the loss in Huffman coding has little impact on the performance when the correlation is high. The phenomenon can be explained by the

overlap rule design we discussed above. In contrast to arithmetic coding, the method of designing the overlap has little impact on the performance in binary DAC. This is due to the suboptimal designs of all overlap rules without minimizing the probability of error at the decoder. Essentially, it is a suboptimal process. Although the input of the DAC encoder is not exactly in a uniform binary distribution, the overlap factor can still handle the situation with little loss in performance.

When the correlation between the source and side information is weak, the performance is as shown in Fig 5.10.

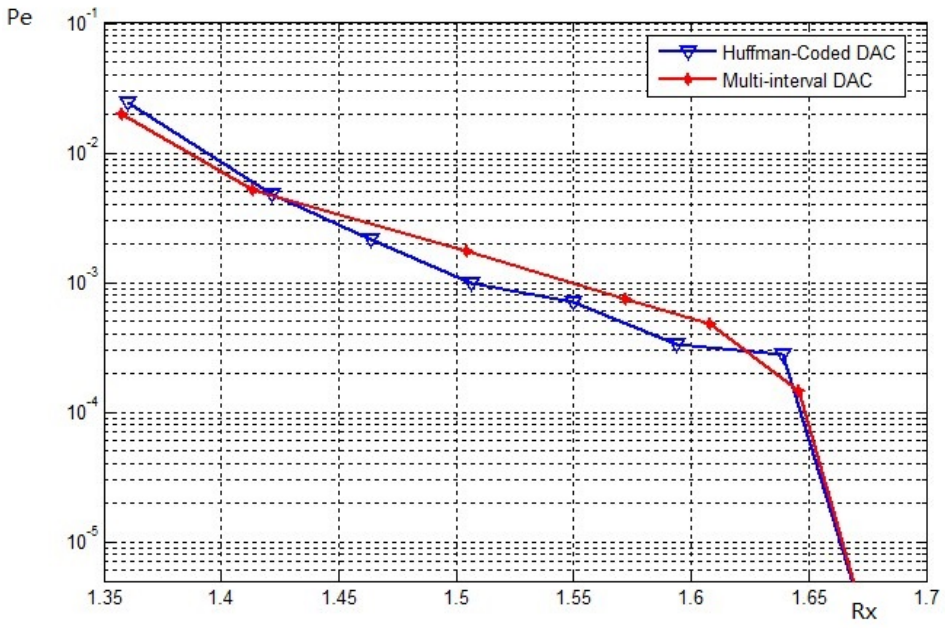

Figure 5.10: Error probability for 4-ary non-diadic source with weakly correlated side information. $(H(X|Y) = 1.1532)$

In Fig 5.10 the performance of the two proposed algorithms is nearly identical to that of the lossless source in Huffman coding. The performance of Multi-interval DAC surpasses that of Huffman-Coded DAC when the compression rate is low, while Huffman-Coded DAC improves as the rate increases with the performance of the two algorithms are equal when the rate achieves entropy.

From the two figures above, it is apparent that the loss in Huffman coding has little impact on the probability of error the 1 bit loss in Huffman coding can be repaired by the

overlap rule design in DAC. As all overlap rule designs are suboptimal, their performance are nearly identical Therefore, the loss in Huffman Coding can be ignored when encoded and decoded in binary DAC.

5.3.2 Time Complexity In 4-ary Simulation

The complexity of the two proposed algorithms can be measured in the operation time of the encoding and decoding process since DAC is proposed for wireless sensor networks which is in favor of an algorithm with a simple encoding process and greater accuracy. For Multi-interval DAC, the larger the overlaps are, the slower the program functions while in Huffman Coded DAC, the complexity involves both the rate and the distribution of the source. In this section, we are going to compare the complexity of the two proposed algorithms through their operating time.

The operation time for the encoding process is listed in Table 5.3, and each time records the amount of time used to encode a length $N = 200$ sequence.

