A REPORT

ON

SEARCH TECHNIQUES FOR DISTRIBUTED COMPRESSION

BY

Pushpendu Ghosh **2015B4A7366G**

AT

INDIAN INSTITUTE OF SCIENCE, BANGALORE

ECE Department

A Practice School-I station of

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

(June 2017)

A REPORT

ON

SEARCH TECHNIQUES FOR DISTRIBUTED COMPRESSION

BY

Pushpendu Ghosh **2015B4A7366G** **Mathematics Computer Science**

Prepared in partial fulfilment of the
Practise School-I Course

AT

INDIAN INSTITUTE OF SCIENCE, BANGALORE
ECE Department

A Practice School-I station of

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

(June 2017)

Acknowledgement

I would like to make use of this opportunity to convey my sincerest and heartfelt gratitude to my mentor Dr. Himanshu Tyagi for giving me the chance to do this project under his guidance. I earnestly thank him for providing me with the prerequisite knowledge, guiding me and giving me valuable advices. At the same time, I thank Prof. K.R.Anupama and the Practice school division to give me the opportunity to work under such a reputed institution of India. I would also like to thank Prof. Satya Sudhakar Yedlapalli, PS-I incharge, for his help in presenting and documenting the research work. This project gave me an extensive research experience to tackle various challenges in implementing known theories to a real life problem. I would also like to express my gratitude to Mr. Sowmya Banerjee, my friends, fellow students and family for their help and support throughout the project.

I thank you yet again.

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE
PILANI, RAJASTHAN

Practise School Division

Station : IISc Bangalore
Centre : E.C.E Department
Duration : 8 weeks
Date of Start : 22nd May 2017
Date of submission :

Title of the project : Search techniques for distributed compression

Name : Pushpendu Ghosh
BITS ID : 2015B4A7366G
Discipline : MSc. in Mathematics & B.E in Computer Science Engineering

Mentor name : Dr. Himanshu Tyagi
Designation : Assistant professor, Dept of ECE, IISc Bangalore

PS Faculty name : Dr. Satya Sudhakar Yedlapalli

Key Words : Optimisation, Locality sensitive Hashing

Project area : Coding theory

Abstract : The project is about reducing the communication between sender and reciever, when the data is constrained. Several optimisation algorithms were analysed to find the nearest neighbours of the sent data. Using various hashing techniques (especially LSH), the information bits sent is hashed into hash code (of fewer bits) and then this hash code is sent to the receiver. Receiver then searches for this hash code in the sample space to find the nearest neighbours of the sent code.

Signature of student
Date

Signature of PS faculty
Date

TABLE OF CONTENT

CHAPTER 1

INTRODUCTION

Distributed source coding (DSC) is an important problem in information theory and communication. DSC problems regard the compression of multiple correlated information sources that do not communicate with each other. By modeling the correlation between multiple sources at the decoder side together with channel codes, DSC is able to shift the computational complexity from encoder side to decoder side, therefore provide appropriate frameworks for applications with complexity-constrained sender, such as sensor networks and video/multimedia compression. One of the main properties of distributed source coding is that the computational burden in encoders is shifted to the joint decoder.

In 1973, David Slepian and Jack Keil Wolf proposed the information theoretical lossless compression bound on distributed compression of two correlated i.i.d. sources X and Y. After that, this bound was extended to cases with more than two sources by Thomas M. Cover in 1975, while the theoretical results in the lossy compression case are presented by Aaron D. Wyner and Jacob Ziv in 1976.Although the theorems on DSC were proposed on 1970s, it was after about 30 years that attempts were started for practical techniques, based on the idea that DSC is closely related to channel coding proposed in 1974 by Aaron D. Wyner.

In this project, DSC has been applied using various hashing techniques to find the nearest neighbours of the sent code. The project deals with an open ended research problem, to minimise the communication and search the space most optimally in a limited time frame. So to achieve the target the project is divided into two parts : Analysis of meta-heuristic searches and Efficiency of Random Projection LSH. The experiments conducted throughout the project to test the efficiency of the algorithms were done using MATLAB_R2016b as computation platform.

