信州大学工学部

学士論文

離散無記憶通信路に対するコストあたり通信路容量の算出アルゴリズムの比較に向けて

指導教員　西新 幹彦 准教授

学科　電子情報システム工学科
学籍番号　16T2803A
氏名　伊坪 快斗

2021 年 03 月 02 日

目次

1　はじめに

通信システムとは，符号シンボルを送受信することによって情報を伝えるシステムである．一般に符号シンボルの送信にはコストがかかる．具体例として，電力や時間などがコストとされる．

通信システムによっては，シンボルのコストは同一である場合もあるが，一般的にはシンボルごとによって異なる．例えば，AWGN 通信では，入力シンボルは任意の実数値であるが，多くの場合はその 2 乗が入力シンボルのコストとみなされる [1]．パケット間隔を用いた通信では，送信間隔がコストとみなされる [2].

情報理論における通信路符号化の問題では，通信路容量を求めることが基本的な問題となる．コストの概念のない通常の通信路符号化では，通信路容量は符号シンボルあたりの情報量という次元で定義される．一方，コスト付き通信路符号化問題と呼ばれる符号シンボルにコストが定義されているような問題では，2 通りの主要な問題設定が存在する．ひとつはコストに関する制約を満たしたうえで通常と同じ通信路容量を求める問題で，これはコスト制約付き通信路容量と呼ばれる．もうひとつはコストあたりの情報量という次元で定義されるコストあたり通信路容量を求める問題である．

これに関して，Gallager[3] は，コスト制約付き通信路容量を制約付き最適化問題の最適解として明らかにし，Verdú[1] は，この結果を用いて，コストあたり通信路容量を制約付き最適化問題の最適解として明らかにしている．

コストあたり通信路容量の算出アルゴリズムを，Blahut が出している．そして，川合西新 [4] がより効率よくコストあたり通信路容量を算出するアルゴリズムを提案している．

本論文では，定常無記憶通信路に対して，2 つのコストあたり通信路容量を数値的に計算するアルゴリズムの比較を行うために，実装を行った．

2　コスト付き通信路符号化問題

本研究に至る主要な背景として，コスト制約付き通信路容量およびコストあたり通信路容量に関する定理を以下に述べる．

2.1　コスト制約付き通信路容量

$\mathcal{X}$ を入力アルファベット，$\mathcal{Y}$ を出力アルファベットとして，通信路の遷移確率と呼ぶ条件付き確率の組 $W = \{W_{(y|x)}\}_{(x,y)\in\mathcal{X}\times\mathcal{Y}}$ を条件

$$\sum_{y} W_{(y|x)} = 1, x \in \mathcal{X} \tag{1}$$

を満たすように任意に定める．入出力アルファベット $\mathcal{X}$, $\mathcal{Y}$ はともに有限であるとする．

入力系列

$$\boldsymbol{x} = x_1 x_2 \cdots x_n \in \mathcal{X}^n \tag{2}$$

が与えられたとき，出力系列

$$\boldsymbol{y} = y_1 y_2 \cdots y_n \in \mathcal{Y}^n \tag{3}$$

の出現する条件付き確率 $W^n_{(\boldsymbol{y}|\boldsymbol{x})}$ が

$$W^n_{(y|x)} = \prod_{i=1}^{n} W_{(y_i|x_i)} \tag{4}$$

で与えられる通信路を定常無記憶通信路と呼び，W と表す．このような通信路を n 回使用することによって次のように情報を伝えることを考える．すなわち，伝送すべきメッセージの集合を $\mathcal{M}_n = \{1, 2, \cdots, M_n\}$ としたとき，それらを長さ n の通信路入力に変換する写像 $\varphi_n : \mathcal{M}_n \to \mathcal{X}^n$ を符号器と呼ぶ．ここで，$\boldsymbol{u}_i = \varphi_n(i)$ をメッセージ i の符号語と呼び，$\mathcal{C}_n = \{\boldsymbol{u}_1, \boldsymbol{u}_2, \cdots, \boldsymbol{u}_{M_n}\}$ を符号と呼ぶ．符号器 φ_n はメッセージ i を送りたいとき，その符号器 u_i を通信路に入力する．一方，通信路の受信側では，あらかじめ定めておいた $\mathcal{Y}^n$ の排反な分割

$$\mathcal{Y}^n = \mathcal{D}_1 \cup \mathcal{D}_2 \cup \cdots \cup \mathcal{D}_{M_n}, \mathcal{D}_i \cap \mathcal{D}_j = \emptyset, i \neq j \tag{5}$$