Algorithm	Operation time(s)
Multi-interval DAC	0.0061
Huffman-Coded DAC	0.0074

Table 5.3: Operation time of encoding process

In Table 5.3 both algorithms have a low complexity encoding process and Multi-interval DAC requires less time since Huffman coding is conducted first in Huffman-Coded DAC, although the variation is not large enough to make an impact. The complexity of the decoding process is shown in Table 5.4 and Fig 5.11.

Algorithm	Operation time(s)					
Rate	0.9	1.1	1.3	1.5	1.7	1.75
Multi-interval DAC	0.396	0.358	0.2818	0.2528	0.02	0.004
Huffman-Coded DAC	11.02	10.19	9.85	9.22	3.01	0.01

Table 5.4: Operation time of decoding process

In Fig 5.11, the complexity decreases when the rate grows larger. Multi-interval DAC runs much faster than Huffman Coded DAC, especially when the rate is low, since the

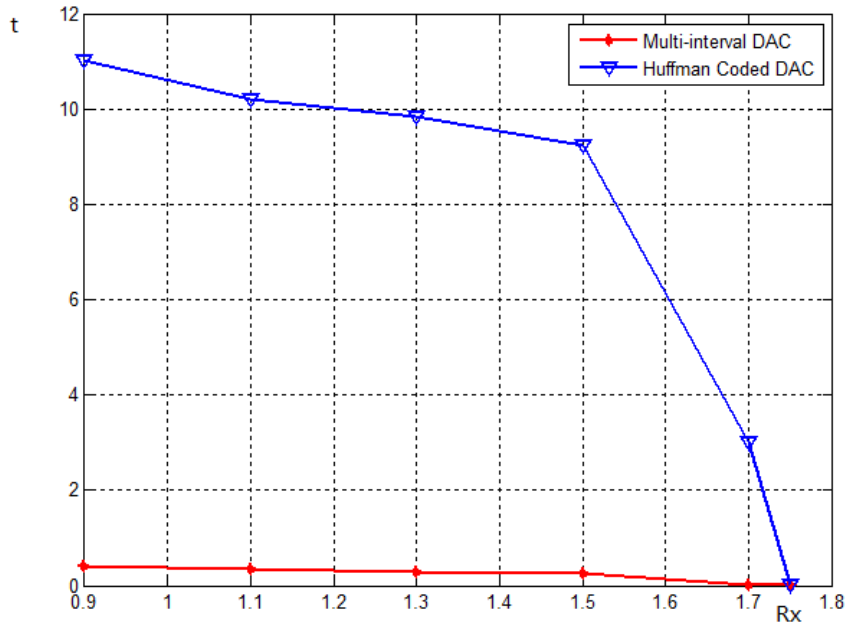

Figure 5.11: Operation time of two proposed algorithms in 4-ary simulation

differential is approximately 300 times when the rate is low. As rate grows larger, the operation time of the Huffman Coded DAC quickly decreases When the rate achieves entropy, the operation time of the two algorithms are nearly equivalent. Therefore, from this perspective Multi-interval DAC is superior to Huffman Coded DAC.

5.3.3 8-ary Source Input Simulation

The discrete random variable X takes values from a 8-ary alphabet A with a distribution listed in Table 5.5.

X	0	1	2	3	4	5	6	7
p	1/2	1/4	1/8	1/16	1/32	1/64	1/128	1/128

Table 5.5: Distribution of source input X

In this set up, the entropy of $XH(X) = 1.9844$. Y is the result of X going through a symmetric channel with crossover probability ϵ, which is a parameter to control the correlation between source input and side information. The simulation is conducted in two

different correlations in high correlation, $\epsilon = 0.02$ which leads to $p(y_i = x_i) = 0.86$ while in weakly correlated situations, ϵ is set as 0.05 so that $p(y_i = x_i) = 0.65$.

The result of strongly correlated sources simulation is shown in Fig 5.12.