CHAPTER 2

OPTIMIZATION

2.1. Particle Swarm Optimisation (PSO)

Particle swarm optimization (PSO) is a population based stochastic optimization technique developed by Dr. Eberhart and Dr. Kennedy in 1995, inspired by social behavior of bird flocking or fish schooling. PSO is a metaheuristic as it makes few or no assumptions about the problem being optimized and can search very large spaces of candidate solutions. However, metaheuristics such as PSO do not guarantee an optimal solution is ever found.

2.1.1. Algorithm

PSO is initialised with a group of random particles (solutions) and then searches for optima by updating generations. In every iteration, each particle is updated by following two "best" values. The first one is the best solution (fitness) it has achieved so far. (The fitness value is also stored.) This value is called pbest. Another "best" value that is tracked by the particle swarm optimiser is the best value, obtained so far by any particle in the population. This best value is a global best and called gbest.

After finding the two best values, the particle updates its velocity and positions with the below equations (1) and (2).

$v_i[t+1] = v_i[t] + c1 * rand * (pbest_i[] - x_i[t]) + c2 * rand * (gbest [] - x_i[t])$ … (1)

$x_i[t+1] = x_i[t] + v_i[t+1]$ … (2)

$v_i[t]$ is the velocity of the i^{th} particle at time t, $x_i[]$ is the current position of the i^{th} particle (solution). $pbest_i[]$ and gbest [] are defined as stated before. rand is a random number between (0,1). c1, c2 are learning factors. usually c1 = c2 = 2.

2.1.2. Pseudo-code

For each particle
Initialize particle
END

Do
For each particle
Calculate fitness value
If the fitness value is better than the best fitness value (pBest) in history

set current value as the new pBest
End

Choose the particle with the best fitness value of all the particles as the gBest
For each particle
Calculate particle velocity according equation (1)
Update particle position according equation (2)
End

While maximum iterations or minimum error criteria is not attained

2.1.3. PSO Variations

2.1.3.1. Binary PSO

The BPSO algorithm was introduced by Kennedy and Eberhart to allow the PSO algorithm to operate in binary problem spaces. It uses the concept of velocity as a probability that a bit (position) takes on one or zero. In the BPSO, Eq. (1) for updating the velocity remains unchanged, but Eq. (2) for updating the position is re-defined by the rule :

$$x_i[t+1] = \begin{cases} 0 & \text{if rand} \geq S(v_i[t+1]) \\ 1 & \text{if rand} < S(v_i[t+1]) \end{cases} \quad \ldots (3)$$

where S(.) is the sigmoid function for transforming the velocity to the probability as the following expression :

$$S(v_i[t+1]) = 1 / (1 + \exp(-v_i[t+1])) \quad \ldots (4)$$

2.1.3.2. Modified Binary PSO

Modified Binary PSO uses the genotype–phenotype concept as follows: Let the velocity of the original BPSO be a continuous search space, and then a binary position can be decoded by the sigmoid function. Here, let the velocity and the binary position of the original BPSO be a genotype $x_{g,i}$ and a phenotype $x_{p,i}$, respectively. Then, the update functions of (1), (3), and (4) of the original BPSO are changed as the following expressions:

$$v_i[t+1] = v_i[t] + c1 * rand * (pbest_i[] - x_{p,i}[t]) + c2 * rand * (gbest[] - x_{p,i}[t]) \quad \ldots (6)$$

$$x_{g,i}[t+1] = x_{g,i}[t] + v_i[t+1] \quad \ldots (7)$$

$$x_{p,i}[t+1] = \begin{cases} 0 & \text{if rand} \geq S(x_{g,i}[t+1]) \\ 1 & \text{if rand} < S(x_{g,i}[t+1]) \end{cases} \quad \ldots (8)$$

where $S(x_{g,i}[t+1]) = 1 / (1 + \exp(-x_{g,i}[t+1]))$ … (9)

2.2. Simulated Annealing

Simulated annealing (SA) is a probabilistic technique for approximating the global optimum of a given function. Specifically, it is a metaheuristic to approximate global optimisation in a large search space. The name and inspiration come from annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects.

2.2.1. Algorithm

Parameters : x_i is the position at the i^{th} iteration, T_i is the temperature at the i^{th} iteration and $f(x_i)$ is the corresponding cost function of x_i .