に基づいて，受けとった出力 y が $y \in \mathcal{D}_i$ ならば符号器に入力されたメッセージは $i \in \mathcal{M}_n$ であったと推定する ($\mathcal{D}_i$ をメッセージ i の復号領域と呼ぶ)．このような操作を復号といい，これを表す写像 $\varphi_n : \mathcal{Y}^n \to \mathcal{M}_n$ を復号器という．また，通信路符号化における復号誤り確率 ε_n を

$$\varepsilon_n \triangleq \frac{1}{M_n} \sum_{i \in \mathcal{M}_n} W^n_{\mathcal{D}_i^c u_i} \tag{6}$$

と定める．ここで，ε_n はメッセージ $i \in \mathcal{M}_n$ が等確率で発生すると仮定したときの平均誤り確率であることに注意する．通信路符号化の基本的な問題では，誤り確率が $\varepsilon_n \to 0$ $(n \to \infty)$ となるような符号器と復号器の組 (φ_n, ψ_n) が存在するという条件のもとで符号化レートを最大限どこまで大きくできるかを考える．

通信路符号化問題のある状況では，符号語のコストを考えなければならないことがある．一般に，$n = 1, 2, \cdots$ に対して，写像 $c_n : \mathcal{X}^n \to \mathbb{R}$ を任意に定め，

$$\boldsymbol{c} = \{c_n\}_{n=1}^{\infty} \tag{7}$$

とおく．ただし，$\mathbb{R}$ は実数全体の集合を表す．このとき，$\boldsymbol{x} \in \mathcal{X}_n$ に対して $c_n(\boldsymbol{x})$ を $\boldsymbol{x}$ のコストと呼ぶ．本論文では，ある $c : \mathcal{X} \to \mathbb{R}$ を用いて

$$c_n(\boldsymbol{x}) = \sum_{i=1}^{n} c(x_i), \boldsymbol{x} = x_1 x_2 \cdots x_n \tag{8}$$

と書ける特別な場合を考える．このようなコストは加法的コストと呼ばれる．以上のもとで，コスト制約付き通信路容量は次のように定義される．

定義 1　レート R が Γ -達成可能とは，ある固定長符号の列 $\{\varphi_n\}_{n=1}^{\infty}$ が存在して

$$\frac{1}{n}c_n(\varphi_n(i)) \leq \Gamma, i = 1, 2, \ldots, M_n \tag{9}$$

$$\liminf_{n\to\infty} \frac{1}{n} \log M_n \geq R \tag{10}$$

$$\lim_{n\to\infty} \varepsilon_n = 0 \tag{11}$$

を満たすことである．

定義 2　コスト制約付き通信路容量を

$$C(\Gamma) \triangleq \sup\{R| \text{ レート } R \text{ がΓ-達成可能 }\} \tag{12}$$

と定義する．

通信路容量 $C(\Gamma)$ は数学的に表すことができる．そのための準備として次のように相互情報量を導入する．入力分布 $p = \{p_x\}_{x\in\mathcal{X}}$ を

$$\sum_x p_x = 1 \tag{13}$$

を満たすように任意に定める．$q = \{q_y\}_{y\in\mathcal{Y}}$ は

$$q = \sum_x p_x W_{yx} \tag{14}$$

を満たし，p を入力分布とした時の通信路 W の出力分布を示す．

確率変数 X, Y に対し $I(X;Y)$ を

$$I(X;Y) \triangleq \sum_{x,y} p_x W_{yx} \log \frac{W_{yx}}{q_y} \tag{15}$$

と定義し，X と Y の間の相互情報量と呼ぶ．

Gallager[3] はコスト制約付き通信路容量に関して次の定理を明らかにしている．

定理 1　(Gallager[3]) 定常無記憶通信路 W のコスト制約付き通信路容量 $C(\Gamma)$ は

$$C(\Gamma) = \sup_{X:E[c(X)]\leq\Gamma} I(X;Y) \tag{16}$$

で与えられる．ここで，Y は X を入力変数としたときの通信路 W の出力変数を表す．

2.2 コストあたり通信路容量

通信路の入力アルファベット上にコストが定義されたもとで，コストあたり通信路容量は以下のように定義される．

定義 3 コストあたりレート R が達成可能とは，ある固定長符号の列 $\{\varphi_n\}_n{}^{\infty}$ が存在して

$$\liminf_{n\to\infty} \frac{1}{\max_i c_n(\varphi_n(i))} \log M_n \geq R \tag{17}$$