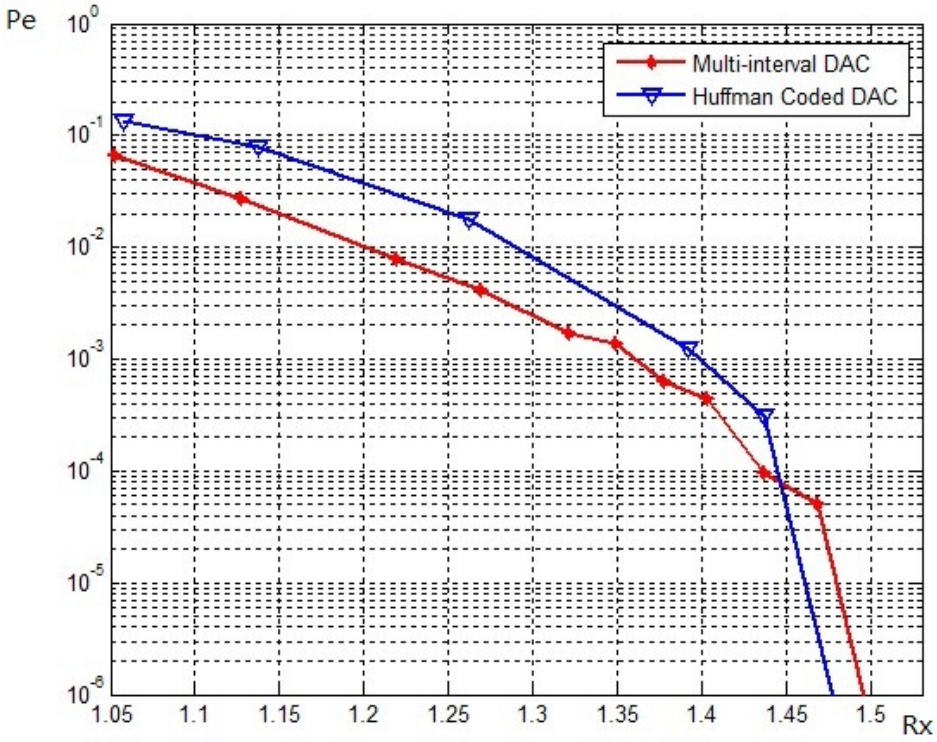

Figure 5.12: Error probability for 8-ary source with strongly correlated side information. $(H(X|Y) = 0.6573)$

In Fig 5.12, the probability of error is decreasing as the rate grows. Multi-interval DAC outperforms Huffman Coded DAC when the rate is low, while the probability of error of Huffman Coded DAC falls more quickly when it is below 10^{-4}. When the rate arrives at 1.45, Huffman Coded DAC performs better when the compression rate is near entropy.

The progression of the two lines is similar to that of the 4-ary case in multiple ways. Multi-interval DAC has a better performance when the rate is low. However, when the probability of error is below 10^{-4}, the error probability decreases at a slow speed. Huffman Coded DAC seems to display difficulty in decoding correctly when the rate is low; however, it can achieve a lower error probability when the rate is near entropy.

The result of weakly correlated source simulation is shown in Fig 5.13.