It starts from a random point (x_0) and the value at the point is computed using the cost function. Initially the parameter 'temperature' (T_i) is kept very high (2000) and every iteration the temperature is cooled by 2% . Depending on the temperature the code generates a random neighbourhood (neib = x_i + (rand(-1,1)) * k * T_i), and calculates the corresponding cost of the neighbourhood. If f(neib) is optimal than $f(x_i)$ then x_{i+1} = neib, else even if neib is not optimal a probabilistic acceptance test determines whether to accept neib as the new solution or not. The more the temperature, the more is the probability to accept neib. The algorithm ends when the temperature is low, or the desired tolerance is achieved.

Probability Acceptance function :

$$\text{acceptance} = \left(\text{rand} \leq \frac{1}{1 + \exp\left(\frac{f(\text{neib}) - f(x_i)}{T_i}\right)} \right)$$

where acceptance is of boolean datatype, which tells if the new neighbourhood is accepted as a better optimal point or not. This probability acceptance method ensures the randomness of the heuristic search.

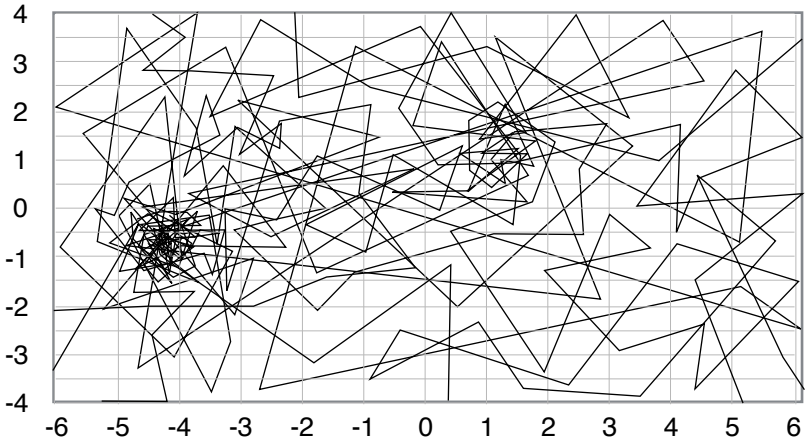


Figure 1 : Movement of particle (as per SA) in a 2D solution space

2.3. Performance Metrics

In the experiment, I applied the original BPSO, the modified BPSO and Simulated Annealing to 4 benchmark functions, and then compared their performance. I used an Intel Core i5 processor with a 2.4-GHz CPU and 2 GB of memory, and MATLAB as the computation platform. Both PSO algorithms used 100 iterations with 30 particles in the swarm and Simulated Annealing used 100 iterations with 30 repeated tries in each iteration.

2.3.1. Benchmark functions

Four Benchmark functions were used for the performance test. Most of these functions have many local optima, it is very hard to find a global optimum point with conventional numerical optimisation methods.

The details of the functions are as follows :

$f_1(x) = x_1^2 + x_2^2 + x_3^2$ $\qquad$ $f_2(x) = \sin(x\text{-}10) / (x\text{-}10)$

$f_3(x) = 100\,(x_1^2 - x_2^2)^2 + (1 - x_1)^2$

$$f_4(x) = \left[\frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j + \Sigma_{i=1} (x_i - a_{ij})^6} \right]^{-1}$$

$$f_5(x) = 0.5 + \frac{\sin^2(\sqrt{x^2+y^2}) - 0.5}{(1 + 0.001(x^2+y^2))^2}$$

where $a_{ij} = \begin{bmatrix} -32 & -16 & 0 & 16 & 32 & -32 & -16 & 0 & \ldots. & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & -16 & -16 & \ldots. & 32 & 32 \end{bmatrix}$

2.3.2. Performance Analysis

The results of the test runs on 30 observations are as follows:

Table 1 : Performance of Gradient Descent Method

Function	Result	Time
f_1	$8.9 \times 10^{-9} \pm 4 \times 10^{-10}$	0.11007 sec
f_2	-0.03376 ± 0.000002	0.17754 sec
f_3	0.000006 ± 0.000001	0.21411 sec
f_4	0.9989 ± 0.0001	0.19873 sec
f_5	0.00172 ± 0.00016	0.14318 sec

Table 2 : Performance of Original BPSO

Function	Result	Time
f_1	0.000081 ± 0.000038	0.29846 sec
f_2	-0.217226 ± 0.000003	0.21024 sec
f_3	0.000016 ± 0.000008	0.31156 sec
f_4	0.9986 ± 0.0000	0.38821 sec
f_5	0.000743 ± 0.000105	0.30102 sec