$$\lim_{n\to\infty} \varepsilon_n = 0 \tag{18}$$

を満たすことである．

定義 4 コストあたり通信路容量を

$$C \triangleq \sup\{R| \text{ コストあたりレート } R \text{ が達成可能 }\} \tag{19}$$

と定義する．

Verdú[1] は上記の Gallager[3] による結果を用いて，コストあたり通信路容量に関して以下の定理を明らかにしている．

定理 2 (Verdú[1]) 定常無記憶通信路 W のコストあたり通信路容量 C は，

$$C = \sup_{\Gamma>0} \frac{C(\Gamma)}{\Gamma} = \sup_X \frac{I(X;Y)}{E[c(X)]} \tag{20}$$

で与えられる．

ここで，$\frac{I(X;Y)}{E[c(X)]}$ をコストあたり相互情報量と呼ぶ．

3 コストあたり通信路容量を算出するアルゴリズム

式 (20) からコストあたり通信路容量を数値的に計算するために，通信路容量やレート歪み関数をを数値的に算出するアルゴリズムである, Blahut 提案のアルゴリズムを紹介した後，川合西新提案のアルゴリズムを紹介する．また，実装した際の実装方法についても説明する．

3.1 Blahut アルゴリズム

コストあたり通信路容量

$$C = \sup_{\Gamma>0} \frac{C(\Gamma)}{\Gamma} \tag{21}$$

を計算するアルゴリズムである Blahut アルゴリズムを以下に示す.

1. 0 から 1.0 の範囲でパラメーター s の値を定める.
2. 初期入力分布 $p^{(1)} = \{p_x^{(1)}\}$ を内点にとる一様分布

$$p_x = \frac{1}{|X|} \tag{22}$$

とする.

3. c_x を求める．それは

$$q_x = exp(\sum_y W_{yx} \log \frac{W_{yx}}{\sum_x p_x W_{yx}} - sc(x)) \tag{23}$$

とする.

4. 誤差を求める式を

$$I_L = \log(\sum_x p_x q_x) \tag{24}$$

$$I_U = \log(\max_x q_x) \tag{25}$$

とする.

5. 式 (24)(25) を引き、差が誤差以下にならなければ

$$p_x = p_x \frac{q_x}{\sum_x p_x q_x} \tag{26}$$

として、2. に戻る．誤差以下であれば

$$E = \sum_x p_x c(x) \tag{27}$$

として、

$$C(E) = sE + I_L \tag{28}$$

とする.

パラメーター s を変化させ，以上より得られる，$C(E)$ および E を用い，$\frac{C(E)}{E}$ を最大化することで，コストあたり通信路容量を得られる.

本研究で，実装した際には，パラメータ s を，0 から 1 まで 0.001 ずつ変化させた．最大化の方法としては，パラメータを変化させていく際に，一つずつ値を比較して最大化した．式 (24)(25) の差で誤差をとるところでは，誤差は 0.000001 以下となるようにした.

3.2 川合西新アルゴリズム

コストあたり通信路容量

$$C = \sup_X \frac{I(X;Y)}{E[c(X)]} \tag{29}$$

を数値的に計算するアルゴリズムを以下に紹介する．コストあたり相互情報量を拡張した概念として，次の関数

$$I(\boldsymbol{p};\boldsymbol{\varphi}) = \frac{1}{E[c(X)]}(H(X) + \sum_x p_x \sum_y W_{yx} \log \varphi_{xy}) \tag{30}$$

を定義する．

1. 初期入力分布 $\boldsymbol{p}^{(1)} = \{p_x^{(1)}\}$ を内点にとる．一様分布

$$p_x^{(1)} = \frac{1}{|X|} \tag{31}$$

としてよい．

2. $N = 1$ とする．
3. $\sup_{\boldsymbol{\varphi}} I(\boldsymbol{p}^{(N)};\boldsymbol{\varphi})$ を達成する $\boldsymbol{\varphi}$ を求める．具体的には，これを $\boldsymbol{\varphi}^{(N)} = \{\varphi_{xy}^{(N)}\}$ とし，

$$\varphi_{xy}^{(N)} = \frac{p_x^{(N)} W_{yx}}{q_y^{(N)}} \tag{32}$$

とする．

4. $\sup_{\boldsymbol{p}} I(\boldsymbol{p};\boldsymbol{\varphi}^{(N)})$ を達成する $\boldsymbol{p}$ を求める．具体的には，これを $\boldsymbol{p}^{(N+1)} = \{p_x^{(N+1)}\}$ とし，

$$p_x^{(N+1)} = \exp(\sum_y W_{yx} \log \varphi_{xy}^{(N)} - \alpha^{(N)} c(x)) \tag{33}$$