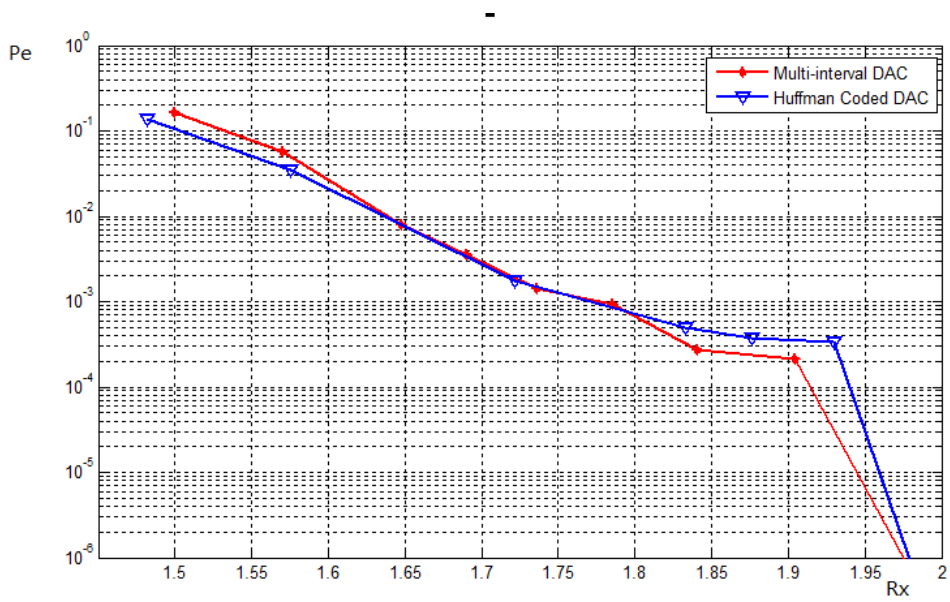

Figure 5.13: Error probability for 8-ary source with weakly correlated side information. ($H(X|Y) = 1.258$)

In figure 5.13 the performance of the two proposed algorithms is more comparable. Both algorithms cannot achieve a satisfactory performance for low correlated sources, and the performance of Multi-interval DAC is slightly superior to that of Huffman Cocded DAC when the rate is slightly below entropy. As a result, for low correlated sources, the performance of the two algorithms are nearly identical.

5.3.4 Time Complexity In 8-ary Simulation

The complexity of the two proposed algorithms is compared through their operation time in 8-ary simulations.

First, the operation time of encoding process for the two algorithms is listed in Table 5.6. Each records the amount of time used to encode a length N=200 sequence:

Algorithm	Operation time(s)
Multi-interval DAC	0.0063
Huffman-Coded DAC	0.0074

Table 5.6: Operation time of encoding process

In Table 5.6 both algorithms have a simple encoding process: the sequence is encoded

in a short time in the two algorithms. Not surprisingly the Huffman Coded DAC requires a little more time to encode since the encoding process of Huffman Coding may have a small amount of loss.

The complexity in the decoding process is compared through the operation time of the two algorithms. As the operation time varies by different rates, the operation time of each rate is recorded to create a fair contrast. The operation time of the two algorithms is listed in Table 5.7 and Fig 5.14. Each records the time to decode a length $N = 200$ sequence and the X axis records the change in rate while the Y axis represents the time in seconds.

Algorithm	Operation time(s)					
Rate	1.05	1.25	1.5	1.7	1.9	1.9844
Multi-interval DAC	0.625	0.512	0.39	0.318	0.1578	0.004
Huffman-Coded DAC	8.16	7.533	7,134	6.804	0.820	0.004