Table 3 : Performance of Modified BPSO

Function	Result	Time
f_1	0.000014 ± 0.000008	0.31007 sec
f_2	-0.217226 ± 0.000023	0.26176 sec
f_3	0.000076 ± 0.000018	0.35321 sec
f_4	1.03119 ± 0.07355	0.40887 sec
f_5	0.00172 ± 0.00016	0.41098 sec

Table 4 : Performance of Simulated Annealing

Function	Result	Time
f_1	$2.48 \times 10^{-7} \pm 2.3 \times 10^{-8}$	0.29976 sec
f_2	-0.217234 ± 0.0005	0.19802 sec
f_3	$5.73 \times 10^{-6} \pm 3.4 \times 10^{-7}$	0.28156 sec
f_4	0.9998 ± 0.0000	0.41034 sec
f_5	$3.97 \times 10^{-6} \pm 1.8 \times 10^{-7}$	0.35102 sec

The above table shows that Gradient Descent method is not reliable and fails to optimise functions with multiple critical points in larger solution space, whereas meta-heuristic algorithm has an escape method to discover better critical points.

CHAPTER 3

LOCALITY SENSITIVE HASHING-RANDOM PROJECTION

Locality-sensitive hashing (LSH) reduces the dimensionality of high-dimensional data. LSH hashes input items so that similar items map to the same "buckets" with high probability (the number of buckets being much smaller than the universe of possible input items). LSH differs from conventional and cryptographic hash functions because it aims to maximise the probability of a "collision" for similar items. Locality-sensitive hashing has much in common with data clustering and nearest neighbour search.
In information theory, the Hamming distance between two strings of equal length is the number of positions at which the corresponding symbols are different.

3.1. Algorithm

Let the sender has a 128 bit code (A) , and the receiver knows A belongs to *X* (sample space of all possible 128 bit codes that sender can have), where $o(X) << 2^{128}$. k randomly generated 128 bit vectors, let it be V_i (i = 1 to k), are already known to both sender and receiver. The sender sends sign(A.V_i) which is k bit code (k<128). Receiver computes the same operation and creates k bit hash codes for each elements of *X*. Finally searching the k bit hash code among the hash codes of the elements of *X*, determines which element of *X* is A.
In this algorithm, if b is any 128 bit code, sign(b.V_i) tells on which side of the vector V_i does b lie (projection) . So the vectors V_i divide the solution space into 2^k buckets with well defined addresses. So the probability of collision is $o(X)/2^k$.

3.2. Problem statement

Initially the sender sent the code A to the receiver. Now the sender changes at most m bits in the code A to get A'. Rather than sending all 128 bits of the code A', this algorithm can be used to send a lesser number of bits, yet creating an accurate guess of the changed code A'.

