Spring 5-2018

Computation of Achievable Rate in Pattern Recognition System

Yu Liu
Washington University in St. Louis

WASHINGTON UNIVERSITY IN ST. LOUIS

School of Engineering and Applied Science

Department of Electrical and System Engineering

Thesis Examination Committee:
Joseph O'Sullivan
Zachary Feinstein
Hiro Mukai

Computation of Achievable Rate in Pattern Recognition System

by

Yu Liu

A thesis presented to the School of Engineering
of Washington University in St. Louis in partial fulfillment of the
requirements for the degree of
Master of Science

May 2018

Saint Louis, Missouri

Contents

List of Figures

Acknowledgments

First, I want to thank my research advisor, Dr. Joseph O'Sullivan, who provided me elaborate instruction during each stage of my research. His patient advises always give me solution to my problems and without his guidance, I could not have finished my research. It is his vigorous and enthusiastic academic attitude that keep me working on this project. What he taught me and what he showed me from his behavior will encourage me to continue working hard in academic field. I promise I will devote myself into academic studies in the future and set being a researcher like him as my goal.

I would also like to say thanks to all the course instructors and advisors that give me various kinds of help during the two years in Washington University. All of them gave me significant help using their high-level teaching and research skills and keen to students. They are the people who impressed me deeply during my whole academic period till now. I would thank the Dean for his contribution to our department and the development for this thesis/dissertation template.

Finally, I appreciate many of the graduate students, my classmates and friends that give me support for my research. Special thanks to the faculty within my department who reviewed this thesis.

Yu Liu

Washington University in St. Louis
May 2018

Dedicated to my parents.

ABSTRACT OF THE THESIS

Computation of Achievable Rate in Pattern Recognition System

by

Yu Liu

Master of Science in Electrical Engineering

Washington University in St. Louis, 2018

Research Advisor: Professor Joseph O'Sullivan

Pattern recognition is widely used in many areas. When analyzing a pattern recognition system, one of the main problem to be considered is, how many bits do I need to express the raw source data and memory data to ensure that the result of the pattern recognition be reliable. The data stored in the system as well as the data received by the system must be compressed by some rate to summary the raw data. The fundamental bound for this lies in the computation of the achievable rate of pattern recognition. Before now, we have the definition and some approaches for this achievable rate region from an information theory point of view, but these approaches can be applied only to some specific cases. There's need for a method to compute this region's boundary and this method should be able to be extended to any general case.

In this thesis, we present a new optimization algorithm associated with other algorithms in alternating optimization problems. This new algorithm will compute a bound of the achievable rate region in pattern recognition by solving the associated optimization problem. We show that this new algorithm can solve the problem we have for computing the boundary of the achievable rate region and can be extended to other areas.

Chapter 1

Introduction

Pattern recognition refers to inferring the state of an environment from incoming and previously stored data [1]. However, in real-world, the volume of the stored data will exceed the capacity of the system's storage if the data is stored in raw. Thus, the data stored need to be compressed in a format that only summarize the main property of the raw data. Then, there's the tradeoff between the compress rate and the reorganization reliable [1]. This tradeoff leads to the main problem discussed in the thesis.

1.1 Pattern Recognition

The pattern recognition refers to the concept of finding the regularity of some input data and put them into some category according to the regularity. This concept is a branch of the machine learning and yet not the same. A simple example could be hand-writing recognition algorithm. Although this concept is connected to the machine learning and mathematical algorithms running by computers, this search for this can be tracked back to the 16th century when Johannes Kepler discovered the planetary motion's law. [5]. As human, we are functionally capable of doing this in our brains. We see different situations every day and can automatically classify these situations into distinct categories. In the machine learning area, we are trying to imitate this process using computer algorithms.
In this thesis, we will use the system diagram shown in Figure 1.1 as our pattern recognition system. We will discuss the meaning of each component in later sections.

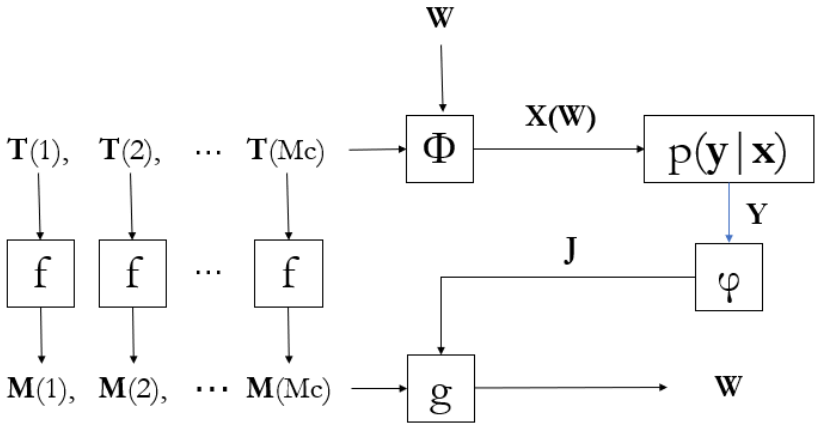

Figure 1.1 Pattern Recognition System

1.2 Achievable Rate Region

To discuss about the achievable rate region in pattern recognition system, we will first introduce the definition of achievable rate in information theory. The definition for pattern recognition is similar yet still have some difference.

1.2.1 Achievable Rate in Information Theory

To discuss the achievable rate in information theory, we first need to introduce serval definition related to this part. In this section, we will talk about the encoder and decoder in communication system, shown in Figure 1.2. The $\widehat{\mathrm{X}}^{\mathrm{n}}$ here is the estimation of X^{n} with $\widehat{\mathrm{X}}^{\mathrm{n}} \in \widehat{\mathcal{X}}$. The R in the coding function $f_n(X^n)$ is the rate we discuss in this section.

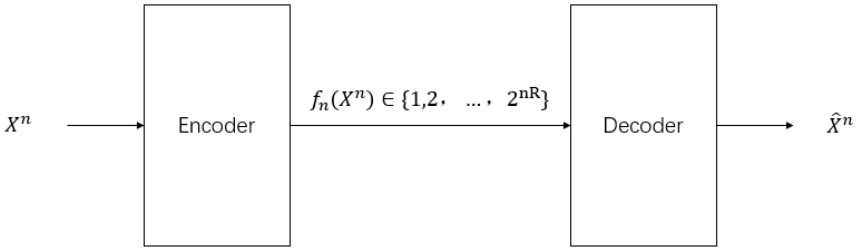

Figure 1.2 Rate distortion encoder and decoder

This will lead to the definition of distortion function. Distortion function is used to measure the cost of representing x by $\hat{x}$. There are many kind of distortion functions, serving different kind of communication channel. One common example of the distortion function is the Hamming distortion. The Hamming distortion is given by [4]

$$\mathrm{d}(\mathrm{x}, \hat{x}) = \begin{cases} 0 \; if \; x = \hat{x} \\ 1 \; if \; x \neq \hat{x} \end{cases} \tag{1.1}$$

Definition The distortion between sequences x^n and $\hat{x}^n$ is defined by [4]

$$\mathrm{d}(x^n, \hat{x}^n) = \frac{1}{n}\sum_{i=1}^{n} d(x_i, \hat{x}_i) \tag{1.2}$$

Though this is not the only way to describe the distortion between two series [4]. Then we can discuss about the distortion code and distortion D associated with the distortion code.

Definition A $(2^{\mathrm{nR}}, \mathrm{n})$-rate distortion code contains an encoding function f_n and a decoding function g_n, which are defined by

$$f_n: \mathcal{X}^n \rightarrow \{1,2, \dots, 2^{nR}\}$$

$$g_n: \{1,2, \dots, 2^{nR}\} \rightarrow \widehat{\mathcal{X}}^n \tag{1.3}$$

The distortion associated with the code is defined as

$$\mathrm{D} = \mathrm{Ed}(X^n, g_n(f_n(X^{\mathrm{n}}))) \tag{1.4}$$

Definition A rate distortion pair (R, D) is said to be achievable if there exists a sequence of $(2^{nR}, n)$-rate distortion code (f_n, g_n) with $\lim_{n\to\infty} \mathrm{Ed}(X^n, g_n(f_n(X^n)))$. [4]

In other words, the achievable rate is the rate we can use to code our source with the probability of error in decoding goes to zero when *n* goes to infinity. We can see there is similarities in the definition of the rate in the rate-distortion theory and in our pattern recognition. Thus, by instinct we can describe the achievable rate in pattern recognition as the rate that can describe our source data and stored data with the recognition result to be reliable. However, we need more accurate definition for the achievable rate, with a specific information statement of the pattern recognition.

1.2.2 Information Statement in Pattern Recognition

There are serval important definition and notations in our later discussion in the achievable rate region in pattern recognition. There are more to discuss in this part, most of the work are done by Professor Joseph A. O'Sullivan in his paper in 2008 [1] but we will only introduce the definition and conclusion in this and the following section. The rate pair we discuss in this thesis is the pair $(\mathrm{R_x}, \mathrm{R_y}, \mathrm{R_c})$, which refer to the memory data compression rate, the sensory data compression rate and the pattern rate, respectively. The system we use to discuss these definitions is Figure 1.2.

I. Pattern rate

The number of patterns that must be stored in the system is $\mathrm{M_c}$. Apparently, this number cannot exceed the capacity of the system it serves. Take human's eyes as an example, we can recognize 256 different colors and have about 2×10^{16} retinal photoreceptors. We denote the state of each photoreceptor as $\mathrm{X} = \{\mathrm{X_1}, \mathrm{X_1}, \dots, \mathrm{X_n}\}$, each X_1 is take from the alphabet $\boldsymbol{\mathcal{X}}$, *n* is the number of photoreceptors. Thus, our upper bound for $\mathrm{M_c}$ is $|\boldsymbol{\mathcal{X}}|^{\mathbf{n}} = 256^{2\times10^{16}}$. This is a fairly large number and clearly not the way we store our patterns. There are two reasons why we don't need to store this large number of patterns in our head, first, there's a strong structure in the pattern data, i.e., large percentage of the $256^{2\times10^{16}}$ are not actually patterns that we recognize or even have in nature. Second, as described before, our pattern recognition is originally used for survival in nature. Most of the pattern

we saw may be irrelevant and thus be ignored by our brain. The patterns that are related to our survival is the patterns we store in our brain. For these two reasons, the actual $\mathrm{M_c}$ for us, and for most pattern recognition systems, is $|\mathcal{X}|^{\mathrm{nR_c'}}$, where $\mathrm{R_c'}$ is the pattern rate, $R_c' \in (0,1)$. We can express R_c' in binary units, $R_c = R_c' \log_2 |\mathcal{X}|$.

II. Sensory data rate

Taking human as an example as we do in previous part, the sensory data rate refers to the rate we compress our data from retinas when we saw a pattern. As stated before, we are limited to display only the compressed data. In the sensory data compress part, our maximum is described as $\mathrm{M_y} = 2^{\mathrm{nR_y}} \ll |\mathcal{Y}|^{\mathrm{n}}$, where *Y* is the retinal data and the $\mathrm{R_y}$ is our sensory data rate.

In our diagram for the pattern recognition (Figure 1.2), the sensory data compression is the part from Y to J, where $\mathrm{J} = \varphi(\mathrm{Y})$.

The sensory data compression also plays as a constrain for the pattern rate. We cannot have the sensory data set smaller than what we stored, i.e., $\mathrm{M_y} \gg \mathrm{M_c}$, thus we have this constrain for pattern rate: $\mathrm{R_c} \ll \mathrm{R_y}$.

III. Memory data compression rate

As described in the previous sections, the memory data stored must be a compressed summarize of the raw data. From Figure 1.2, the $\mathrm{T(w)} = (\mathrm{X(w)}, \mathrm{w}), \mathrm{w} = \{1,2, \dots, \mathrm{M_c}\} = \mathcal{M}_\mathrm{c}$, is considered as a "template" for each pattern and $\mathrm{X(w)} = (\mathrm{X_1(w)}, \dots, \mathrm{X_n(w)})$.

The information memorized is considered as a class, $\mathrm{M(w)}$, is the output of the encoder *f*. In other words, $\mathrm{M}(w) = \mathrm{f}(\mathrm{T}(w))$, w is the memorized class label. The compressed rate of the memory data can then be described by the number of the index, M_x, as $R_x = \frac{1}{n} \log M_x$.

