An Implementation of Constant Composition Distribution Matching

Luo Yu 516030910530

Abstract— **Distribution matching transforms independent and Bernoulli(1/2) distributed input bits into a sequence of output symbols with a desired distribution. Fixed-to-fixed length, invertible, and low complexity encoders and decoders based on constant composition and arithmetic coding are presented.**

I. INTRODUCTION

A DISTRIBUTION MATCHER transforms independent Bernoulli(1/2) distributed input bits into output symbols with a desired distribution. A dematcher performs the inverse operation and recovers the input bits from the output symbols. There will be a brief introduction about the background of distribution problem and CCDM in section 2 and we shall formulate the problem in section 3. In section 4, we will explain the meaning of constant composition and describe our algorithm. The algorithm implementation and result demo will be after that.

II. BACKGROUND AND RELATED WORKS

Constant composition distribution matching was introduced in [1], with the basic idea of this algorithm and its arithmetic coding. This project was determined to achieve the algorithm it introduced in Python, and make a supplement of this algorithm. This supplement includes the arrangement of the composition bag, achievement of bootstrap scheme.

III. PROBLEM STATEMENT

A one-to-one f2f distribution matcher is an invertible function f. We denote the inverse function by f^{-1}.The mapping imitates a desired distribution P_A by mapping m Bernoulli(1/2) distributed bits B^m to length n strings $\tilde{A}^n = f(B^m) \in A^n$. The output distribution is$P_{\tilde{A}^n}$. The concept of one-to-one f2f distribution matching is illustrated in Fig. 1.

A. Problem Formulation

Firstly, the basic properties of a matcher should be determined, including the input length, output length, output amplitude and empirical distribution. As long as the construction of distribution matcher is finished, we can input Bernoulli(1/2) distributed bits with length k and get an output sequence with length n. The output sequence should be with amplitude M and distribution P_A. For example, if $k = 10, M = 3$ (which indicated that the number of items in P_A should be 3) and $P_A = [0.3, 0.3, 0.4]$, an ideal output should be like $[1, 1, 1, 2, 2, 2, 2, 0, 0, 0]$.

*This work was not supported by any organization

[1]H. Kwakernaak is with Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, 7500 AE Enschede, The Netherlands h.kwakernaak at papercept.net

[2]P. Misra is with the Department of Electrical Engineering, Wright State University, Dayton, OH 45435, USA p.misra at ieee.org

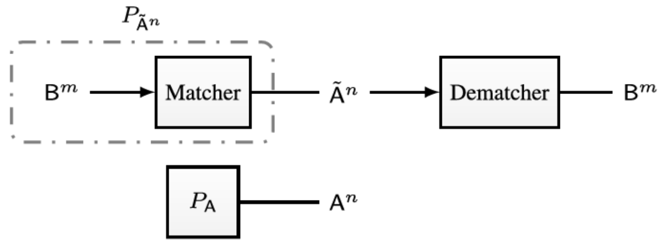

Fig. 1. Matching a data block $B^m = B_1...B_m$ to output symbols $\tilde{A}^n = \tilde{A}^1...\tilde{A}^n$ and reconstructing the original sequence at the dematcher.

B. Result Measurement

The entropy of a discrete random variable A with distribution P_A is

$$H(A) = \sum_{a \in supp(P_A)} -P_A(a) log_2 P_A(a) \tag{1}$$

where $supp(P_A) \subseteq A$ is the support of P_A. The informational divergence of two distributions on A is

$$D(P_{\hat{A}} \parallel P_A) = \sum_{a \in supp(P_{\hat{A}})} P_{\hat{A}}(a) log_2 \frac{P_{\hat{A}}(a)}{P_A(a)} \tag{2}$$