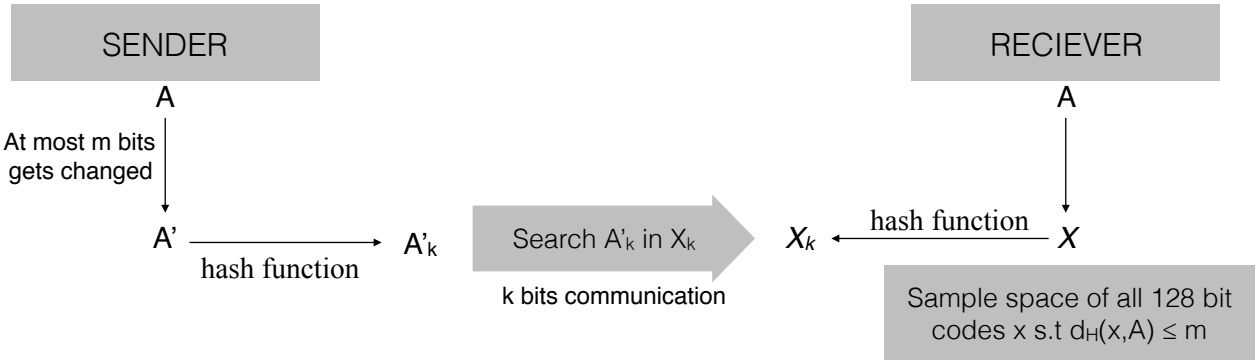


Figure 2 : Illustration of communication

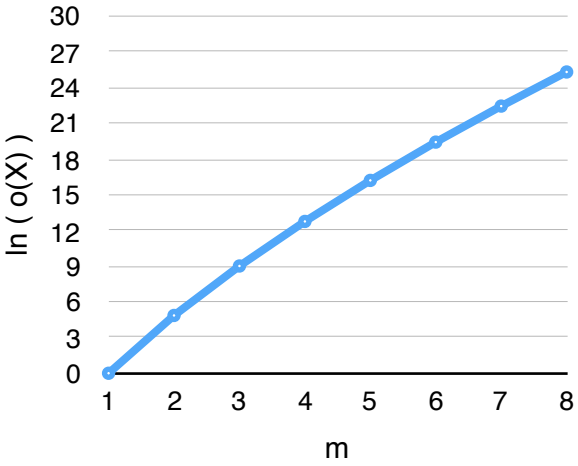
$$o(X) = \sum_{r=1}^{m} {}^{128}C_r$$

3.3. Sample space

The size of the sample space X increases exponentially. $o(X) = \sum_{r=1}^{m} {}^{128}C_r$

Table 5 : Variation of o(X) with respect to m

m	o(X)
0	1
1	129
2	8,257
3	349,633
4	11,017,633
5	275,584,033
6	5,699,195,233



Here X consists of all possible 128 bit codes that has an hamming distance less than m, from the initial code A.

3.4. Analysis of probability of collision

The more the number of bits in the hash code lesser is the probability of collision. The probability of collision is given by :

$$\text{Probability} = \frac{o(X)}{2^k} = \frac{\sum_{r=1}^{m} {}^{128}C_r}{2^k}$$

Throughout the experiment, the probability of collision was maintained to be 0.001. So the size of hash code depends on m, and the probability of collision = 0.001.

Table 6 : Requirement of k with respect to m at P = 0.001

m	k
0	12
1	19
2	25
3	31
4	36
5	41
6	45

$$k = \text{CEIL} (\text{LOG2} (o(X) / P)) + \delta$$

3.5. Results

The below result is derived from 1000 simulated run (100 when m=4 due to time complexity), where Success % depicts the percentage of total run in which the code gave the correct guess with no collision of solution. If a result gives more than 1, 128 bit code as the correct result (happens when the hash code of correct code collides with another code), then the run is considered as fail. Time refers to the average time required in each run.

In this experiment, two hash functions were tested : 1. XOR operation 2. Random Projection

XOR operation is the trivial technique in which a 128 bit code x is bitwise AND operated with k random 128 bit codes, to get k 128 bit codes. These k bits are individually XORed to get a 1 bit code, which as a whole converts x into a k bit code.

Table 7 : Performance report of the LSH (XOR operation) experiment

m	k	Time	Success %
0	12	-	-
1	19	0.000212 sec	96.2
2	25	0.01818 sec	88.8
3	31	1.78630 sec	76.7
4	36	108.95612 sec	72
5	41	4745.57837 sec	100 (only 1 try)
6	45	-	-

Table 8 : Performance report of the LSH (random projection) experiment

m	k	Time	Success %
0	12	-	-
1	19	0.000331 sec	100
2	25	0.02694 sec	95.1
3	31	2.24063 sec	94.4
4	36	121.85622 sec	93
5	41	6874.28971 sec	100 (only 1 try)
6	45	-	-

CHAPTER 4

CONCLUSION

Successful application of random projection LSH was implemented to the problem statement and compared with XOR LSH, which clearly shows that effectiveness of hashing using random projection is better than XOR hashing.

The main problem and scope of improvement of this code is time complexity when m increases. An optimised search algorithm can be applied rather than the current exhaustive search process. Challenge lies on how to neglect most of the observations from X, depending on the results found from previous hash comparisons. The idea of approach is that the search space will expand one by one, depending on the requirement and the previous knowledge of already compared hash codes.