The memory data compression rate also plays a role in the constrain for the pattern rate. The number of pattern stored cannot exceed the number of the index, i.e., $M_x \gg M_c$ and thus, $R_c \ll R_x$.

1.2.3 Achievable Rate in Pattern Recognition

With the introduction of the three rates described the pattern recognition system, wo can now consider the achievable rate region. First, we will introduce the notations used in the section. The definitions we use in this section is cited from [1].

Definition Environment for the pattern recognition system is defined as:

$$\mathcal{E} = \{\mathcal{M}_c, \mathcal{X}, \mathcal{Y}, p(x), p(y|x), p(w), \mathcal{C}_x, \Phi\} \tag{1.5}$$

This environment is used for later discussion on our achievable rate region.

Definition Pattern recognition code (M_c, M_x, M_y, n) is contains:

$$\begin{gathered}\mathcal{M}_c = \{1, \dots, M_c\}, \mathcal{M}_x = \{1, \dots, M_x\}, \mathcal{M}_y = \{1, \dots, M_y\} \\ f: \mathcal{X}^n \times \mathcal{M}_c \to \mathcal{M}_x \times \mathcal{M}_c, f\big(t(w)\big) = f(x, w) = (i, w) \triangleq m(w) \\ \varphi: \mathcal{Y}^n \to \mathcal{M}_y, \varphi(y) = j \\ \mathrm{g}: \mathcal{M}_y \times (\mathcal{M}_x)^{M_c} \to \mathcal{M}_c, g(j, \mathcal{C}_u) = \widehat{w}\end{gathered} \tag{1.6}$$

Where $\mathcal{C}_{\mathrm{u}}$ is the output of applying f to $\mathcal{C}_{\mathrm{x}}$.

Definition The probability of an error for a code $(\mathrm{f}, \varphi, \mathrm{g})$ in $\mathcal{E}$ is defined as:

$$\mathrm{P}_{\mathrm{e}}^{\mathrm{n}}(\mathrm{w}) = \Pr(\widehat{\mathrm{W}} \neq \mathrm{w} | \mathrm{W} = \mathrm{w}) \tag{1.7}$$

Definition A rate $\mathrm{R} = (\mathrm{R_x}, \mathrm{R_y}, \mathrm{R_c})$ is said to be achievable if for any $\varepsilon > 0$ and for n sufficiently large, we have a $(\mathrm{M_c}, \mathrm{M_x}, \mathrm{M_y}, \mathrm{n})$ code $(\mathrm{f}, \mathrm{g}, \varphi)$ with rates

$$R_c' = \frac{1}{n}\log M_c$$

$$R_x' = \frac{1}{n}\log M_x \tag{1.8}$$

$$R_y' = \frac{1}{n}\log M_y$$

Such that $R_c' > R_c$, $R_x' < R_x$, $R_y' < R_y$, and $P_e^n < \varepsilon$.

Then, we can formulate the formula for the achievable rate pair (R_x, R_y, R_c). In Professor O'Sullivan's paper [1], he used two auxiliary random variables, U and V to define the achievable rate region.

First, consider the set of rates with the two auxiliary random variables:

$$\begin{aligned}\mathcal{R}_{UV} = \{R: R_x &\geq I(U;X)\\ R_y &\geq I(V;Y)\\ R_c &\leq R_x + R_y - I(XY;UV)\}\end{aligned} \tag{1.9}$$

And the two sets of random variables pairs:

$$\begin{aligned}\mathcal{P}_{in} = \{UV: &U - X - Y,\\ &X - Y - \mathrm{V},\\ &U - (X,Y) - V\}\end{aligned} \tag{1.10}$$

$$\begin{aligned}\mathcal{P}_{\mathrm{out}} = \{\mathrm{UV}: &\mathrm{U} - \mathrm{X} - \mathrm{Y},\\ &\mathrm{X} - \mathrm{Y} - \mathrm{V}\}\end{aligned} \tag{1.11}$$

Next, we consider the two set:

$$\begin{aligned}\mathcal{R}_{\text{in}} &= \{\text{R: R} \in \mathcal{R}_{\text{UV}} \text{ for some UV} \in \mathcal{P}_{\text{in}}\} \\ \mathcal{R}_{\text{out}} &= \{\text{R: R} \in \mathcal{R}_{\text{UV}} \text{ for some UV} \in \mathcal{P}_{\text{out}}\}\end{aligned} \tag{1.12}$$

Denote the convex hull of $\mathcal{R}_{in}$ as $\overline{\mathcal{R}_{in}}$.

Then, we introduce the main result in Professor O'Sullivan's paper [1] about the boundary of the achievable rate region:

Theorem 1 (inner bound)

$$\mathcal{R}_{in} \subseteq \mathcal{R} \tag{1.13}$$

In other words, every rate $\text{R} = (\text{R}_\text{x}, \text{R}_\text{y}, \text{R}_\text{c}) \in \mathcal{R}_{\text{in}}$ is achievable.

Theorem 2 (better inner bound)

$$\overline{\mathcal{R}_{\text{in}}} \subseteq \mathcal{R} \tag{1.14}$$

In other words, every rate $\text{R} = (\text{R}_\text{x}, \text{R}_\text{y}, \text{R}_\text{c}) \in \overline{\mathcal{R}_{\text{in}}}$ is achievable.

Theorem 3 (outer bound)

$$\mathcal{R}_{\text{out}} \supseteq \mathcal{R} \tag{1.15}$$

In other words, every rate $\text{R} = (R_x, R_y, R_c) \notin \mathcal{R}_{out}$ is not achievable.

There is a theorem on the bound of cardinality.

There are multiple ways to express the inner and outer bound by shuffling the mutual information and mathematical calculating. Here we will introduce one of the transforms which will be used in our later computation and deriving the algorithm.

$$\begin{aligned} R'_{UV} = \{R: R_x \geq I(U;X) \\ R_y \geq I(V;Y) \\ R_c \leq I(U;V) - I(U;V|XY)\} \end{aligned} \tag{1.16}$$

$$\begin{aligned} \mathcal{R}'_{in} = \{R: R \in R'_{UV} \text{ for some } UV \in \mathcal{P}_{in}\} \\ \mathcal{R}'_{out} = \{R: R \in R'_{UV} \text{ for some } UV \in \mathcal{P}_{out}\} \end{aligned} \tag{1.17}$$

The $\mathcal{R}'_{in}$ and $\mathcal{R}'_{out}$ here is equivalent to the $\mathcal{R}_{in}$ and $\mathcal{R}_{out}$ in (1.10) and (1.12).

1.3 Binary Example

In this section we introduce the binary example for achievable rate region. This example was given in Professor O'Sullivan's paper [1].

In the binary case we consider the following situation:

$$\begin{aligned} \mathcal{X} = \{0,1\} \\ \mathrm{P}(\mathrm{x}=0) = \mathrm{P}(\mathrm{x}=1) = 0.5 \\ \mathrm{Y} = \mathrm{X} \oplus \mathrm{W}, \mathrm{W} \sim \mathrm{Bonuli}(\mathrm{p}), \mathrm{p} = 0.2 \end{aligned} \tag{1.18}$$

There are two approaches for getting the achievable rate region [1]. Both are not computational results so that they cannot be extended to other cases, but the binary case is a good case to study and test our

algorithm in later sections. In fact, through this thesis we will use this binary case to test all of our algorithms.

Approach 1 Numerical Result.

The method involved the Monte Carlo method. We take a vast number of random probability distributions $p(uv|xy)$, compute the associated $I(\mathrm{U};\mathrm{X}), \mathrm{I}(\mathrm{V};\mathrm{Y})$ and $\mathrm{I}(\mathrm{UV};\mathrm{XY})$. Then, for each pair $(\mathrm{r_x}, \mathrm{r_y})$, we find the maximum $r_x + r_y - I(UV;XY)$. The maximum value is considered as our r_c. The result is the estimate surface of the achievable rate region.

In O'Sullivan's paper [1], the plot for the inner bound and outer bound is shown in Figure 1.3.

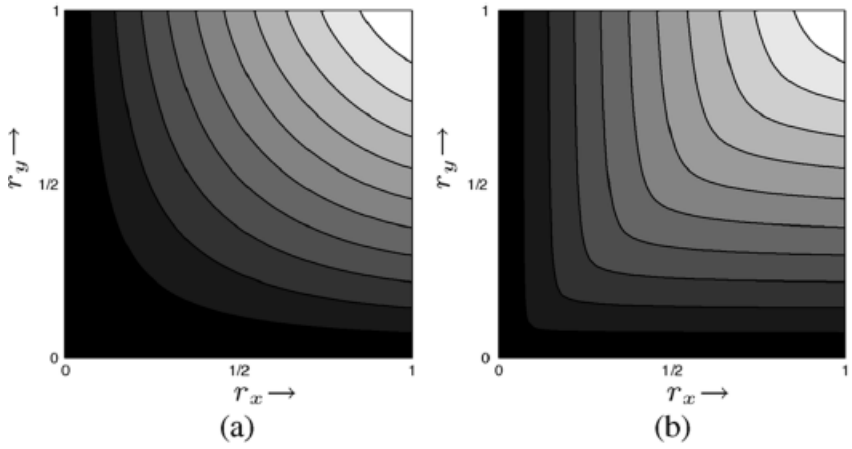

Figure 1.3 Inner and outer bound plot by Numerical Method [1]

Approach 2 Formulas approach.

This approach starts with a guess for the achievable rate region's surface and then prove the correctness. For simplicity, we only introduce the result for this approach.

The surface for the inner and outer bound is given by

$$r_{in}(r_x, r_y) = s(r_x, r_y) \tag{1.19}$$

$$r_{out}(r_x, r_y) = s^*(r_x, r_y)$$

Where

$$\begin{gathered} s(r_x, r_y) = 1 - h(q_x * q * q_y) \\ s^*(r_x, r_y) = \sup\theta(s(r_{x1}, r_{y1}) + (1-\theta)(s(r_{x2}, r_{y2}) \\ q_x = h^{-1}(1 - r_x) \\ q_y = h^{-1}(1 - r_y) \\ \mathrm{h(x)} = -\mathrm{x}\log(x) - (1-x)\log(1-x) \\ \mathrm{x} * \mathrm{y} = \mathrm{x}(1-\mathrm{y}) + \mathrm{y}(1-\mathrm{x}) \end{gathered} \tag{1.20}$$

These two approaches can only be applied on certain situations. Theoretically, the first approach can be applied on other situations, but it would take infinite time to run if the alphabet is large enough. The second one does not suit for other cases. Thus, these approaches are not what we desire for the computation of achievable rate region. We would want an algorithm with more applicable situations.

1.3 Notations

In this thesis, we use $\mathrm{I(X;Y)}$ to denote the mutual information between X and Y, where X and Y denote the variables. We use the "U-X-Y" to denote a Markov chain formed by *U, X* and *Y*, where $p(u, x, y) = p(x)p(u|x)p(y|x)$.

1.4 Working in this thesis

The computation of the boundary of achievable rate region takes an important part in many areas. From previous sections, it is clear that there's a lack of a method to compute the boundary of this achievable rate region using the formula from section 1.2.3. The method we use in section 1.3 is neither effective nor universal. There's a need for an algorithm to compute the boundary which can be extended to use in both pattern recognition and other area such as multi-user communication system.

In later sections, we will discuss the way we try to derive a new method from Blahut-Arimoto's algorithm for channel capacity and rate-distortion function, as well as other algorithms for optimization problems in information theory. We will use the binary case shown in section 1.3 as an example for our algorithm and test its accuracy.

Chapter 2

Blahut-Arimoto Method

The Blahut and Arimoto's algorithm for computing channel capacity and rate-distortion function are both introduced in 1972. Both algorithms can be described as alternating minimization algorithms, which will be discussed in the chapter.

The Blahut-Arimoto algorithm provides basic algorithm for doing alternating minimization, which will be used in later optimization algorithm.

This chapter will briefly introduce the Blahut-Arimoto algorithm and its MATLAB implementation.