Table 5.7: Operation time of decoding process

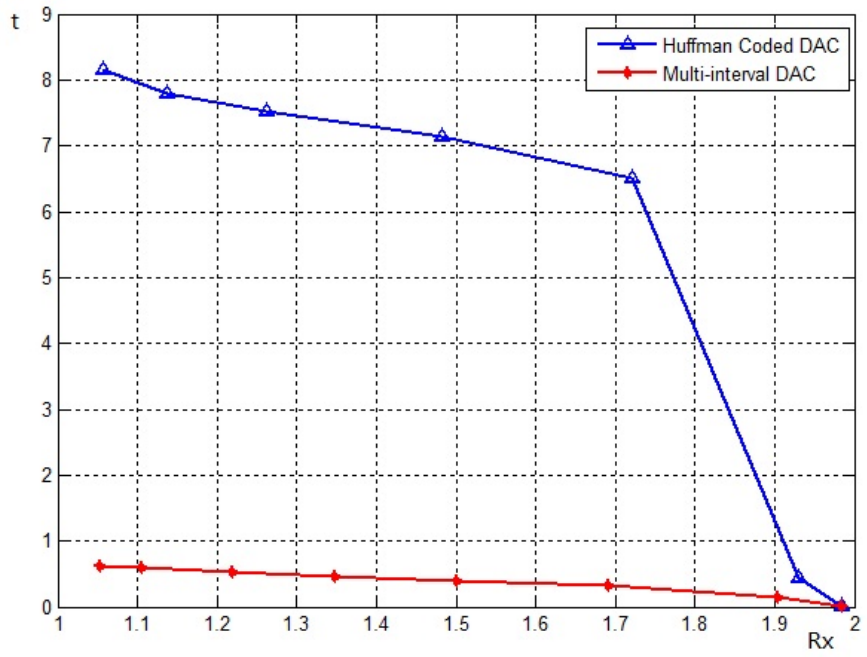

Figure 5.14: Operation time of decoding process

In Fig 5.14 the operation time for Multi-interval DAC is much smaller than that of Huffman Coded DAC, especially when the rate is low. The difference may be as large as

300 times. As the rate increases, the operation time for Huffman Coded DAC decreases rapidly and the operation time for both algorithms are nearly the same when rate arrives at entropy.

5.4 The decoding complexity in HC-DAC

It is noticed that the complexity for the decoding process of HC-DAC is fairly high with $M = 2048$, and it makes it impractical for applications in WSN. In this section, a study on various selection on M is conducted through simulations, and it may give a better tradeoff on the complexity and probability of error.

X and Y are two random variables taking values from a 8-ary alphabet A with a distribution listed in Table 5.8. Y is the result of X going through a symmetric channel

X	0	1	2	3	4	5	6	7
p	1/2	1/4	1/8	1/16	1/32	1/64	1/128	1/128

Table 5.8: Distribution of source input X

with crossover probability ϵ, and $\epsilon = 0.02$ refers to strongly correlated sources while $\epsilon = 0.05$ for weakly the correlated sources. M is selected as 256, 512, 1024 and 2048. The simulation results are shown in Fig 5.15 and Fig 5.16:

In Fig 5.15, the results are expected: the performance with $M = 2048$ is superior to the other three arrangements, and the differences become larger when the rate increases. It is also noticed that the probability of error in $M = 1024$ simulation is close to that of $M = 2048$, especially when the rate is low.

In Fig 5.16, it is interesting to notice that the four arrangements have nearly identical performances. Different selection on M does not change the performance much in this situation.

The complexity of the decoding is evaluated through the operation time:

In Table 5.9, the complexity of the decoding process decreases when M is selected as a smaller number. The operation time (per codeword) increase nearly exponentially with the

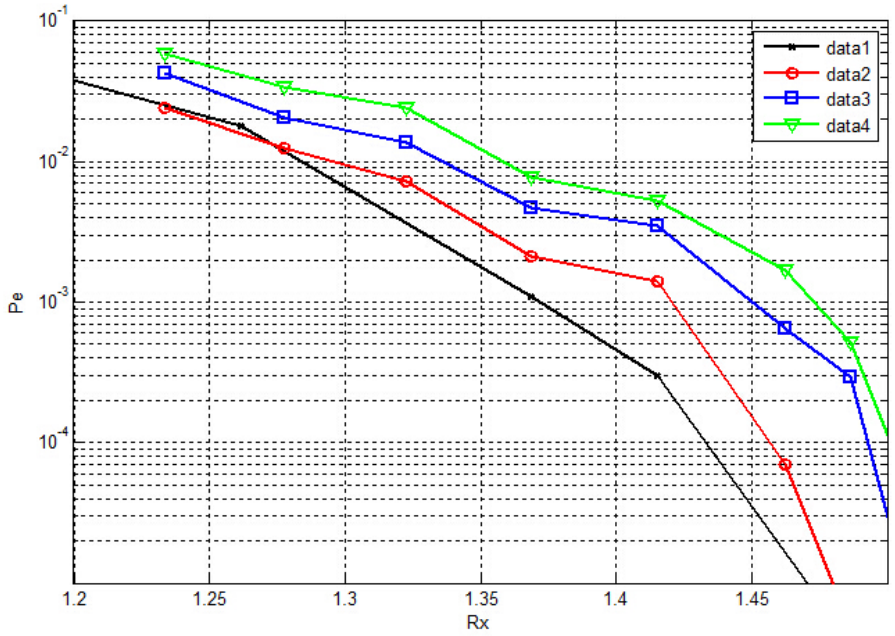

Figure 5.15: Error probability for 8-ary source with strongly correlated side information. $(H(X|Y) = 0.6573)$