とする．ただし，

$$\alpha^{(N)} : \sum_x \exp(\sum_y W_{yx} \log \varphi_{xy}^{(N)} - \alpha^{(N)} c(x)) = 1 \tag{34}$$

とする．

5. N を 1 増やして 3. に戻る．

以上より得られる

$$\sup_{\boldsymbol{\varphi}} I(\boldsymbol{p}^{(N)};\boldsymbol{\varphi}) = I(\boldsymbol{p}^{(N)};\boldsymbol{\varphi}^{(N)}) \tag{35}$$

および

$$\sup_{\boldsymbol{p}} I(\boldsymbol{p};\boldsymbol{\varphi}^{(N)}) = I(\boldsymbol{p}^{(N+1)};\boldsymbol{\varphi}^{(N)}) \tag{36}$$

はコストあたり通信路容量 C に収束する値となる．

したがって，ループの回数 N を十分大きくすることによって誤差の少ない値が得られる．

式 (34) では，方程式の解を求めなければならない．本研究ではこれをニュートン法で実装した．しかし，ニュートン法でこの方程式を解いた際に，収束までに時間がかってしまったため，収束を早めるために仮処置として，漸化式を $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$ から，$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \times 0.5$ としたまた，ループの回数は，経験的に小数点以下 6 桁が誤差なく得られる 100 回とした．

4　比較と考察

前章で説明したように Blahut 提案のアルゴリズムと川合西新提案のアルゴリズムを実装した．比較するに当たって、二元消失通信路と，二元対称通信路の 2 つの通信路で，消失確率と反転確率をそれぞれ 0.2，コストを 2,4 とし，実行した．実行結果を表 1 に示す．実行結果からは，川合西新提案のアルゴリズムの方が高速のように見える．しかし，2 つのアルゴリズムは，最適化されておらず，公平に比較はできていない．最適化されていない点として，Blahut 提案のアルゴリズムでは，$\frac{C(E)}{E}$ の最大値のとり方が，パラメータ s を，0 から 1 まで 0.001 ずつ変化させ，一つずつ値を比較している点が上げられる．川合西新提案のアルゴリズムでは，N 回ループを抜ける条件が，100 回としてる点や，ニュートン法で，収束までの時間が長すぎるため，仮処置を行っているという点があげられる．それぞれの解決案として，Blahut 提案のアルゴリズムでは，パラメータ s の変化量と最大値は，比較対象のパラメーターと合わせて調整を行い，最大化問題は，効率のよいアルゴリズムを模索する必要がある．川合西新提案のアルゴリズムは，N 回ループを比較対象のアルゴリズムとあわせて調整し，ニュートン法については，他の方程式を解くアルゴリズムを使用する必要がある．

5　まとめ

本論文では．コストあたり通信路容量を数値的に計算するアルゴリズムを川合西新提案のアルゴリズムと Blahut 提案のアルゴリズムの 2 通りで実装し比較した．結果としては，川合西新提案のアルゴリズムのほうが高速な結果となったが，本論文内で扱った川合西新提案のアルゴリズムは N 回ループを行っていたり，Blahut 提案のアルゴリズムは最大値を求める際に大

表 1　実行結果

通信路	Blahut の実行時間 [sec]	川合西新の実行時間 [sec]
二元消失通信路	7.191	1.652
二元対称通信路	289.772	2.151

きいをとっていたりするため，パラメーター次第では結果が大きく異なる可能性がある．これについては，今後の課題である．

謝辞

本研究を進めるにあたり，丁寧なご指導を頂いた西新幹彦准教授に感謝の意を表する．

参考文献

[1] Sergio Verdú, “On Channel Capacity per Unit Cost,” IEEE Transactions on Information Theory, vol.36, no.5, pp.1019–1030, Sep. 1990

[2] Venkat Anantharam, Sergio Verdú, “Bits Through Queues,” IEEE Transactions on Information Theory,vol.42, no.1, pp.4–18, Jan. 1996.

[3] Robert G. Gallager, Information Theory and Reliable Communication, John Wiley & Sons, 1968

[4] 川合康太，西新幹彦「離散無記憶通信路に対するコストあたり通信路容量の算出に関する考察」，The 42nd Symposium on Information Theory and its Applications (SITA2019)，no2.2.3，pp.117–122, 2019 年 11 月.