2.1 Alternating Minimization Algorithm

Alternating minimization algorithm, introduced by Csiszar and G. Tusnady in 1984, deals with the optimization problem with multiple variables and is used vastly in many areas [6]. When solving an optimization problem with two variables, minimize with both variables simultaneously is usually not easy. Thus, there's need for an alternate algorithm. The basic idea of alternating minimization algorithm is to take turn minimizing over one variable with another variable fixed.

The minimization problem can be described as this:

$$\min_{(\mathrm{P},\mathrm{Q})\in\mathcal{P}\times\mathcal{Q}} \mathrm{D}(\mathrm{P},\mathrm{Q}) \tag{2.1}$$

Then for any arbitrary initial point $\mathrm{Q}_0 \in \mathcal{Q}$, for $\mathrm{n} \geq 1$, iteratively compute

$$\begin{aligned} P_n &\in \arg\min_{P\in\mathcal{P}} D(P, Q_{n-1}) \\ Q_n &\in \arg\min_{Q\in\mathcal{Q}} D(P_n, Q) \end{aligned} \tag{2.2}$$

This algorithm solves a sequence of minimization problems instead of the original one. If this algorithm converges, the converged value is the optimal solution of the original problem. [6]
The alternating maximization algorithm works as the same. This algorithm is important in the computation of channel capacity and rate-distortion function since there are more than one variable in the optimization problems in these two areas.

2.2 Blahut-Arimoto Algorithm of Channel Capacity

Blahut's algorithm for channel capacity involves the minimization of I-divergence in the previous section. This section will talk about the algorithm and its MATLAB implementation.
The Blahut's algorithm for the maximization problem plays a key role in later algorithm and the MATLAB code for this part can be used in later implementation. This algorithm provides a way for the alternating maximization algorithm

2.2.1 Optimization problem and Algorithm

In this section, we consider the same probability model as shown in previous section.
Capacity of a channel can be expressed as

$$C = \max_{x} I(X;Y)$$

$$= \max_{p \in \pi^n} I(p, Q)$$

$$= \max_{p \in \pi^n} \max_{P \in \mathcal{P}^{\triangle}} \sum_j \sum_k p_j Q_{k|j} \log \frac{P_{j|k}}{p_j} \tag{2.3}$$

$$= \max_{p \in \pi^n} \max_{P \in \mathcal{P}^{\triangle}} \left[\sum_j \sum_k p_j Q_{k|j} \log \frac{P_{j|k}}{p_j} - \sum_j \sum_k q_k P_{j|k} \right]$$

This optimization problem leads to the Blahut algorithm for channel capacity.

Blahut-Arimoto algorithm:

- Choose initial guess for $\mathbf{p}^0 \in \pi^n$, $p_j^0 > 0$ for all j, set n=0;
- Update P

$$P_{j|k}^n = \frac{p_j^n Q_{k|j}}{\sum_j p_j^n Q_{k|j}} \tag{2.4}$$

- Update p

$$p_j^{n+1} = \frac{\exp\left(\sum_k Q_{k|j} \log P_{j|k}^n\right)}{\sum_j \exp\left(\sum_k Q_{k|j} \log P_{j|k}^n\right)} \tag{2.5}$$

- If converges, stop; otherwise, set n=n+1, repeat. [3]

2.2.2 Implementation

The main part of the MATLAB code for the implementation of this algorithm is shown below. The function takes in q as its input and computes the P and p by the algorithm above. Then compute the channel capacity by computing the mutual information between X and Y.

```
for iter = 1:MaxIter
    for j = 1:n
        P(:,j) = p(j)*Q(:,j);
        P(:,j) = P(:,j)/sum(P(:,j));
    end

    for j = 1:n
        p1(j) = exp(Q(:,j)'*(log_2(P(:,j))));
    end
    p1 = p1/sum(p1);


    if norm(p1 - p) < error_tolerance
        break
    else
        p = p1;
    end
end
```

2.2.3 Result for Binary Example

Take the basic binary channel for an example, the channel is shown in Figure 2.2. Suppose X is the input and Y is the output of this channel.

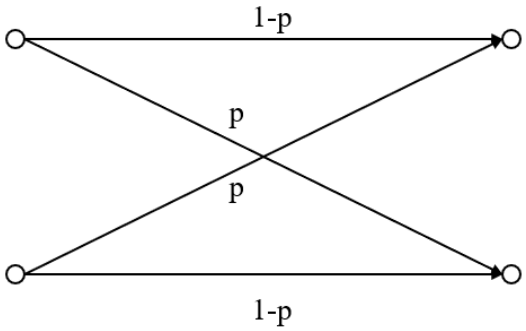

Figure 2.1 Binary Channel Case

In the binary case, the cross over probability is p, then the conditional probability is:

$$\mathrm{P}(\mathrm{Y}=0|\mathrm{X}=0)=\mathrm{P}(\mathrm{Y}=1|\mathrm{X}=1)=1-p$$

$$\mathrm{P}(\mathrm{Y}=1|\mathrm{X}=0)=\mathrm{P}(\mathrm{Y}=0|\mathrm{X}=1)=p$$

The formula for the binary channel capacity is:

$$\mathrm{C}=1-\mathrm{h}(p) \tag{2.6}$$

where $\mathrm{h}(p)=-p\log p-(1-p)\log(1-p)$. [4]

The backward matrix for this channel $\mathrm{Q_{kjj}}$ is $\begin{pmatrix}1-\mathrm{p} & \mathrm{p}\\ \mathrm{p} & 1-\mathrm{p}\end{pmatrix}$, as shown in section 2.2. Use this backward matrix for the algorithm's input and for varying p, we can have the channel capacity plot for the binary channel. The left plot in Figure 2.3 shows the result of the Blahut's algorithm for channel capacity. The right plot in Figure 2.3 shows the computational result of binary channel's capacity using formula (2.6).

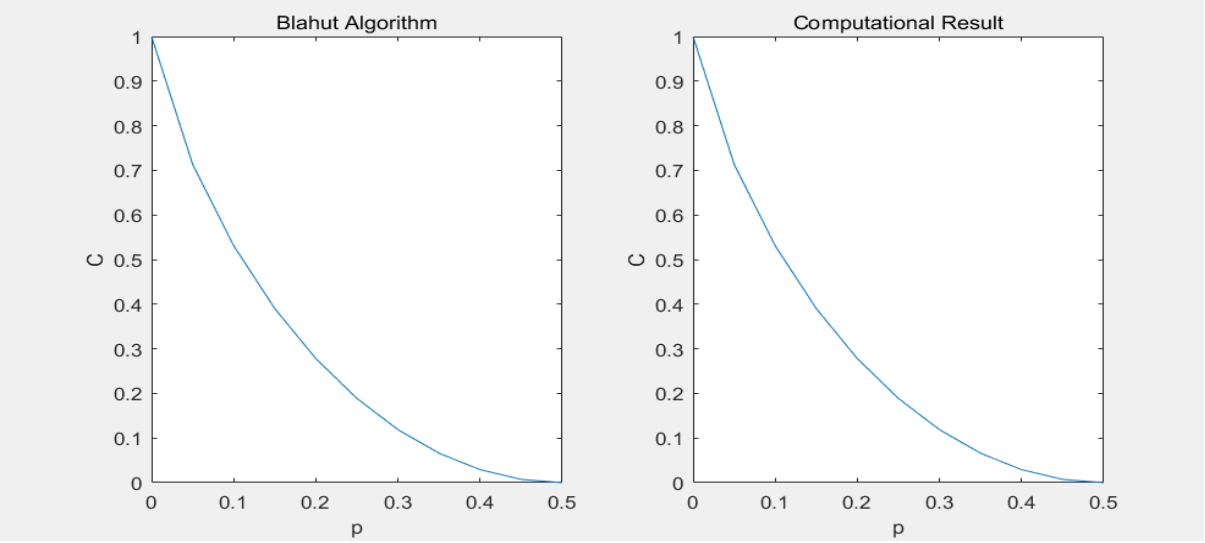

Figure 2.2 Channel Capacity Result for Binary Channel

The above Figure is the result comparison between the algorithm's result and analytical result. In these two plots, the upper left point (0,1) is the point where the crossover probability is 0, indicating that in this case, the channel can transfer every input without any error, since the output is the same with the input. At the down right corner, the channel's capacity is 0 because the crossover probability is 0.5, indicating that for any input, the output is random. In this case, the channel cannot transfer any information.

The two curves above is exactly the same, indicating that the algorithm implementation is correct.

2.3 Rate-Distortion Algorithm

The Rate-Distortion algorithm was presented in Blahut's 1972 paper [2]. This algorithm is an alternating optimization algorithm for minimizing problems. This section will talk about the algorithm and its MATLAB implementation.

The Blahut's Rate-Distortion Algorithm provides a way for the alternating minimization algorithm. Together with the Blahut's algorithm for channel capacity, which provides a way for the alternating maximization algorithm, we can derive an algorithm for the achievable rate region. The MATLAB code for this algorithm also severs well in the later implementation.

2.3.1 Optimization problem and Algorithm

For this part, we assume that we have the probability distribution model. This algorithm was presented in Blahut's 1972 paper [2][3]. Let $\rho_{jk} \geq 0$ be the distortion measure (similar to the distortion function described in section 1.2.1). The rate-distortion function is described as

$$\begin{aligned} \mathrm{R(D)} &= \min_{Q \in Q_D} I(p, Q) \\ &= \min_{Q \in Q_D} \min_{q \in \pi^m} \sum_j \sum_k p_j Q_{k|j} \log \frac{Q_{k|j}}{q_k} \end{aligned} \tag{2.7}$$

Where Q_D refers to the set of probabilities that satisfies the distortion constrain. Q_D is defined as

$$\mathrm{Q_D} = \{\mathrm{Q} \in \mathcal{Q}^{\Delta} : \sum_{\mathrm{j}} \sum_{\mathrm{k}} \mathrm{p_j Q_{k|j}}\, \rho_{\mathrm{jk}} \leq \mathrm{D}\} \tag{2.8}$$

For deriving the algorithm of the minimization problem (2.7), we need to introduce the Lagrange multiplier *s* since this is a constrained optimization problem. For each Lagrange multiplier *s*, there is a corresponding distortion constrain $\mathrm{D_s}$. Then for each specific multiplier *s*, the corresponding rate-distortion function is

$$\begin{aligned} \mathrm{R(D_s)} &= \min_{\mathrm{Q \in Q_D}} \sum_{\mathrm{j}} \sum_{\mathrm{k}} \mathrm{p_j Q_{k|j}} \log \frac{\mathrm{Q_{k|j}}}{\sum_{\mathrm{j'}} \mathrm{p_{j'} Q_{k|j'}}} \\ &\quad -\mathrm{s}\left(\sum_{\mathrm{j}} \sum_{\mathrm{k}} \mathrm{p_j Q_{k|j}}\, \rho_{\mathrm{jk}} - \mathrm{D_s}\right) \\ &= \mathrm{sD_s} + \min_{\mathrm{Q \in Q_D}} \min_{\mathrm{q \in \pi^m}} \sum_{\mathrm{j}} \sum_{\mathrm{k}} \mathrm{p_j Q_{k|j}} \log \frac{\mathrm{Q_{k|j}}}{\mathrm{q_k e^{s\rho_{jk}}}} \end{aligned} \tag{2.9}$$

Blahut's algorithm for rate-distortion function

- Initial guess for $\mathbf{q}$: $q^0 \in \pi^m, q^k > 0$ for all k, set $n = 0$
- Update $\mathbf{Q}$:

$$Q_{k|j}^n = \frac{q_k^n e^{s\rho_{jk}}}{\sum_{k'} q_{k'}^n e^{s\rho_{jk'}}} \tag{2.10}$$

- Update $\mathbf{q}$:

$$q_k^{n+1} = \sum_j Q_{k|j}^n p_j \tag{2.11}$$

- If q converges, stop; else, $\mathbf{n = n + 1}$, iterate.