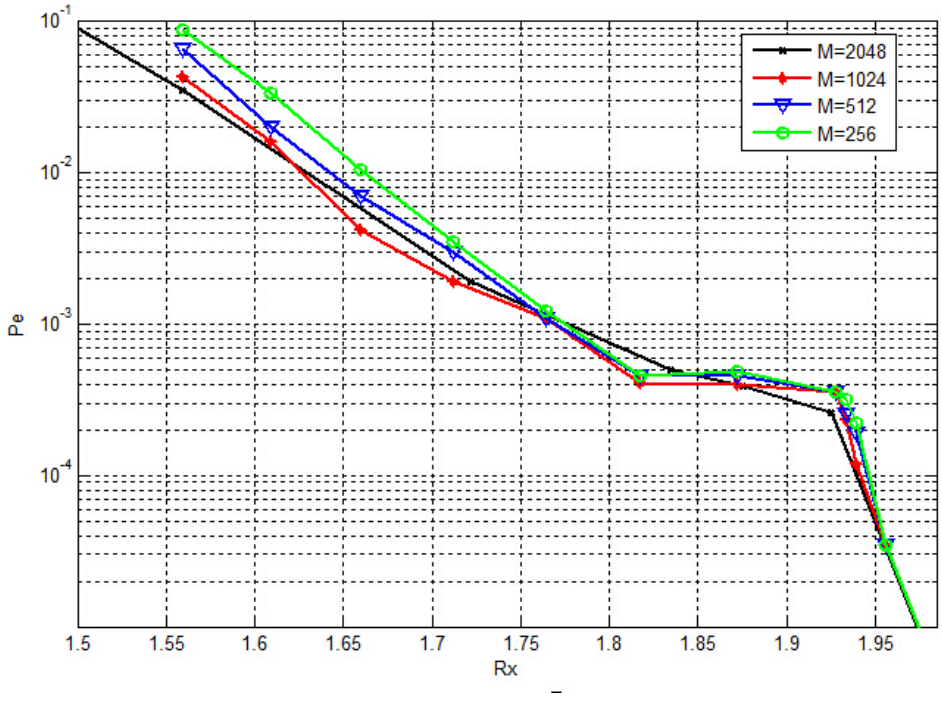

Figure 5.16: Error probability for 8-ary source with weakly correlated side information. $(H(X|Y) = 1.258)$

Algorithm	Operation time(s)					
Rate	1.05	1.25	1.5	1.7	1.9	1.9844
M=2048	8.16	7.533	7,134	6.804	0.820	0.004
M=1024	3.881	3.651	3.42	3.04	0.679	0.004
M=512	2.227	1.902	1.723	0.80	0.517	0.004
M=256	0.997	0.93	0.88	0.60	0.355	0.004

Table 5.9: Operation time of the decoding process on different selection of M

selection on M when the rate is low, and it becomes much closer when the rate increases.

The performance and complexity evaluation on different selection of M in HC-DAC may give us an approach to apply different M to different R_x to have a better tradeoff of the performance and the decoding complexity: $M = 256$ appears to be more efficient for weakly correlated sources; while M can also be selected as a small number when the rate is low for strongly correlated sources to reduce the complexity.

5.5 Discussion and Conclusion

Two algorithms are proposed regarding to non-binary DAC and the performance of two algorithms are simulated in different situations.

Both algorithms achieve a good performance. The performance of two algorithms are nearly the same for weakly correlated source sequences, while a little differential in performance exists for strongly correlated source sequences. Multi-interval DAC can achieve a good performance when rate is low. However, the performance of Huffman Coded DAC is falling more quickly when rate grows larger and outperforms Multi-interval DAC when rate arrives at a location near entropy.