付録 A　ソースコード

A.1　Blahut 提案のアルゴリズムのプログラム

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h> //引数 -lm 忘れないようにね
#include <time.h>

#define LOOPMAX 10000
#define INSIZE 2
#define OUTSIZE 2
#define E 0.2
#define ERROR 0.000001
#define SMAX 1.0
#define SADD 0.001

FILE *fp;
FILE *fp2;

double arimoto(double w[OUTSIZE][INSIZE], double s, double* rCG, double* rG);

int main(){
    //printf("hello\n");

    char *fname = "CG_G.csv";
    char *fname2 = "G_CGG.csv";

    double w[OUTSIZE][INSIZE] = {{E, 1.0 - E}, {1.0 - E, E}};
    double ans = 0.0;
    double rG = 0.0;
    double rCG = 0.0;
    double maxMutal = 0.0;
    double tmpMutal = 0.0;
    double s = 0.0;
    double maxS = 1000.0;

    ans = log(2.0)*(1 - E);

    fp = fopen(fname, "w");
    if(fp == NULL){
        printf("%s ファイルが開けません￥n", fname);
        return -1;
    }
    fp2 = fopen(fname2, "w");
      printf("%s ファイルが開けません￥n", fname);
        return -1;
    }   if(fp2 == NULL){


    for (double i = 0.0; i < SMAX; i = i + SADD){
        tmpMutal = arimoto(w, i, &rCG, &rG);
        //fprintf(fp,"%lf,%lf,%lf\n",rCG,rG,i);
        //fprintf(fp2,"%lf,%lf\n",rG,rCG/rG);
        if (tmpMutal > maxMutal){
            maxMutal = tmpMutal;
            maxS = i;
        }
    }

    printf("maxMutal = %lf\n", maxMutal);
    printf("maxS = %lf\n", maxS);

    s = 0.0;
    maxS = 100.0;
```

```
    clock_t start,end;
    start = clock();
    for (int i = 0; i < LOOPMAX; i++) {
        maxMutal = 0.0;
        tmpMutal = 0.0;
        s = 0.0;
        maxS = 100.0;


        // arimoto(w, maxS, &rCG, &rG);

        for (double j = 0.0; j < SMAX; j = j + SADD){
            tmpMutal = arimoto(w, j, &rCG, &rG);
            if (tmpMutal > maxMutal){
                maxMutal = tmpMutal;
                maxS = j;
            }
        }
    }

    end = clock();
    printf("time = %lf\n",(double)(end-start)/CLOCKS_PER_SEC);

    printf("maxMutal = %lf\n", maxMutal);
    printf("maxS = %lf\n", maxS);
    printf("CG = %lf\n", rCG);
    printf("G = %lf\n", rG);
    printf("%lf\n", ans);
    //27
    fclose(fp);
    fclose(fp2);

}

double arimoto(double w[OUTSIZE][INSIZE],double s, double* rCG, double* rG){
    double p[INSIZE];

    double q[OUTSIZE];
    double phi[INSIZE][OUTSIZE];

    double alpha = 0.277259;
    double alpha0;
    double sum = 0.0;

    double mutal = 0.0;
    double cost;
    double tmp = 0;

    double num = 0;
    double den = 0;
    double a[INSIZE];

    double c[INSIZE] = {2,4};//ここがコスト

    double cx;
    double cTmp[INSIZE] = {0,0};
    double IL = 100.0;
    double IU = 0.0;
    double G = 0.0;
    double CG = 0.0;

    double oldp[INSIZE];
    double max = 0.0;

    for (int i = 0; i < INSIZE; i++){
        p[i] = 1.0 / INSIZE;
    }
    //2
```

```
    while( fabs(IL - IU) > 0.000001){
        for (int y = 0; y < OUTSIZE; y++){
            q[y] = 0.0;
            for (int x = 0; x < INSIZE; x++){
                q[y] += p[x] * w[y][x];
            }
        }//式 13 のちょっと後

        for (int x = 0; x < INSIZE; x++){
            cx = 0.0;
            for (int y = 0; y < OUTSIZE; y++){
                if (w[y][x] != 0.0){
                    cx += (w[y][x] * log(w[y][x] / q[y]));
                }
                //printf("aiue\n");
            }
            cTmp[x] = exp(cx - s * c[x]);
        }
        //3