2.3.2 Implementation

The MATLAB code for the Blahut's algorithm for the rate-distortion function is shown below. After solving for the optimal Q and q, we can compute the distortion by $\sum_j \sum_k p_j Q_{k|j} \rho_{jk}$ and the rate by (2.9).

```
for iter=1:10000
    for j = 1:m
        Q(:,j) =  q.* exp(s*d(j,:));
        Q(:,j) = Q(:,j)/sum(Q(:,j));
    end
    q1=sum(Q.*p);
    %check convergence
    if norm(q1-q)<error
        break;
    else
        q = q1;
    end
end
```

2.3.3 Result for Binary Case

For testing the accuracy of the MATLAB code, again we used the binary case in information theory [4] to test the solution given by the code from 2.4.2. First, give the definition of rate-distortion function.
Theorem The rate distortion function for an i.i.d. source X with distribution $p(x)$ and the bounded distortion function $\mathrm{d}(\mathrm{x}, \hat{x})$ is given by

$$\mathrm{R(D)} = \min_{p(\hat{x}|x):\sum_{(x,\hat{x})} p(x)p(\hat{x}|x)d(x,\hat{x})\leq D} I(X;\hat{X}) \tag{2.12}$$

Then we can introduce the binary case ant its rate distortion function.
Theorem The rate distortion function for a Bernoulli(p) source with Hamming distortion (1.1) is given by

$$\mathrm{R(D)} = \begin{cases} H(p) - H(D), & 0 \leq D \leq \min\{p, 1-p\} \\ 0, & D > \min\{\mathrm{p}, 1-\mathrm{p}\} \end{cases} \tag{2.13}$$

The channel's scratch map is shown in Figure 2.4.

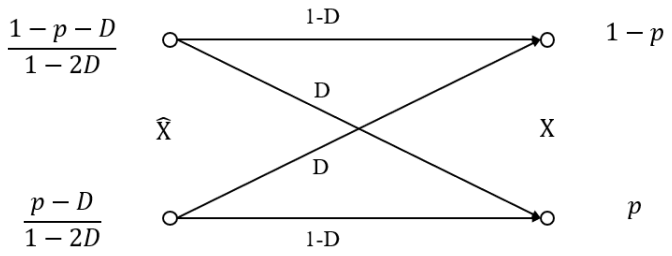

Figure 2.3 Binary Channel for Rate-Distortion

Take $p = 0.5$, Hamming distortion function as our distortion function, we compare the result of the Blahut's algorithm implementation (section 2.4.3) and the formula (2.13). The result is shown in Figure 2.5. The right-hand plot is the outcome of the formula 2.13. It's worth noticing that the Lagrange multiplier *s* in this algorithm should take negative values since there is a minus before the constrain part in (2.9).

From Figure 2.5, we can see that the outcome of the Blahut's Algorithm and the formula in 2.13 is the same.

In Figure 2.5, these two curves have similar shape with Figure 2.4, but the meaning of these curves are different. In Figure 2.5, the upper left point (0,1) indicates the case where the estimation is the same with the input. In this case, we only need one bit to represent the input to get errorless result. On the down right corner, when the crossover probability is 0.5, we can never represent the input with the probability of error arbitrarily small.

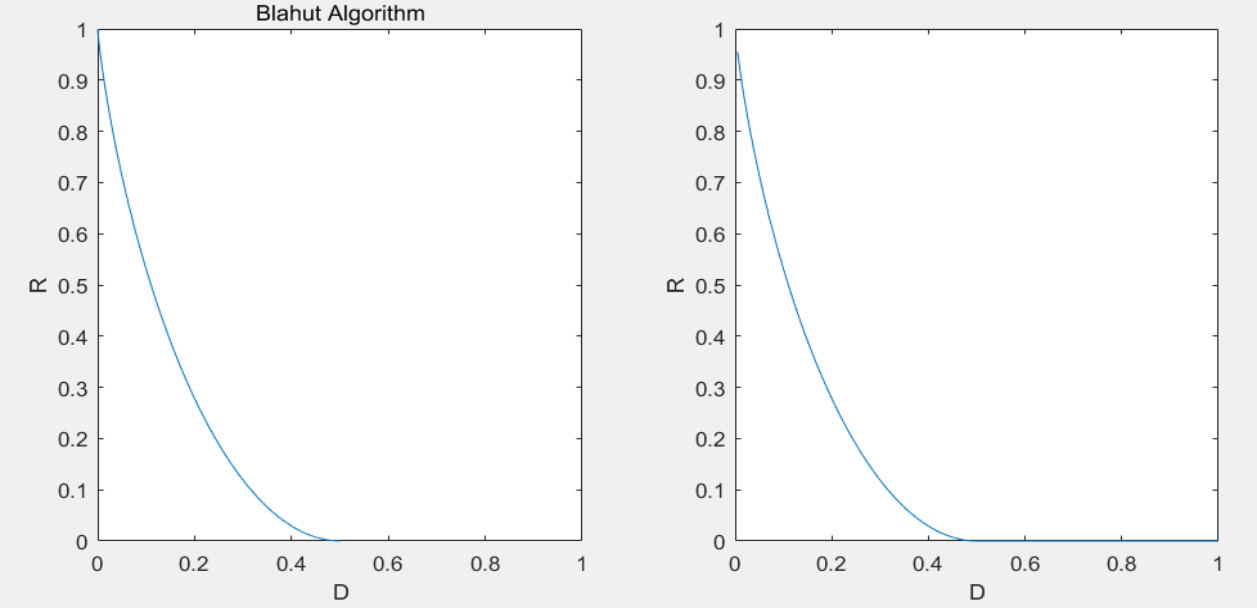

Figure 2.4 MATLAB result comparison

With these two algorithms in hand, we can now derive our first attempt algorithm for our pattern recognition achievable rate.

2.4 Conclusion

In this section we introduced two alternating minimization algorithms in information theory. These two algorithms are proven to be correct and have been proven to converge. The reason we are introducing these two algorithms is that they are useful algorithms in computing maximization and minimization problems in information theory. In our thesis we are dealing with probability distributions and they have more constrains than normal random variables. As a result, we will need

these algorithms as examples for deriving our algorithm for computing the achievable rate region in later sections.

Chapter 3

Algorithm for the Main Problem

With the Blahut-Arimoto algorithms for alternation minimization and maximization problem, we can now focus on our target optimization problem. The target optimization problem includes maximizations and minimization problems with multiple variables. By combing the two algorithms above, we can derive the algorithms for the target optimization problem and implement it on MATLAB to come to a result. The derivation of this algorithm includes the method mentioned in Chapter 2.

The optimization problem discussed in this section is for the computation of the inner bound of the achievable rate region.

3.1 Optimization Problem Description

As shown in Chapter 1, the target optimization problem is to find the pair (R_x, R_y, R_c) by the following formulas (here we chose the first transform for simplicity)

$$\begin{aligned} R_x &\geq I(U;X) \\ R_y &\geq I(V;Y) \\ R_c &\leq I(U;V) - I(U;V|X,Y) \end{aligned} \tag{3.1}$$

Which can be described as two maximization problems and one minimization problem. The optimization problems for R_X and R_Y are quite straightforward, but we need to modify the formula for R_C's problems to make it easier to derive our algorithm.

3.1.1 Simplify the optimization problem

For simplicity, we modify the formula for R_C.

$$\begin{aligned} R_c &\le I(U;V) - I(U;V|X,Y) \\ &= \mathrm{I(U;VX,Y;U)} - \mathrm{I(X,Y;U|V)} \\ &= \mathrm{H(U)} - \mathrm{H(U|X,Y)} - \mathrm{I(X,Y;U,V)} \\ &= \mathrm{H(U)} - \mathrm{H(U|X)} - \mathrm{I(X,Y;U|V)} \\ &= \mathrm{I(U;X)} + \mathrm{I(V;Y)} - \mathrm{I(V;Y)} - \mathrm{I(X,Y;U|V)} \\ &= \mathrm{I(U;X)} + \mathrm{I(V;Y)} - \mathrm{I(V;Y)} - \mathrm{I(X,Y;U|V)} \\ &= \mathrm{I(U;X)} + \mathrm{I(V;Y)} - \mathrm{I(V;X,Y)} - \mathrm{I(XY;U|V)} \\ &= \mathrm{I(U;X)} + \mathrm{I(V;Y)} - (\mathrm{XY;UV}) \end{aligned} \tag{3.2}$$

Thus, the problem can be simplified as:

$$\min -I(XY;UV) \tag{3.3}$$

Which is equivalent to:

$$\min_{q(u,v)} \sum_u \sum_v \sum_x \sum_y p(x,y)p(u,v|x,y)log\frac{P(x|u)}{p(x)} \tag{3.4}$$

This minimization problem can lead to the optimal $q(u,v)$ with constrains on R_x and R_y.

3.1.2 Optimization problem statement

After simplifying the formula for R_C, we have a minimization problem for finding the optimal $q(u,v)$ with some constrains on R_x and R_y. The constrains are on the mutual information between U and X, and V and Y. From Blahut-Arimoto's algorithm for the rate-distortion, we can write the constrains into the (3.4) using Lagrange multipliers for the constrains. The optimization problem can be write as:

$$\begin{aligned} &\min_{q(u,v)} \sum_u \sum_v \sum_x \sum_y p(x,y)p(u,v|x,y) log \frac{P(x|u)}{p(x)} \\ &-\lambda_x \left[\max_{p(x|u)} \sum_x \sum_u p(x)p(u|x)\, log \frac{P(x|u)}{p(x)} \right] \\ &-\lambda_y \left[\max_{p(y|v)} \sum_y \sum_v p(y)p(v|y)\, log \frac{P(y|v)}{p(y)} \right] \end{aligned} \tag{3.5}$$

3.2 Algorithm by Combing Blahut's Algorithm

In this section, we derive an algorithm solely from (3.5). The algorithm came mostly from the Blahut-Arimoto's algorithms and will be proven to be not suitable for our problem. An alternate way of finding the algorithm for our problem will be discussed.

3.2.1 Use Similar Algorithm as Blahut-Arimoto's

For deriving the algorithm for (3.1.5), we can use part of the Blahut-Arimoto's algorithm. Notice that the maximum part is similar to the channel capacity part in Blahut's algorithm and the minimum part is similar to the rate-distortion part. The difference here is that in our optimization problem, although we are also calculating for the achievable rate, we do not have a distortion function for this problem. However, we can still use the fundamental idea of Blahut's rate-distortion algorithm.

Algorithm for computing the achievable rate:

- Initial guess for $p^{(0)}(u|x), p^{(0)}(v|y)$, set k=0
- Update $q(u,v)$

$$\mathrm{q^{(k)}(u,v)} = \sum_{\mathrm{x}} \sum_{\mathrm{y}} \mathrm{p(x,y)p^{(k)}(u|x)p^{(k)}(v|y)} \tag{3.6}$$

- Update $p(x|u), p(y|v)$

$$\begin{aligned} p^{(k)}(x|u) &= \frac{p^{(k)}(u|x)p(x)}{\sum_{x'} p^{(k)}(u|x')p(x')} \\ p^{(k)}(y|v) &= \frac{p^{(k)}(v|y)p(y)}{\sum_{y'} p^{(k)}(v|y')p(y')} \end{aligned} \tag{3.7}$$

- Update $p(u|x), p(v|y)$

$$\begin{aligned} p^{(k+1)}(u|x) &= \frac{q(u)e^{\lambda_x d_x^{(k)}(u,x)}}{\sum_{u'} q(u')e^{\lambda_x d_x^{(k)}(u',x)}} \\ p^{(k+1)}(v|y) &= \frac{q(v)e^{\lambda_y d_y^{(k)}(v,y)}}{\sum_{v'} q(v')e^{\lambda_y d_y^{(k)}(v',y)}} \end{aligned} \tag{3.8}$$

In this algorithm, we chose $d_x(u,x) = \log\frac{\mathrm{p^{(k)}(x|u)}}{\mathrm{p(x)}}, d_y(v,y) = \log\frac{\mathrm{p^{(k)}(y|v)}}{\mathrm{p(y)}}$.

- If $p(u|x), p(v|y)$ both converge, stop; else, set $\mathrm{k = k + 1}$, iterate.

In this algorithm, the $p^{(k+1)}(u|x)$ step will take the same formula as in (2.10), with $d_x(u,x)$ and $d_y(v,y)$ defined as in the algorithm. Here we chose to use $d_x(u,x)$ and $d_y(v,y)$ in our algorithm for simplicity and can help transform from the Blahut's algorithm to ours.