APPENDIX

%% MATLAB CODE FOR RANDOM PROJECTION LSH %%

```
clc;
tic
global dimension;
dimension=128;
change=3;

yinit=randi([0,1],1,dimension);
xinit=yinit;

noHF = 29;

c0=1;
c1=nchoosek(dimension,1);
c2=nchoosek(dimension,2);
c3=nchoosek(dimension,3);

X=[];
X=[X;zeros(c0,3)];
X=[X;combnk(1:dimension,1),zeros(c1,2)];
X=[X;combnk(1:dimension,2),zeros(c2,1)];
X=[X;combnk(1:dimension,3)];

noDP=c0+c1+c2+c3;

avone=(1+nchoosek(dimension-1,1)+nchoosek(dimension-1,2)+nchoosek(di
mension-1,3))...
    /(1+nchoosek(dimension,1)+nchoosek(dimension,
2)+nchoosek(dimension,3));
avzero=1-avone;

avpoint=zeros(1,dimension);
for i=1:dimension
    if(xinit(i)==1)
        avpoint(i)=avone;
    else
        avpoint(i)=avzero;
    end
end

random=ceil(rand*noDP);
changedy=X(random,:);

A=randi([0,1],noHF,dimension)-.5;

ref=zeros(1,noHF);

for j=1:noHF
    for k=1:dimension
        ref(j)=ref(j)+(xinit(k)-avpoint(k))*A(j,k);
    end
end
dot=repmat(ref,noDP,1);
```

```
querydot=ref;
for i=1:noDP
    for j=1:noHF
        for k=1:3
            if(X(i,k)~=0)
                if(xinit(X(i,k))==0)
                    dot(i,j)=dot(i,j)+A(j,(X(i,k)));
                else
                    dot(i,j)=dot(i,j)-A(j,(X(i,k)));
                end
            end
        end
    end
end
for i=1:noDP
    for j=1:noHF
        if(dot(i,j)>=0)
            dot(i,j)=1;
        else
            dot(i,j)=0;
        end
    end
end

for j=1:noHF
    for k=1:3
        if(changedy(k)~=0)
            if(xinit(changedy(k))==0)
                querydot(j)=querydot(j)+A(j,(changedy(k)));
            else
                querydot(j)=querydot(j)-A(j,(changedy(k)));
            end
        end
    end
end
for j=1:noHF
    if(querydot(j)>=0)
        querydot(j)=1;
    else
        querydot(j)=0;
    end
end


Dist=zeros(noDP,1);
for i=1:noDP
    distance=0;
    for j=1:noHF
        if (querydot(j)~=dot(i,j))
            distance=distance+1;
        end
    end
    Dist(i)=distance;
end

X=[X Dist];
X=sortrows(X,4);
```

```
for i=1:noDP
    if(X(i,4)~=0)
        guesses=i-1;
        break;
    end
end
fprintf('\nCode gives best %d guesses by making %d buckets, and
their distance from querycode are :'...
    ,guesses,max(Dist));

topguesses=[];
for i=1:guesses
    add=0;
    for j=1:3
        if(X(i,j)~=changedy(j))
            if(changedy(j)==0||X(i,j)==0)
                add=add+1;
            else
                add=add+2;
            end
        end
    end
    topguesses=[topguesses;add];
end
topguesses

toc
```

REFERENCE

[1] Approximate nearest neighbor search: binary codes and vector quantization, Yannis Avrithis, University of Athens, 2015

[2] A. Andoni and P. Indyk., Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. Comm.ACM, 51:117–122, 2008.

[3] Locality Sensitive hashing and Beyond, Microsoft Research : https://www.youtube.com/watch?v=t_8SpFV0l7A.

[4] A Novel Probability Binary Particle Swarm Optimization Algorithm and Its Application, Ling Wang, 2008.

[5] Polar Codes for Non asymmetric Slepian–Wolf Coding, Saygun Önay, 2012.

[6] A Tutorial on Principal Component Analysis, Jonathon Shlens, 2005.

[7] Particle Swarm Optimisation, James Kennedy and Russell Eberhart, PurdueSchool of Engineering and Technology, IEEE, 1995

[8] EE5585 Data Compression : Lecture 27, Arya Mazumdar, Fangying Zhang, 2013

[9] Probability-based Binary Particle Swarm Optimization Algorithm and Its Application to WFGD Control, LanLan Zhen, Ling Wang , 2008, International Conference on Computer Science and Software Engineering

[10] NoiselessCoding of Correlated Information Sources, David Slepian, Jack Wolf, IEEE 1973