The normalized informational divergence for length n random vectors $\hat{A}^n = \hat{A}_1...\hat{A}_n$ and A^n is defined as

$$\frac{D(P_{\hat{A}^n} \parallel P_{A^n})}{n} \tag{3}$$

IV. CONSTANT COMPOSITION DISTRIBUTION MATCHING

The empirical distribution of a vector c of length n is defined as

$$P_{\bar{A},c}(a) = \frac{n_a(c)}{n} \tag{4}$$

where $n_a(c) = |\{i : c_i = a\}|$ is the number of times symbol a appears in c. People call $P_{\bar{A},c}$ the type of c. An n-type is a type based on a length n sequence. A code book Cccdm$\subseteq$ A^n is called a constant composition code if all codewords are of the same type, i.e., $n_a(c)$ does not depend on the codeword c. The algorithm will be introduced below.

A. Constant Composition Determination

With length n and distribution P_A, we may determine the constant compositions by multiply them. For example, $n = 10$ and $P_A = [0.3, 0.3, 0.4]$, the constant compositions should be $[0, 0, 0], [1, 1, 1], [2, 2, 2, 2]$.

B. Code Book Establishment

We describe the code book with intervals. With the idea of Huffman tree, we lengthen the codes while divide intervals into smaller ones continually. Initially, the input interval spans from 0 to 1. As the input model is Bernoulli(1/2) we split the interval into two equally sized intervals and continue with the upper interval in case the first input bit is 1; otherwise we continue with the lower interval. After the next input bit arrives we repeat the last step. After m input bits we reach a size 2^{-m} interval. After every refinement of the input interval the algorithm checks for a sure prefix of the output sequence. Every time we extend the sure prefix by a new symbol, we must calculate the probability of the next symbol given the sure prefix. That means we determine the output intervals within the sure interval of the prefix. The model for calculating the conditioned probabilities is based on drawing without replacement. There is a bag with n symbols of k discriminable kinds. n_a denotes how many symbols of kind a are initially in the bag and n'_a is the current number. The probability to draw a symbol of type a is n'_a/n. If we pick a symbol a both n and n'_a decrement by 1.

Considering an example of input interval with $k = 4$, and there are $[0, 0, 1, 1]$ in the composition bag initially. See Fig. 2.

Initially there are 2 0s and 2 1s in the bag.The distribution of the first drawn symbol is $P_{A_1}(0) = P_{A_1}(1) = \frac{1}{2}$. When drawing a 0, there are 3 symbols remaining: one 0 and two 1s. Thus, the probability for a 0 reduces to 1/3 while the probability of 1 is 2/3. If two 0s were picked, two 1s must follow. This way we ensure that the encoder output is of the desired type. Observe that the probabilities of the next symbol depend on the previous symbols, e.g., we have

$$P_{A_2|A_1}(0|0) \neq P_{A_2|A_1}(0|1) \qquad (5)$$

in general.

As for the output, this constructing progress still works. An example is in Fig. 3, with composition bag $[1, 2, 3, 4]$.

C. Encoding and Decoding

The arithmetic encoder can link an output sequence to an input sequence if the lower border of the output interval is inside the input interval. In the example (Fig. 4) 00 may link to both 0101 and 0011, while for 01 only a link to 0110 is possible. There are at most two possible choices because by the input interval size is less than twice the output interval size. Both choices are valid and we can perform an inverse operation. In our implementation, the encoder decides for the output sequence with the lowest interval border. As a result, the codebook C_{ccdm} of this Example is 0011, 0110, 1001, 1100.