3.2.2 MATLAB implementation

```
P_ux_1=Q_u'.*(exp(Lx*Dx));
for i = 1:sx
    P_ux_1(:,i)=P_ux_1(:,i)/sum(P_ux_1(:,i));
end
    %-----------------------------------------
P_vy_2=Q_v'.*(exp(Ly*Dy));
for j = 1:sy
    P_vy_2(:,i)=P_vy_2(:,i)/sum(P_vy_2(:,i));
end
```

The MATLAB codes for this algorithm is shown above. This function takes the joint probability of X and Y as input and calculates the rates by (3.1) and (3.2).

For simplicity, we only show the (3.8) part of this algorithm, other step of this algorithm is similar to the Blahut's algorithm.

3.2.3 Result and conclusion

For testing the result of this algorithm, we chose the binary case as shown in Chapter 1, with $p(x) = [0.5,0.5]$, $Y = X \oplus W$, $W \sim Bernoulli(p)$, $p = 0.2$. The alphabet of U and V are both same with X and Y, respectively. i.e., $|\mathcal{U}| = |\mathcal{X}|, |\mathcal{V}| = |\mathcal{Y}|$. Chose Lagrange multipliers λ_x and λ_y varies from -5 to 0. Again, the Lagrange multipliers here should take negative values for there is a minus before the constrain in (3.5).

Unfortunately, this code doesn't return any feasible solution to our problem. The converge speed of this algorithm exceed acceptance and the probability distribution with shoot to infinity within serval steps of iteration. This result is clearly unwanted and thus there's need for an alternative algorithm for this computation. Since the problem of this algorithm is the fast convergence, the next step for our algorithm is to find a way to slow down the convergence.

3.3 Algorithm with Slow Convergence

In the previous section, we have shown that simply combing the Blahut's algorithm will bring us nowhere near our solution. Thus, there's need for a modified algorithm. The algorithm we will talk about in this section is derived from a new function with an additional part serving as the slow convergence part. In this section, we will briefly introduce our idea of how to slow down the convergence, then introduce the modified algorithm with its MATLAB implementation and result.

3.3.1 Slow convergence

There are multiple ways to do the slow convergence part [7]. In here, we chose to add a relative entropy into (3.5).

Definition The relative entropy between two probability mass functions $p(x)$ and $q(x)$ is defined as

$$D(p||q) = \sum_{x \in X} p(x) \log \frac{p(x)}{q(x)} \tag{3.9}$$

Relative entropy is zero if and only if $p = q$. In our algorithm, we will use this property to slow our algorithm.

3.3.2 Modified Algorithm

After adding the slow convergence part, the optimization problem becomes:

$$\min_{q(u,v)} \sum_u \sum_v \sum_x \sum_y p(x,y)p(u,v|x,y) log \frac{P(x|u)}{p(x)} \tag{3.10}$$

$$-\lambda_x [\max_{p(x|u)} \sum_x \sum_u p(x)p(u|x) \, log \frac{P(x|u)}{p(x)}]$$

$$-\lambda_y[\max_{p(y|v)} \sum_y \sum_v p(y)p(v|y)\, log \frac{P(y|v)}{p(y)}$$

$$+v_x \sum_{x,u} p(x)p(u|x) log \frac{p(ulx)}{\pi(u|x)}$$

$$+v_y \sum_{y,v} p(y)p(v|y) log \frac{p(vly)}{\pi(v|y)}$$

$$+some\ function$$

In (3.3.2), the $\sum_{x,u} p(x)p(u|x)\log\frac{p(ulx)}{\pi(u|x)}$ part is the relative entropy between $p(u|x)$ and $\pi(u|x)$. This part is for the slow convergence. As shown in 3.3.1, this part goes zero if and only if $p(u|x)$ and $\pi(u|x)$ are equal. For the algorithm, if we take $\pi(u|x) = p^{(k)}(ulx)$, then for each step in the algorithm, $p^{(k+1)}(u|x)$ will be as close as possible to $p^{(k)}(u|x)$, which will slow down our algorithm. The function added to the back is some function which make this a probability.

To derive the algorithm from (3.3.2), instead of modifying algorithm from Blahut-Arimoto's algorithm, we need to take the derivative with respect to $p(u|x)$ and $p(v|y)$ respectively and derive the formula that makes them optimal.

Take the derivative with respect to $p(u|x)$ gives us

$$\sum_y \sum_v \frac{p(x,y)}{p(x)} p(v|y) \log \frac{p(u|x)}{q^{(k)}(u|v)} - \lambda_x \frac{p(x)}{P(x)} \log p^{(k)}(x|u)$$
$$+v_x \frac{p(x)}{p(x)} \log \frac{p(u|x)}{p^{(k)}(u|x)} - \tilde{\mu}(x) = 0 \qquad (3.11)$$

Solve for $p(u|x)$ will give us

$$
\begin{aligned}
&\mathrm{p}^{(k+1)}(u|x) \\
&= \frac{e^{(\sum_v(\sum_y p(y|x)p(v|y))\log q^{(k)}(u|v))+\lambda_x \log p^{(k)}(x|u)+\nu_x \log p^{(k)}(u|x))}}{z^{(k+1)}(x)} \\
&= \frac{e^{((\nu_x+\lambda_x)\log p^{(k)}(u|x)-\lambda_x q^{(k)}(u)+\sum_v(\sum_y p(y|x)p(v|y))\log q^{(k)}(u|v)))}}{z^{(k+1)}(x)}
\end{aligned}
\tag{3.12}
$$

Where $\boldsymbol{z^{(k+1)}(x)}$ is some function that makes this a probability distribution. In our algorithm, we will take the summation over the nominator with respect to u.

Then we can formulate our algorithm with (3.12), the update of $p^{(k)}(x|u)$, $p^{(k)}(y|v)$ will be the same as before in (3.4).

Algorithm:

- Initial guess for $p^{(0)}(u|x)$, $p^{(0)}(v|y)$, set k=0
- Update $q(u,v)$

$$
q^{(k)}(u,v) = \sum_x \sum_y p(x,y)p^{(k)}(u|x)p^{(k)}(v|y) \tag{3.13}
$$

- Update $p(x|u), p(y|v)$

$$
\begin{aligned}
p^{(k)}(x|u) &= \frac{p^{(k)}(u|x)p(x)}{\sum_{x'} p^{(k)}(u|x')p(x')} \\
p^{(k)}(y|v) &= \frac{p^{(k)}(v|y)p(y)}{\sum_{y'} p^{(k)}(v|y')p(y')}
\end{aligned}
\tag{3.14}
$$

- Update $p(u|x), p(v|y)$

$$p^{(k+1)}(u|x) = \frac{e^{((\nu_x+\lambda_x)\log p^{(k)}(u|x)-\lambda_x q^{(k)}(u)+\sum_v(\sum_y p(y|x)p(v|y))\log q^{(k)}(u|v))}}{\sum_{u'} e^{((\nu_x+\lambda_x)\log p^{(k)}(u'|x)-\lambda_x q^{(k)}(u')+\sum_v(\sum_y p(y|x)p(v|y))\log q^{(k)}(u'|v))}}$$

$$p^{(k+1)}(v|y) = \frac{e^{((\nu_y+\lambda_y)\log p^{(k)}(v|y)-\lambda_y q^{(k)}(v)+\sum_u(\sum_x p(x|y)p(u|x))\log q^{(k)}(v|u))}}{\sum_{v'} e^{((\nu_y+\lambda_y)\log p^{(k)}(v'|y)-\lambda_y q^{(k)}(v')+\sum_u(\sum_x p(x|y)p(u|x))\log q^{(k)}(v'|u))}} \quad (3.15)$$

- If $p(u|x), p(v|y)$ both converge, stop; else, set $k = k + 1$, iterate.

We will then implement this algorithm and test the accuracy.

3.3.3 MATLAB implementation

The MATLAB codes for this algorithm is shown below. This function takes the joint probability of X and Y as input and calculates the rates by (3.1) and (3.2). Again, in this part we only show the $p(u|x)$ and $p(v|y)$ part for simplicity.

```
for i=1:sx
        for m=1:su
            P_ux_1(m,i) = exp((Vx + Lx) *
log_2(P_ux(m,i)) - Lx * log_2(Q_u(m)) +
sum(sum(P_yx(:,i).*(P_vy')) .* log_2(Q_uv(m,:))));
        end
        P_ux_1(:,i) = P_ux_1(:,i)/sum(P_ux_1(:,i));
    end
    %-----------------------------------------
    for j=1:sy
        for n=1:sv
            P_vy_1(n,j) = exp((Vy + Ly) *
log_2(P_vy(n,j)) - Ly * log_2(Q_v(n)) +
sum(sum(P_xy(:,j).*(P_ux_1')) .* log_2(Q_vu(n,:))));
        end
        P_vy_1(:,j) = P_vy_1(:,j)/sum(P_vy_1(:,j));
    end
```

3.3.4 Result

The result for the code shown in 3.3.3 is shown in Figure 3.1.

For testing, we still chose to use the binary case for simplicity and easier comparison. We chose $|\mathcal{X}| = |\mathcal{Y}| = \{0,1\}$, $p(x=0) = p(x=1) = 0.5$, $Y = X \oplus W$, where $W \sim Bernoulli(p)$, $p = 0.2$. We chose U and V as $|\boldsymbol{\mathcal{U}}| = |\boldsymbol{\mathcal{V}}| = |\boldsymbol{\mathcal{X}}| = |\boldsymbol{\mathcal{Y}}|$. The Lagrange multipliers are chosen from -3 to 0 and the multipliers for the slow convergence part are both 4. The multipliers for the slow convergence only serve for slowing down the algorithm and do not influence the result, and thus there's no need to vary these two values.

The choice of the initial guess for $p(u|x)$ and $p(y|v)$ are randomly generated by MATLAB and normalized to be probability distributions.

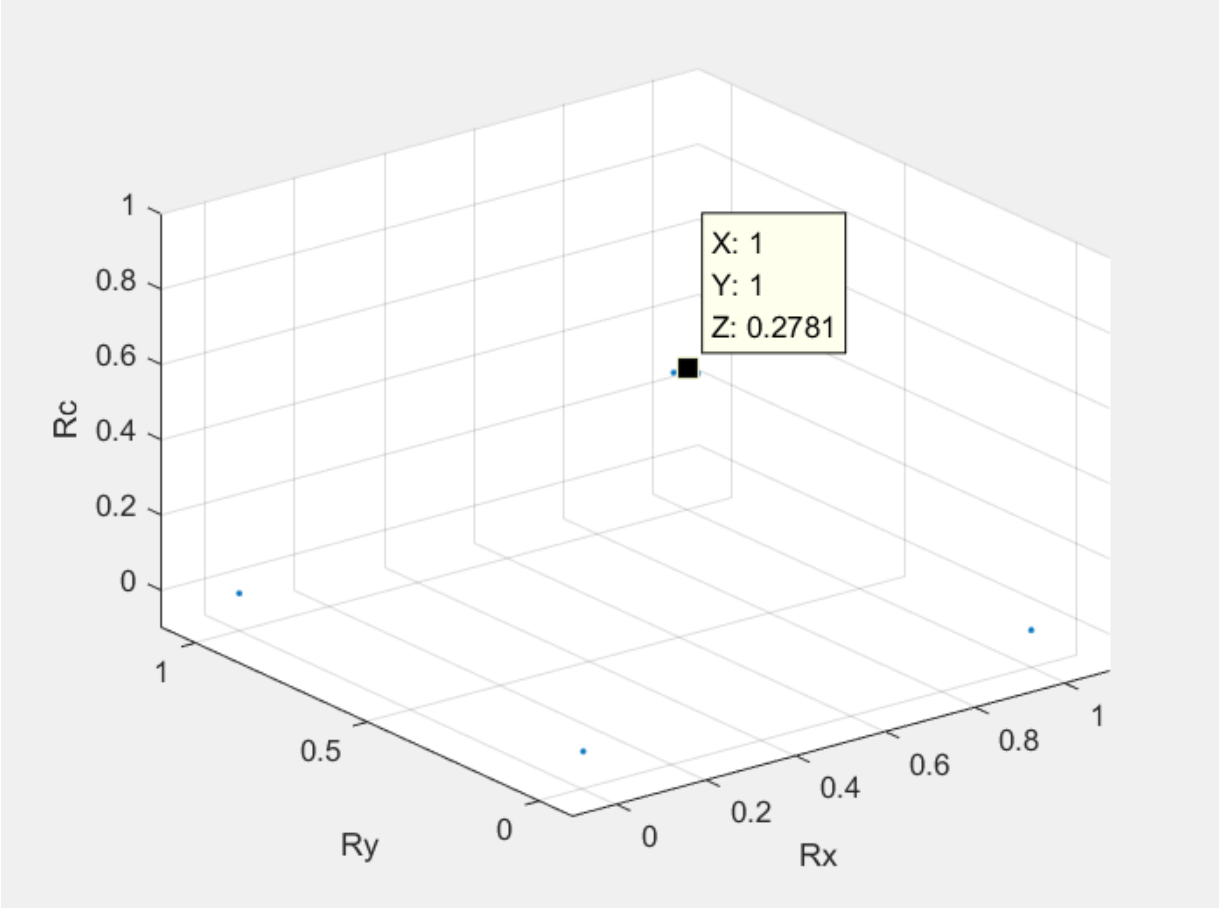

Figure 3.1 MATLAB result for algorithm in 3.3.3

We can see from the result that the solution this algorithm can find are merely some boundary points which do not help us find the achievable rate region. We can decide the accuracy of these four corner points by some analyze. In this part, we will use the point's coordinate value to represent each point. The pair (x, y, z) represent (R_x, R_y, R_c) respectively. We use (3.1) to analyze each point.