        IL = 0.0;
        max = 0.0;
        for (int x = 0; x < INSIZE; x++){
            IL += p[x] * cTmp[x];
            if (max < cTmp[x]) {
                max = cTmp[x];
            }
        }
        IL = log(IL);
        IU = log(max);
        //式 4
        //式 5



        for (int x = 0; x < INSIZE; x++){
            oldp[x] = p[x];
        }

        for (int x = 0; x < INSIZE; x++){
            sum = 0;
            for (int i = 0; i <INSIZE; i++){
                sum += oldp[i] * cTmp[i];
            }
            p[x] = oldp[x] * cTmp[x] / sum;
        }
    }

    for (int x = 0; x < INSIZE; x++){
        G += p[x] * c[x];
        fprintf(fp, "%lf,", p[x]);
    }

    CG = s * G + IL;

    //printf("%f\n",s);

    mutal = CG / G;
    *rCG = CG;
    *rG = G;
    return mutal;

}
```

A.2 川合西新提案のアルゴリズムのプログラム

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h> //引数 -lm 忘れないようにね
#include <time.h>

#define LOOPMAX 100
#define LOOPMAX2 10000
#define INSIZE 2
#define OUTSIZE 2
#define E 0.2
#define ERROR 0.00001

FILE *fp;

double arimoto(double w[OUTSIZE][INSIZE]);

int main(){
    //printf("hello\n");

    char *fname = "a.csv";

    double w[OUTSIZE][INSIZE] = {{E, 1.0 - E}, {1.0 - E, E}};
    double ans = 0.0;

    fp = fopen(fname, "w");
    if(fp == NULL){
        printf("%s ファイルが開けません¥n", fname);
        return -1;
    }

    ans = log(2.0)*(1 - E);
    //c = log(2.0)-((1.0-E)*log(1.0/(1.0-E)) + E*log(1.0/E));

    printf("%lf\n", arimoto(w));

    clock_t start,end;
    start = clock();
    for (int i = 0; i < LOOPMAX2; i++) {
        arimoto(w);
    }
    end = clock();
    printf("time = %lf\n",(double)(end-start)/CLOCKS_PER_SEC);
    fclose(fp);
    printf("%lf\n", ans);
    //27
}

double arimoto(double w[OUTSIZE][INSIZE]){
    double p[INSIZE];

    double q[OUTSIZE];
    double phi[INSIZE][OUTSIZE];

    double alpha = 0.277259;
    double alpha0;
    double sum = 0.0;

    double mutal = 0.0;
    double cost;
    double tmp = 0;

    double num = 0;
    double den = 0;
```

```
    double a[INSIZE];

    double c[INSIZE] = {2,4};

    for (int i = 0; i < INSIZE; i++){
        p[i] = 1.0 / INSIZE;
    }
    //36

    for (int n = 0; n < LOOPMAX; n++){

        for (int y = 0; y < OUTSIZE; y++){
            q[y] = 0.0;
            for (int x = 0; x < INSIZE; x++){
                q[y] += p[x] * w[y][x];
            }
        }//式 13 のちょっと後

        for (int y = 0; y < OUTSIZE; y++){
            for (int x = 0; x < INSIZE; x++){
                phi[x][y] = p[x] * w[y][x] / q[y];
            }
        }//37

        sum = 0.0;
        do{
            num = 0;
            den = 0;
            alpha0 = alpha;
            for (int x = 0; x < INSIZE; x++){
                a[x] = 0;
                for (int y = 0; y < OUTSIZE; y++){
                    if (phi[x][y] != 0 ){
                        a[x] += w[y][x] * log(phi[x][y]) - alpha0 * c[x];
                    }
                }
                num += exp(a[x]);
                den += exp(a[x]) * -c[x];
            }
            alpha = alpha0 - ((num - 1) / den )* 0.5;//仮処置
            //fprintf(fp,"%lf,%d\n", alpha, n);
            //printf("%lf %d\n", alpha, n);
        } while (fabs((alpha - alpha0) / alpha) > ERROR);
        for (int x = 0; x < INSIZE; x++){
            p[x] = exp(a[x]);
        }//38
    }

    cost = 0.0;
    for (int x = 0; x < INSIZE; x++){
        tmp = -log(p[x]);
        for (int y = 0; y < OUTSIZE; y++){
            if (phi[x][y] != 0 ){
                tmp += w[y][x] * log(phi[x][y]);
            }
        }
        mutal += p[x] * tmp;

        cost += p[x] * c[x];

        //printf("%lf\n", p[x]);
    }

    mutal = mutal / cost;
    //35
    return mutal;
}
```