In Huffman Coded DAC, the candidates in binary DAC are all decoded by Huffman Coding to compute the weight. This process ensures the accuracy by two decoding algorithm. In some situation of binary DAC, the two alike candidates are decoded. They may result in two totally different configurations in Huffman Codes. This is the reason why Huffman Coded DAC is able to achieve a good performance when the compression rate is high.

As for the complexity of encoding and decoding process, which is of great interest in Slepian-Wolf problem. Both algorithms have a very short encoding process, since the encoding process for two proposed algorithms are both based on DAC encoding which has a linear complexity. As for complexity of decoding process, Multi-interval DAC is a lot superior to Huffman Coded DAC.

From my point of view, the result is reasonable. Both of the decoding process of the two approaches are based on a suboptimal tree structure to find out the most likely sequence. M is the maximum number of leaf nodes that are allowed in the decoding tree. I suspect the differential in the performance is largely determined by the length of non-binary sequences when the number of leaf nodes first arrives at M. The number of leaf nodes in decoding tree of binary DAC grows exponentially when the rate is low. Therefore, the number of symbols in non-binary sequence in HC-DAC is smaller than that of MI-DAC so that MI-DAC is more reliable. When rate grows larger, the number symbols in the sequence when the number of leaf nodes arrives at M in MI-DAC is smaller than HC-DAC. It is more likely that the tree in binary DAC is growing in quite a slow speed. Therefore, the performance of HC-DAC is superior when the rate is high.

Chapter 6

Conclusion

The thesis presents a thorough study on the Distributed Arithmetic Coding(DAC) scheme. An investigation on the impact of the overlap factor in binary DAC is conducted through simulations on different structure of overlap designs. The result shows the performance can not be optimized by the overlap factor in binary DAC. By intuition, the overlap factor is likely to have a larger impact on non-binary DAC, which is left as future work.

In chapter 5, two algorithms regarding to the non-binary DAC scheme are proposed. The first algorithm is an extension of non-binary arithmetic coding with a specific structure of the overlap design, while the second algorithm is to compress non-binary source by converting a non-bianry sequence into a binary sequence in Huffman coding and encoding the result in binary DAC. The simulation results of these two algorithms on 4-ary and 8-ary souces illustrate that both algorithms can be used in the applications in WSNs. Huffman Coded DAC is more accuracy in a high rate compression while Multi-interval DAC provides less complexity in the encoding and decoding processes.

References

[1] Anne Aaron and Bernd Girod. Compression with side information using turbo codes. In *Data Compression Conference, 2002. Proceedings. DCC 2002*, pages 252–261. IEEE, 2002.

[2] Jan Bajcsy and Patrick Mitran. Coding for the slepian-wolf problem with turbo codes. In *Global Telecommunications Conference, 2001. GLOBECOM'01. IEEE*, volume 2, pages 1400–1404. IEEE, 2001.

[3] Thomas M Cover. A proof of the data compression theorem of slepian and wolf for ergodic sources (corresp.). *Information Theory, IEEE Transactions on*, 21(2):226–228, 1975.

[4] Yong Fang and Liang Chen. Improved binary dac codec with spectrum for equiprobable sources. *Communications, IEEE Transactions on*, 62(1):256–268, 2014.

[5] Javier Garcia-Frias. Compression of correlated binary sources using turbo codes. *Communications Letters, IEEE*, 5(10):417–419, 2001.

[6] Bernd Girod, Anne Margot Aaron, Shantanu Rane, and David Rebollo-Monedero. Distributed video coding. *Proceedings of the IEEE*, 93(1):71–83, 2005.

[7] Marco Grangetto, Enrico Magli, and Gabriella Olmo. Symmetric distributed arithmetic coding of correlated sources. In *Multimedia Signal Processing, 2007. MMSP*

2007. IEEE 9th Workshop on, pages 111–114. IEEE, 2007.

[8] Marco Grangetto, Enrico Magli, and Gabriella Olmo. Distributed arithmetic coding for the slepian–wolf problem. *Signal Processing, IEEE Transactions on*, 57(6):2245–2257, 2009.