For point (0,0,0) This point represents the case where U is independent of X and V is independent of Y. In this case, the mutual information between U and X, V and Y are both 0. Thus, the mutual information between U and V is also zero.

For point (1,0,0) This point represents the case where U=X and V is independent of Y. Thus, by the definition of mutual information, $\mathrm{I(U;X)} = 1$. In other word, we only need one bit to represent X by U. However, this point doesn't serve well for our computation of the achievable rate region.

For point (0,1,0) This point is similar to the previous one except that in this case, V=Y and U is independent of X.

For point (1,1,0.2781) This point is the only point with nonzero R_c. To decide the accuracy, we need to figure out the meaning of the point and the analytical result for R_c in this case. According to the previous cases, this point represents the case where U=X and V=Y. Thus, we can have

$$\begin{aligned} \mathrm{I}(\mathrm{U};\mathrm{V}) &= \mathrm{H}(\mathrm{U}) - \mathrm{H}(\mathrm{U}|\mathrm{V}) \\ &= \mathrm{H}(\mathrm{X}) - \mathrm{H}(\mathrm{X}|\mathrm{Y}) \\ &= \mathrm{h}(x) - \mathrm{h}(p) \end{aligned} \tag{3.16}$$

Where $\mathrm{h}(p) = -p\log p - (1-p)\log(1-p)$.

By computation, the value of (3.16) is exactly 0.2781, which proves the accuracy of this point.

From the analyze above, the result of this algorithm is proven to be accurate. However, these points only represent limiting situations and we can't derive an accurate achievable rate region by these points. Aside from the boundary points, we also need to know the points between these four points.

3.4 Conclusion

In this section, we present two algorithm attempts for computing the achievable rate region in pattern recognition. The results of both algorithms are proven to be not suitable for solving this problem. The first algorithm converges too fast and will not generate any feasible solutions. The second algorithm is derived after adding a part for the slow convergence in the formula. However, the second algorithm can only have solutions one some limiting situations and generate four solutions. To compute the accurate achievable rate region we needed, we will need the points between these four points. The challenge of computing the achievable rate region with two variables at the same time lies in the first step of our algorithm, where we use the multiple of the two variables we are trying to optimize over. We believe that the multiple part is the reason why we can only generate the corner points.

In the next section, we will talk about a sub-problem of the original problem for the achievable rate region. By solving the sub-problem, we can have more understanding of this problem.

Chapter 4

Algorithm for a Sub-Problem

From chapter 3 we can see that the formulation of our algorithm doesn't go smoothly. The result of our algorithm only converges to the corner points of our region. Although we have proven that the result is correct, we still don't have an algorithm that computes the bound of the achievable rate region. We will need some other approaches for our computation.

In this chapter, we will discuss a sub-problem of our main problem discussed in the previous sections. Then, by formulating the algorithm for the sub-problem, we will then discuss the accuracy of this new algorithm and seek some method to apply the new algorithm to our main problem.

4.1 Optimization Problem Description

In this section, we will present the alternate optimization problem, which is a sub-problem of the main problem discussed in chapter 3. This sub-problem will be easier to solve.

4.1.1 Guesses for the achievable rate region

Even though the result of the previous algorithm is not what we excepted, the result of it is still worth studying.

From Figure 3.1, we have the four corner points for our boundary. Although these points can't lead us any closer to the achievable rate region, it still tells us something about this region.

One of those things is that, we can guess what the shape of this region by these vertices.

From Figure 3.1, we have the vertices of this region. Since the bound of this region, presented in (3.1) in the previous chapter, is given by the mutual information.

Plot of mutual information For some variables X and Y, if we varies the probability distribution of X and fixed the conditional probability between them, we have the plot of the mutual information between X and Y shown in Figure 4.1.

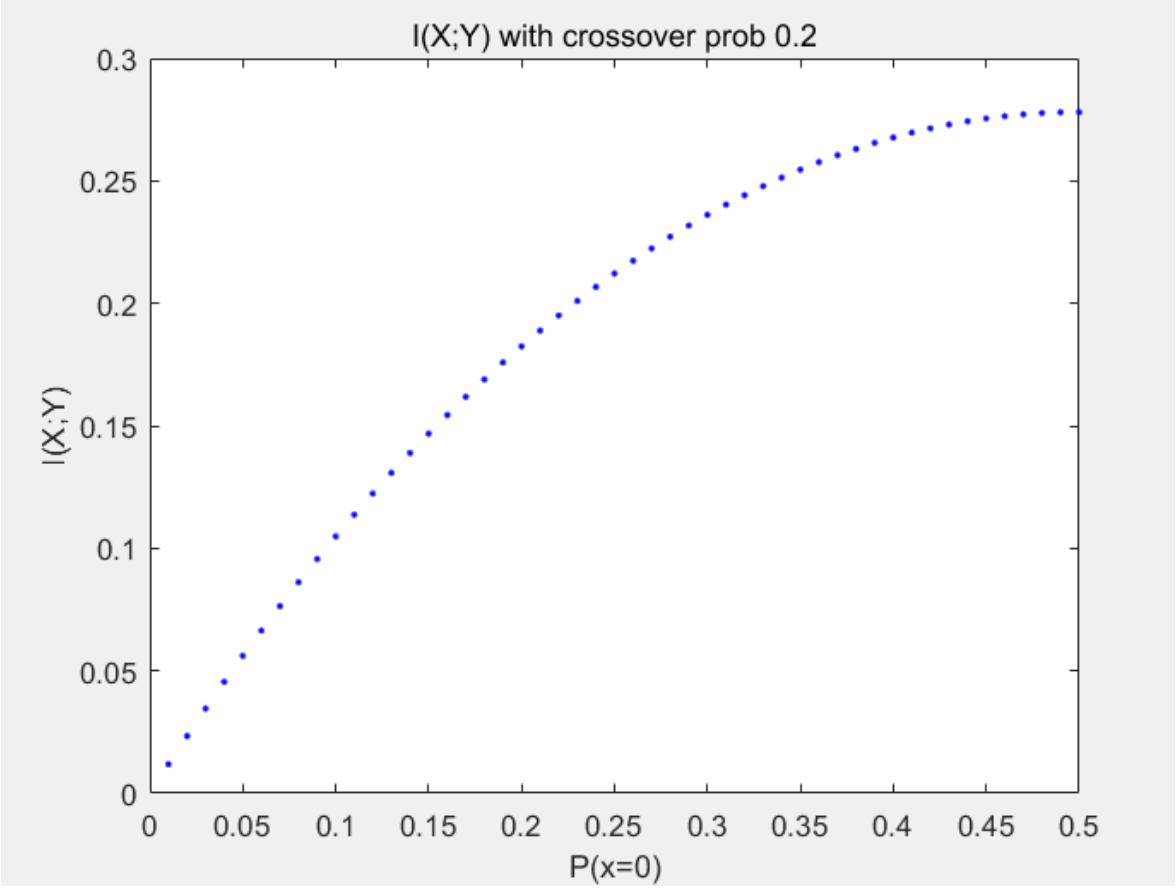

Figure 4.1 Mutual Information Plot

Combined with the plot of mutual information, we can have a guess for the shape of the achievable rate region shown in Figure 4.2. This figure is given by the convex hull of four curves. Figure 4.2 is only a guess of the shape for the achievable rate region and is not necessarily the same with the real shape of the region.

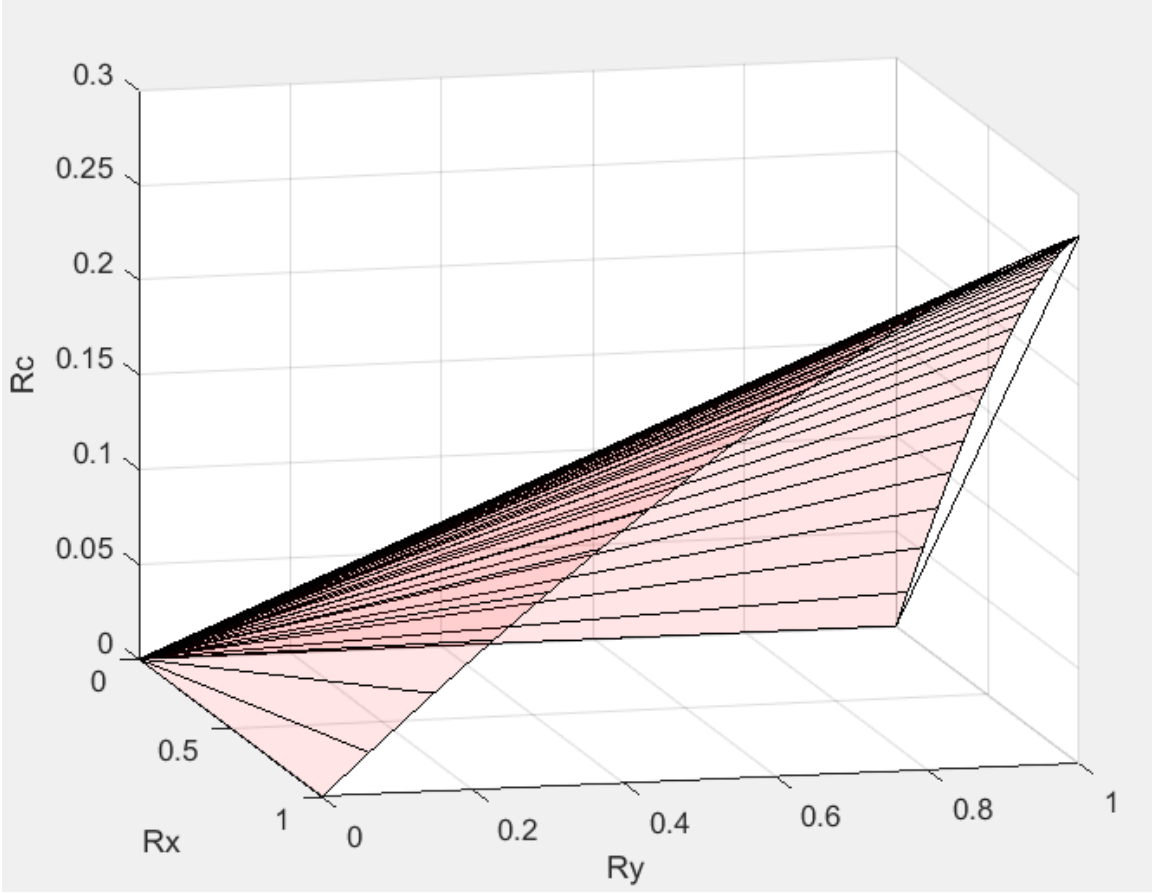

Figure 4.2 Guess about the shape of achievable rate region

Another observation about the result from 3.3.4 is, we can find some rules in the vertices. We can see that the corner points happened when

$$p(u|x) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{4.1}$$

In other words, the vertex only happens when the auxiliary variable(s) is(are) equal to the original variable(s). With this observation, we can assume that the boundary line of the achievable rate region must contain two curve representing U=X and V=Y respectively.