D. Propositions and Proof

From the examples above, we can draw that the number of output intervals must be less than the number of input intervals so that the encoding process can be achieved. But

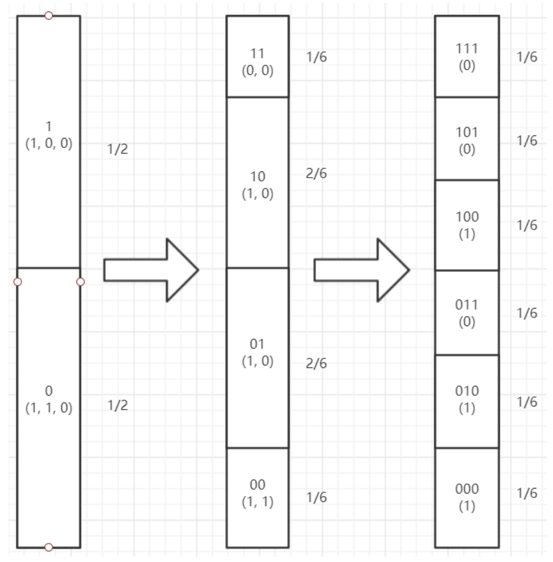

Fig. 2. The code book construction with composition bag $[0, 0, 1, 1]$.

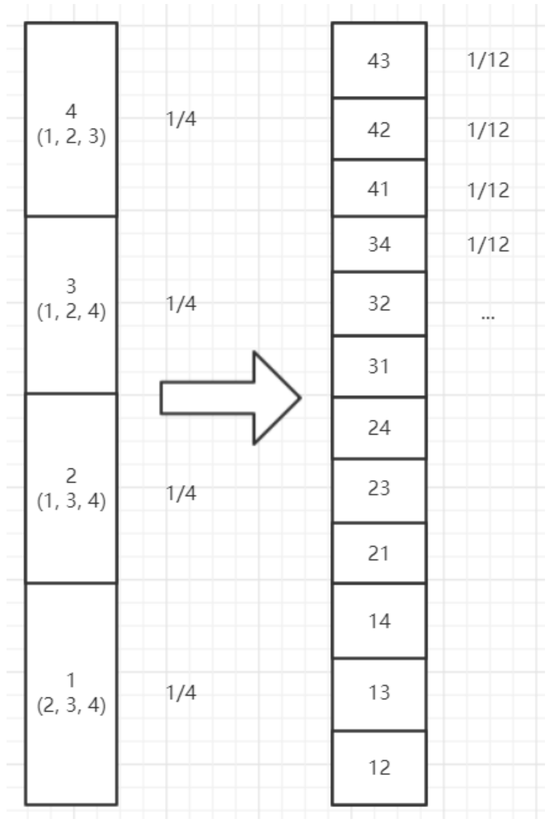

Fig. 3. The construction process with composition bag $[1, 2, 3, 4]$.

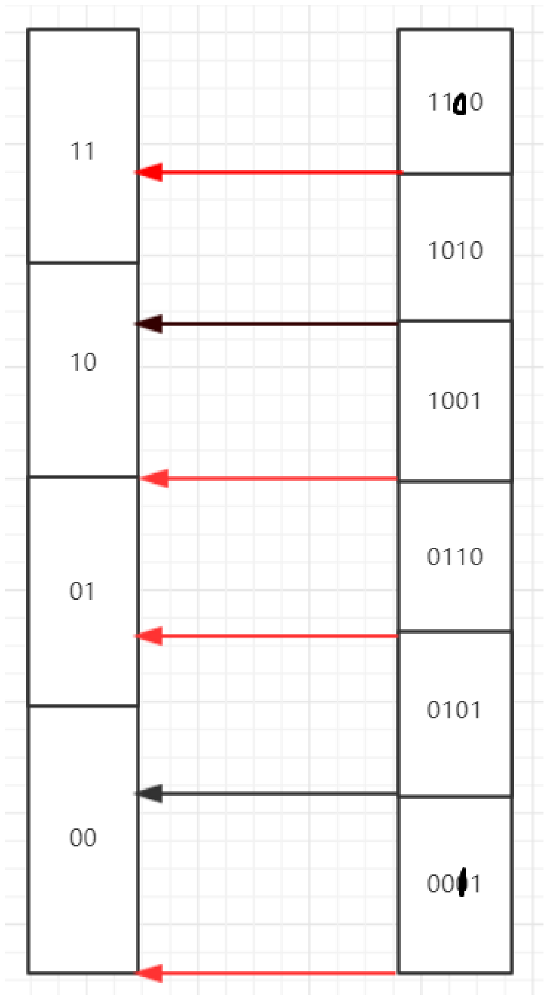

Fig. 4. An encoding/decoding example.

there's still another condition under which this encoding process is achievable.