[9] Marco Grangetto, Enrico Magli, Roberto Tron, and Gabriella Olmo. Rate-compatible distributed arithmetic coding. *IEEE Communications Letters*, 12(8):575–577, 2008.

[10] Xun Guo, Yan Lu, Feng Wu, Wen Gao, and Shipeng Li. Distributed multi-view video coding. In *Electronic Imaging 2006*, pages 60770T–60770T. International Society for Optics and Photonics, 2006.

[11] Joachim Hagenauer. The turbo principle: Tutorial introduction and state of the art. In *Proc. International Symposium on Turbo Codes and Related Topics*, pages 1–11, 1997.

[12] Paul G Howard and Jeffrey Scott Vitter. Analysis of arithmetic coding for data compression. *Information processing & management*, 28(6):749–763, 1992.

[13] David A Huffman et al. A method for the construction of minimum redundancy codes. *proc. IRE*, 40(9):1098–1101, 1952.

[14] Naoto Kimura and Shahram Latifi. A survey on data compression in wireless sensor networks. In *Information Technology: Coding and Computing, 2005. ITCC 2005. International Conference on*, volume 2, pages 8–13. IEEE, 2005.

[15] Frank R Kschischang, Brendan J Frey, and H-A Loeliger. Factor graphs and the sum-product algorithm. *Information Theory, IEEE Transactions on*, 47(2):498–519, 2001.

[16] Madhurima Potluri. *Distributed source coding using non binary LDPC.* PhD thesis, Wichita State University, 2005.

[17] Madhurima Potluri, Suma Chilumuru, Sumakanth Kambhampati, and Kamesh R Namuduri. Distributed source coding using non-binary ldpc codes for sensor network applications. In *Information Theory, 2007. CWIT'07. 10th Canadian Workshop on*, pages 105–109. IEEE, 2007.

[18] S Sandeep Pradhan and Kannan Ramchandran. Distributed source coding using syndromes (discus): Design and construction. *Information Theory, IEEE Transactions on*, 49(3):626–643, 2003.

[19] Claude Elwood Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001.

[20] Amin Shokrollahi. Ldpc codes: An introduction. In *Coding, cryptography and combinatorics*, pages 85–110. Springer, 2004.

[21] David Slepian and Jack K Wolf. Noiseless coding of correlated information sources. *Information Theory, IEEE Transactions on*, 19(4):471–480, 1973.

[22] Jagadish Venkataraman. Analysis of slepian wolf coding. *Electrical Engineering*, page 111, 2003.

[23] Zunliang Wang, Xueming Li, and Min Zhao. Distributed coding of gaussian correlated sources using non-binary ldpc. In *Image and Signal Processing, 2008. CISP'08. Congress on*, volume 2, pages 214–218. IEEE, 2008.

[24] IANH WIllEN, Radford M Neal, and John G Cleary. Arithmetic coding for data coiupression. *Communications of the ACM*, 30(6), 1987.

[25] Aaron D Wyner. On source coding with side information at the decoder. *Information Theory, IEEE Transactions on*, 21(3):294–300, 1975.

[26] Zixiang Xiong, Angelos D Liveris, and Samuel Cheng. Distributed source coding for sensor networks. *Signal Processing Magazine, IEEE*, 21(5):80–94, 2004.

[27] Mahdi Zamani and Farshad Lahouti. Distributed source coding using symbol-based and non-binary turbo codes-applications to wireless sensor networks. *Communications, IET*, 2(8):1089–1097, 2008.

[28] Junwei Zhou, K-W Wong, and Jianyong Chen. Distributed block arithmetic coding for equiprobable sources. *Sensors Journal, IEEE*, 13(7):2750–2756, 2013.

[29] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on information theory*, 23(3):337–343, 1977.

[30] Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *Information Theory, IEEE Transactions on*, 24(5):530–536, 1978.