4.1.2 Alternative Optimization Statement

With the assumption we made in 4.1.1, we can then state the sub-problem of the original problem we discussed in previous sections. This sub-problem will serve as an alternative optimization problem in later discussions.

We suggest the following optimization problem:

$$\begin{aligned} &\max I(V;X) \\ &\text{s.t.} \quad \mathrm{I(V;Y)} \leq R_y \\ &\mathrm{U = X} \\ &\mathrm{X - Y - V} \end{aligned} \tag{4.2}$$

Apparently, we will need to modify this to some optimization problem that can derive an algorithm from.

The problem (4.2) is equivalent to

$$\begin{aligned} &\max_{p(v|y)} I(V;X) + \lambda I(V;Y) \\ &\text{s.t.}\, \mathrm{X - Y - V} \end{aligned} \tag{4.3}$$

The algorithm for this also involve some other algorithms. In later sections, we will introduce the algorithm for solving similar problems and derive the algorithm for (4.3).

4.2 Related Optimization Algorithm

The algorithm introduced in this section is firstly presented in [some reference]. This algorithm solves the optimization problem:

$$\max_{y(v|x)} \max_{p(y|v)} \sum_{y} \sum_{x} p(v|y)p(x,y) \log \frac{p(v|x)p(y|v)}{p(v|y)p(y)} \tag{4.4}$$

Algorithm:

- Initial guess for $p^{(0)}(v|x)$, $\mathrm{k} = 1$
- Update $p(y|v)$

$$p^{(k)}(y|v) = \frac{\sum_x p(v|y)p(x,y)}{\sum_{y'} \sum_x p(v|y')p(x,y')} \tag{4.5}$$

- Update $p(v|x)$

$$p^{(k+1)}(v|x) = \frac{\sum_y p(v|y)p(x,y)}{\sum_{v'} \sum_y p(v'|y)p(x,y)} \tag{4.6}$$

- If converges, stop; else, take $\mathrm{k} = \mathrm{k} + 1$, repeat.

4.3 Derive Algorithm

The algorithm shown in 4.2 is similar to our alternative optimization problem (4.2) without the constrains on $\mathrm{I(V;Y)} \leq R_y$. We can write (4.3) into the form similar to (4.4) with this constrain on $\mathrm{I(V;Y)}$, this will lead us to:

$$\max_{p(v|y)} \left[\max_{p(v|x)} \max_{p(y|v)} \sum_{v,y} \sum_{x} p(v|y)p(x,y) \log \frac{p(v|x)p(y|v)}{p(v|y)p(y)} \right. \tag{4.7}$$

$$\left. + \lambda \max_{p(y|v)} \sum_{v} \sum_{y} p(v|y)p(y) \log \frac{p(y|v)}{p(y)} \right]$$

In (4.7), we are maximizing over $p(y|v)$ for twice. For simplicity, we merge these two maximizations into one problem and yields:

$$\max_{p(v|y)} \max_{p(y|v)} \left[\left(\max_{p(v|x)} \sum_{v,y} \sum_{x} p(v|y)p(x,y) \log \frac{p(v|x)p(y|v)}{p(v|y)p(y)} \right) + \lambda \sum_{v} \sum_{y} p(v|y)p(y) \log \frac{p(y|v)}{p(y)} \right] \tag{4.8}$$

For this problem, we can assume that updating $p(v|x)$ step is the same with (4.6), and what left for us to solve is the updating $p(v|y)$ and $p(y|v)$. To solve what the $p^*(y|v)$ should be, take the derivative with respect to $p(y|v)$ then solve for $p^*(y|v)$. The derivate of (4.8) with respect to $p(y|v)$ is given by:

$$\sum_{x} p(v|y)p(x,y) \frac{1}{p(y|v)} + \lambda p(v|y)p(y) \frac{1}{p(y|v)} \tag{4.9}$$

We take the derivative to be some function that makes it zero.

$$\frac{1}{p(y|v)} \left[[p(v,y)](1+\lambda) \right] = \nu(v) \tag{4.10}$$

Then we can solve for the $p^*(y|v)$.

$$\mathrm{p}^{(*)}(y|v) = \frac{p^{(*)}(v|y)p(y)}{\sum_{y'} p^{(*)}(v|y')p(y')} \tag{4.11}$$

After solving for the $p^*(y|v)$, we can know find the optimal solution for the $p(v|y)$. We apply similar method to solve this optimization problem: take the derivative and set it to zero. The derivative should be like this:

$$\sum_x p(x,y)\log\frac{p^{(k)}(v|x)p^{(k)}(y|v)}{p(v|y)p(y)} - \sum_x p(x,y) + \lambda p(y)\log\frac{p^{(k)}(y|v)}{p(y)} + \gamma(y) = 0 \tag{4.12}$$

In (4.), the $\gamma(y)$ is some function that makes the solution a probability distribution. Set the derivative to zero gives us:

$$\begin{aligned} \mathrm{p}^{(k+1)}(v|y) &= \left[\frac{p^{(k)}(y|v)}{p(y)}\right]^{1+\lambda} e^{\left[\frac{1}{p(y)}\sum_x p(x,y)\log p^{(k)}(v|y)-1+\gamma(y)\right]} \\ &= \frac{\left[\frac{p^{(k)}(y|v)}{p(y)}\right]^{1+\lambda} e^{[\sum_x p(x|y)\log p^{(k)}(v|x)]}}{\sum_{v'}\left[\frac{p^{(k)}(y|v)}{p(y)}\right]^{1+\lambda} e^{[\sum_x p(x|y)\log p^{(k)}(v|x)]}} \end{aligned} \tag{4.13}$$

We can simplify this to:

$$p^{(k+1)}(v|y) = \frac{\left[p^{(k)}(y|v)\right]^{1+\lambda} e^{[\sum_x p(x|y)\log p^{(k)}(v|x)]}}{\sum_{v'}[p^{(k)}(y|v)]^{1+\lambda} e^{[\sum_x p(x|y)\log p^{(k)}(v|x)]}} \tag{4.14}$$

Then, we will have our algorithm for solving (4.3):

Algorithm for sub-problem:

- Initial guess for $p^{(0)}(v|y)$, set $\mathrm{k} = 0$
- Update $p(v|x)$

$$p^{(k)}(v|x) = \frac{\sum_y p(v|y)p(x,y)}{\sum_{v'}\sum_y p(v'|y)p(x,y)} \tag{4.15}$$

- Update $p(y|v)$

$$p^{(k)}(y|v) = \frac{p^{(k)}(v|y)p(y)}{\sum_{y'} p^{(k)}(v|y')p(y')} \tag{4.16}$$

- Update $p(v|y)$

$$p^{(k+1)}(v|y) = \frac{\left[p^{(k)}(y|v)\right]^{1+\lambda} e^{\left[\sum_x p(x|y)\log p^{(k)}(v|x)\right]}}{\sum_{v'}\left[p^{(k)}(y|v')\right]^{1+\lambda} e^{\left[\sum_x p(x|y)\log p^{(k)}(v'|x)\right]}} \tag{4.17}$$

- If converges, stop; else, set

4.4 MATLAB Implementation

The MATLAB code for algorithm above is shown below. This function will take the joint probability distribution of X and Y and solve for the optimal $p(v|y)$. Then compute the rate R_y and R_c by (3.1) and (3.2). The Lagrange multiplier $\boldsymbol{\lambda}$ takes value from -1 to 1, as a result of the $(1+\lambda)$ part in the algorithm. In this problem, the R_x will always take the value of 1 and thus we do not display the plot with R_x. For simplicity, we only show the last two steps of this algorithm.

```
for n = 1:sv
        for j = 1:sy
            P_yv(j,n) = sum(P_vy(n,j) .* P_x_y(:,j));
        end
        P_yv(:,n) = P_yv(:,n)/sum(P_yv(:,n));
    end
for j = 1:sy
        P_vy_1(:,j) = (P_yv(j,:)).^(1+L)' .*
exp(log_2(P_vx)*P_xy(:,j));
        P_vy_1(:,j) = P_vy_1(:,j)/sum(P_vy_1(:,j));
    end
```

4.5 Result

To check the accuracy of our algorithm, we first test our algorithm with the following probability distributions:

$$\mathcal{X} = \{0,1\}$$

$$\mathrm{P}(\mathrm{x} = 0) = \mathrm{P}(\mathrm{x} = 1) = 0.5 \tag{4.18}$$

$$\mathrm{Y} = \mathrm{X} \oplus \mathrm{W}, \mathrm{W} \sim \mathrm{Bonuli}(p), p = 0.2$$

We will use $|\mathcal{V}| = 2$ for our first test and in following sections.

The result of our algorithm is shown in Figure 4.3. The blue curve is the curve generated by our algorithm, the red curve is the line between the minimum and the maximum of the blue curve.

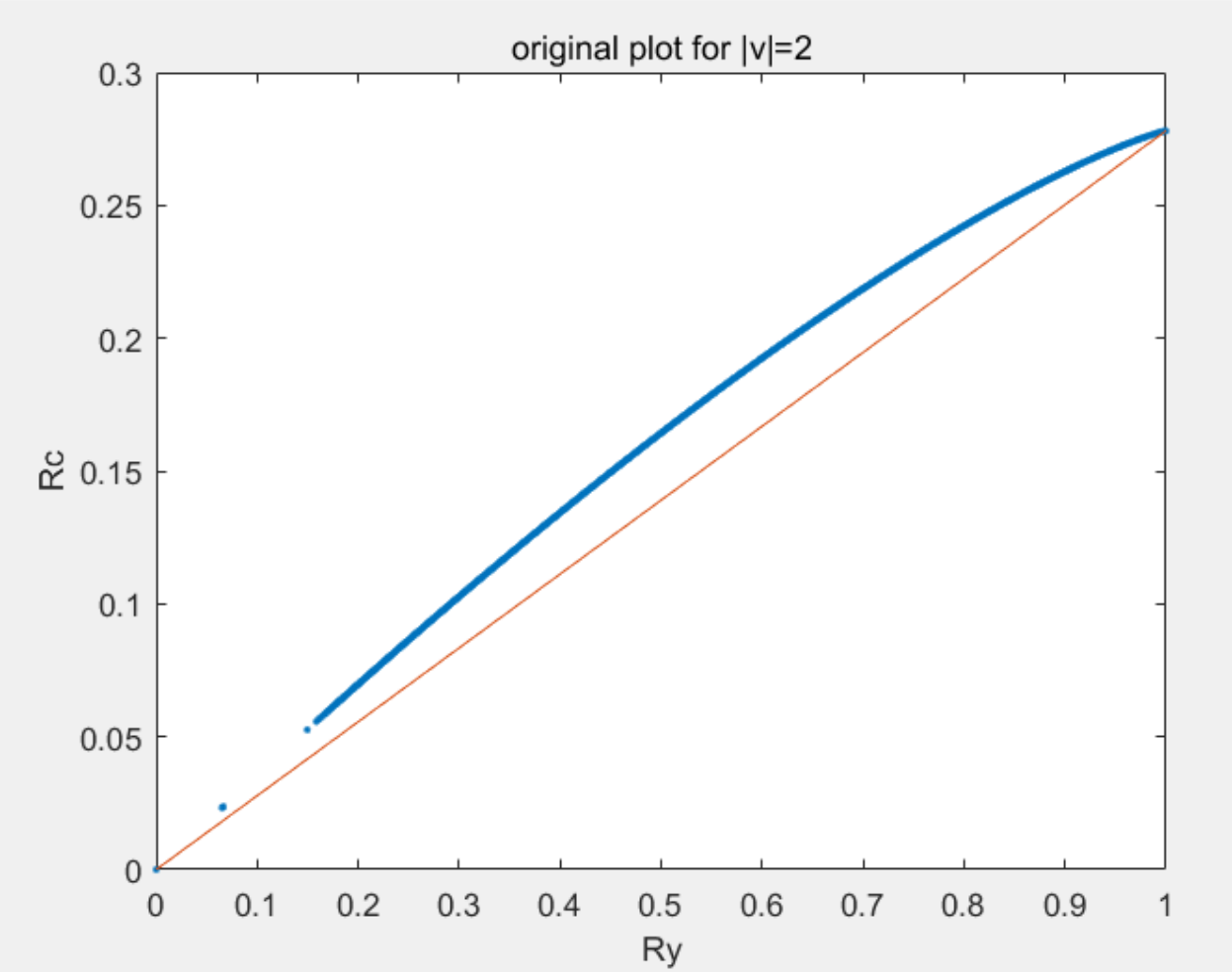

Figure 4.3 The plot for Ry-Rc

The curve is plotted by points instead of line segments.