Proposition: After all the refinements, intervals share a same interval size (possibility).

Proof: Consider a composition bag, where each kind of symbol only appear once. Under this condition every output symbol share a same possibility apparently. If we observe 2 elements like 1 and 2, and reduce the intervals like this: if two symbols is different only because 1 and 2 swapped, delete one of them and add its interval size to another. The result after this reduction is still that every symbol share a same possibility apparently. Keep going and this proposition is proved.

V. ALGORITHM IMPLEMENTATION

In this section, we will introduce how we implemented this algorithm, including constructing composition bags and intervals, and the process of encoding or decoding.

A. Composition Bag

We consider an approximation algorithm to construct the composition bag we need. Simply multiply output length n and the distribution P_A, ignore the fractional parts and make up on the last item. For example, with $n = 10$ and $P_A = [0.55, 0.33, 0.12]$, the bag would be $[5, 3, 2]$, which means that there are 5 0s, 3 1s and 2 2s in the bag initially.

B. Code Book Establishment

Take the output interval for example. Construct two list, one to record the size of each interval, and the other to record the current symbol of each interval. If the composition bag is with value [1, 1, 1, 1], which means that there are 1 0, 1 1, 1 2 and 1 3 in the bag initially, the initial value of outputInterval_symbol and outputInterval_size should be [[]] and [1], and the result after refinements should be [[1, 2], [1, 3], [1, 4], [2, 1], [2, 3], [2, 4], [3, 1], [3, 2], [3, 4], [4, 1], [4, 2], [4, 3]], [1/12, 1/12, 1/12, 1/12, 1/12, 1/12, 1/12, 1/12, 1/12, 1/12, 1/12, 1/12].

The key of our algorithm implementation is using mutex and threads. Our interval partition function can partition an interval into intervals and generate new symbols and sizes, with given symbol to operate and corresponding composition bag. Any operation on the interval lists shall influence the indexes of items in them, so we must use a mutex to ensure that there is only one thread operating on intervals. The first thread shall operate on [[]] and [1] with target symbol [] and bag [1, 1, 1, 1]. After partition, this thread will creates new threads to operate on symbols it generated.

VI. CONCLUSIONS AND CONSIDERABLE IMPROVEMENT

This project achieved the algorithm of CCDM successfully, but there's still more aspects which require improvement. We'll propose them below.

A. Composition Bag Constructing Improvement

The basic objective while constructing composition bag is to make its distribution as similar to the ideal distribution as possible. Our current algorithm is too rough to achieve this objective. There are plenty of ideas about improve its performance. For example, design a random function which will determine which item in the bag will be 'plus one' according to its possibility.

B. Code Book Establishment Algorithm Optimization

Though we use threads to achieve code book establishment, this implementation is still single threaded, because the mutex limited that only one thread may operate on global variables - the interval lists. To improve efficiency, we may consider optimize the use of mutex, or implement this algorithm with linked list instead of array, which shall allow us to operate on different symbols (intervals) at the same time.

C. Algorithm Performance Evaluation

In section 3, we have introduced how to measure the informational divergence of two distribution. To evaluate the performance of our algorithm implementation, we need to test coding with length greater than 100 for hundreds of times, which is hard to accomplish under existing conditions. This can only be fixed after code book establishment algorithm optimization.

References

[1] P. Schulte and G. Bcherer, ”Constant Composition Distribution Matching,” in IEEE Transactions on Information Theory, vol. 62, no. 1, pp. 430-434, Jan. 2016.