From Figure 4.3, we can see that the curve has similar trend with Figure 4.1, but not exactly the same. The result shows a good start of our algorithm. In later sections, we will have some further analyzations for this algorithm, and combine this curve with our previous algorithm to seek a way to compute the achievable rate region.

4.6 Further analysis

With the result from 4.5, we formulate an algorithm that can calculate the upper bound for the achievable rate with one auxiliary random variable fixed. We plot the curve in Figure 4.3 into a 3D plot.

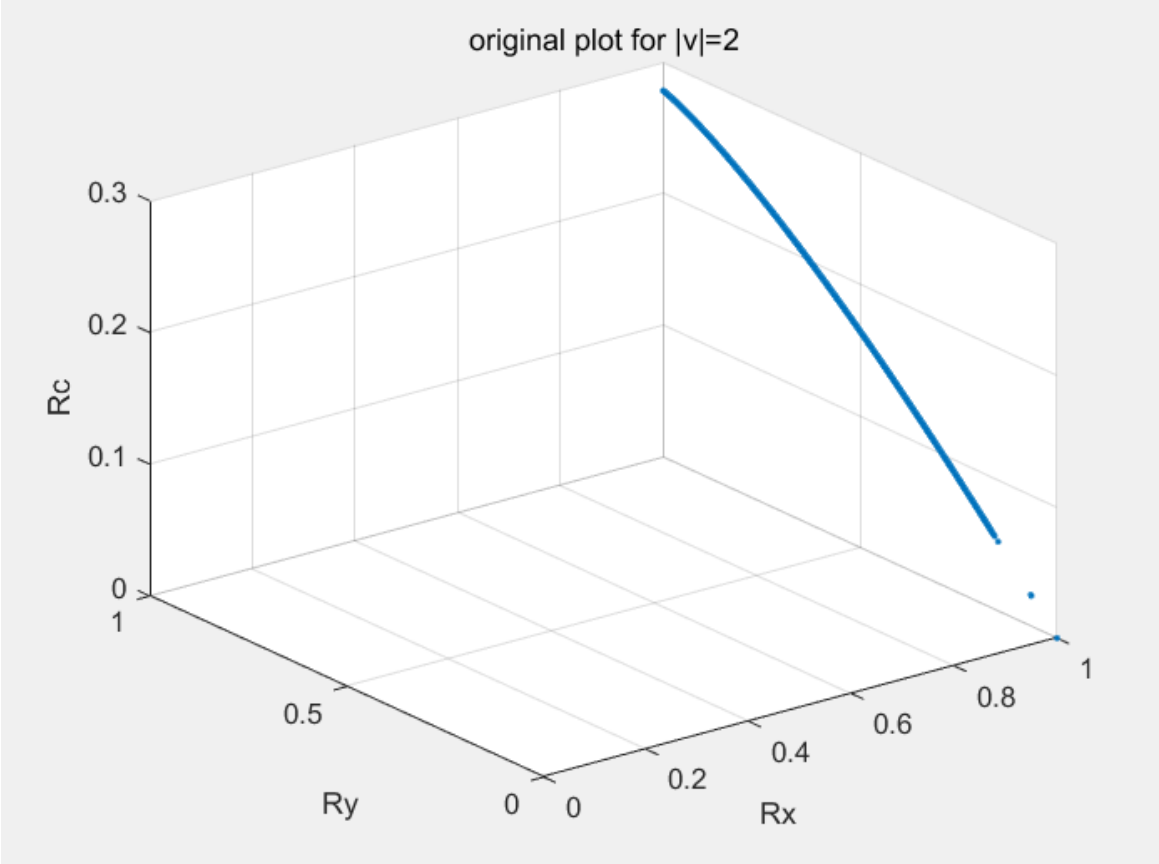

Figure 4.4 3D plot for Ry-Rc curve

Then, for every point on this curve, we should be able to generate a $R_x - R_c$ curve with the auxiliary variable V fixed. The algorithm we use to generate this curve is the same algorithm we have shown in chapter 3. We use

$$p^{(k+1)}(u|x)$$
$$= \frac{e^{((\nu_x+\lambda_x)\log p^{(k)}(u|x)-\lambda_x q^{(k)}(u)+\sum_v(\sum_y p(y|x)p(v|y))\log q^{(k)}(u|v))}}{\sum_{u'} e^{((\nu_x+\lambda_x)\log p^{(k)}(u'|x)-\lambda_x q^{(k)}(u')+\sum_v(\sum_y p(y|x)p(v|y))\log q^{(k)}(u'|v))}} \quad (4.19)$$

To compute this curve. The surface formulated by those curves is the achievable rate region we need to compute in this thesis. The result is shown in Figure 4.5.

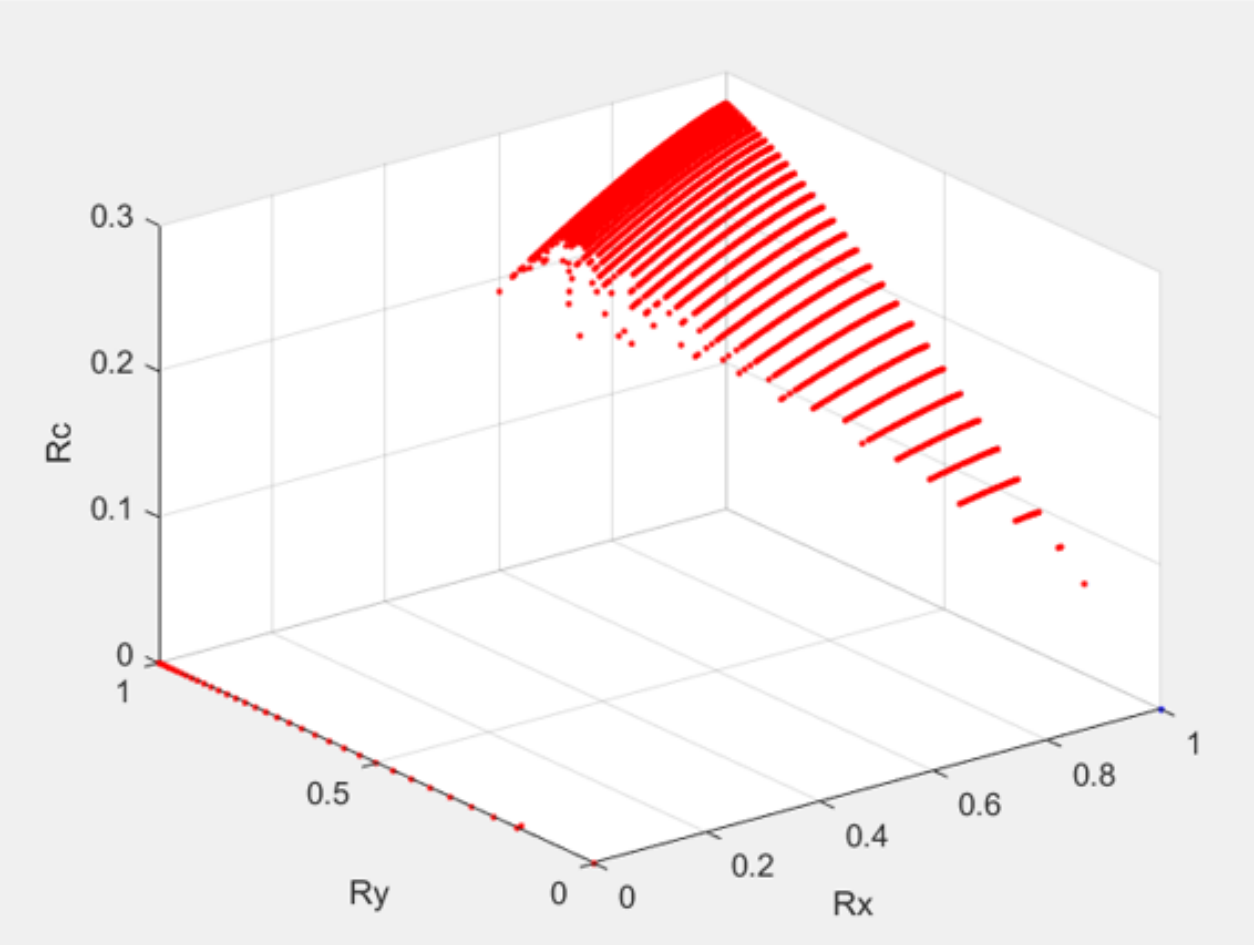

Figure 4.5 Achievable rate region plot

From the Figure 4.5 we can see that we have a surface with the down left part missing. This is because of a poor choice of the starting point and the multiplier for the slow convergence part. To generate a better shaped surface for the achievable rate region, we would suggest finding better starting point and slow convergence multiplier combination. Also, this surface is only generated from the $R_y - R_c$ curve. By symmetric, we assume the $R_x - R_c$ curve should be the same. For a

more accurate surface plot, we suggest using both curves and adopt similar method to generate the achievable rate region.

Chapter 5

Conclusion

In this thesis, we introduced the pattern recognition system and its achievable rate region. We introduced the formulas for the achievable rate and two approaches for getting the surface of the region. Then, we introduced the Blahut-Arimoto's algorithm for computing the channel capacity and rate-distortion function. With these algorithms, we derived our first two algorithm attempts for computing the achievable rate region. Although these two algorithms cannot generate out desirable surface, upper corner points generated by the algorithm is proven to be correct.

Then, we considered a sub-problem of the original problem, namely, compute the R_y with fixed $p(u|x)$. The sub-problem is proven to be easier to solve. We derived an algorithm for solving the sub problem and have a basic guess for the shape of the achievable rate region. Then we showed that we could adopt some methods to generate the surface of the achievable rate region from the result of the sub-problem.

The challenge for computing the achievable rate region as a whole optimization problem, as what we presented in chapter 3, lies in the part where we need to multiply the two variables during the algorithm. Since we are doing optimization over both variables, this might result in the algorithm only converge to the corner point. For a complete algorithm, we would consider an algorithm that optimize over one variable while fixed the other one with some optimal value.

For future research direction, we could consider finding better starting point and corresponding multipliers to generate better surface of the achievable rate region.

References

[1] Westover M B, O'Sullivan J A. Achievable rates for pattern recognition[J]. IEEE Transactions on Information Theory, 2008, 54(1): 299-320.

[2] Blahut R. Computation of channel capacity and rate-distortion functions[J]. IEEE transactions on Information Theory, 1972, 18(4): 460-473.

[3] O'Sullivan J A. Alternating minimization algorithms: from Blahut-Arimoto to expectation-maximization[M]//Codes, Curves, and Signals. Springer, Boston, MA, 1998: 173-192.

[4] Cover T M, Thomas J A. Elements of information theory[M]. John Wiley & Sons, 2012.

[5] Nasrabadi N M. Pattern recognition and machine learning[J]. Journal of electronic imaging, 2007, 16(4): 049901.

[6] Niesen U, Shah D, Wornell G. Adaptive alternating minimization algorithms[C]//Information Theory, 2007. ISIT 2007. IEEE International Symposium on. IEEE, 2007: 1641-1645.

[7] O'Sullivan J A. Iterative algorithms for maximum likelihood sequence detection[M]//Codes, Graphs, and Systems. Springer, Boston, MA, 2002: 137-156.

[8] Lecture Note from ESE 529, Advanced Topics in Information Theory, 2012,11,06.