信州大学

大学院理工学系研究科

修士論文

# 状態数より少ない符号器を用いた情報源符号化について

指導教員　西新 幹彦 准教授

専攻　電気電子工学専攻
学籍番号　14TM247E
氏名　LIU DAOBIN

2016 年 2 月 17 日

# 目次

# 1　研究の背景と目的

現在の社会は，スマートデバイス製品に対する使用率の上昇及び無線データ通信の使用者数の増加に伴い，無線データ通信サービスが凄まじい勢いで普及している．使用者は，その使用範囲を伝統的な文字や画像など小範囲に限定せず，4K テレビに代表されるような高画質の映像など膨大なデータの通信サービスを追い求め続けている．従って，現代の情報社会において扱うテータ量は膨大なものとなっており，未来においては，デジタルが「より多く」「より広く」「より速く」「より俊敏」に動くことのできる通信環境が整備され続けているので，情報を効率よく符号化し通信時間，通信容量を抑える情報圧縮技術は必要不可欠なものである．

このデータ圧縮の手法は大きく 2 つに分類される．1 つは，情報源の確率分布を推定する方法であり，もう 1 つはこれを行わない方法である．確率分布を推定する場合，推定された分布を符号化確率として用い，これが正確であれば符号化効率は良くなる．一方，大抵の情報源は内部に複数の状態をもっていて，状態ごとにシンボルの出現確率が異なる．したがって状態ごとに正確な符号化確率を用いる必要がある．状態数が多ければ符号化確率もたくさん用意しなくてはならない．

一般に，符号化に用いることのできるメモリは限りがあるので，状態数が非常に多いと，それらを区別することができなくなる．そこで，限られたメモリで効率の良い符号化を行うためには，複数の状態を 1 つの符号化確率で代表させることが考えられる．すると，必然的に正確でない符号化確率を用いることになる．したがって，このとき，似たような状態を集めることによって，符号化効率の悪化を最小にとどめることが問題となる．本研究では，限られた数の符号化確率を用いて符号化効率を最適にするための符号化確率の選び方について実験的に検討し，準最適な解を高速に導くアルゴリズムを提案する．

# 2　冗長度とダイバージェンス

情報を通信する際は，情報源から出力されたデータを転送する前に，データを小さくすることを目的とした符号化を行うと効率がよくなる．このような符号化をデータ圧縮という．

本研究では符号化確率を用いたデータ圧縮を考える．一つの符号化確率は一つの符号器とみなすことができる．情報源アルファベットを $\mathcal{X}$ とする．情報源の確率が $P$ であるとき，符号化確率として $P$ を用いることが最適である．このとき，シンボル $x \in \mathcal{X}$ に対し，符号語長は $-\log P(x)$ とみなされる．すると平均符号語長 $L^*$ は

$$L^* = \mathrm{E}[-\log P(X)] = -\sum_{x \in \mathcal{X}} P(x) \log P(x) = H(P)$$

すなわち分布 $P$ のエントロピーとなる．

符号化確率が一般に $Q$ の場合，シンボル $x \in \mathcal{X}$ に対する符号語長は $-\log Q(x)$ とみなされ，平均符号語長は

$$\begin{aligned} L &= \mathrm{E}[-\log Q(X)] \\ &= -\sum_{x\in\mathcal{X}} P(x)\log Q(x) \\ &= -\sum_{x\in\mathcal{X}} P(x)\log P(x) + \sum_{x\in\mathcal{X}} P(x)\log\frac{P(x)}{Q(x)} \\ &= H(P) + D(P\|Q) \end{aligned}$$

となる．ここに，$D\,(P\|Q) \triangleq \sum_{x\in\mathcal{X}} P(x)\log\frac{P(x)}{Q(x)}$ $(0\log\frac{0}{0}=0$ と定義される.$)$ は $P$ と $Q$ の間のダイバージェンスと呼ばれる．符号語長の最適な値からの増加分は冗長度と呼ばれるが，

$$L - L^* = D(P\|Q)$$

となることから，平均冗長度はダイバージェンスによって測ることができる．

本研究では，符号化確率を用いてデータ圧縮をおこなうので，複数の状態を 1 つの符号器で符号化すると，平均冗長度は必ず正の値となる．

# 3 基礎的検証

本研究の基礎的検証として，非可算無限個の情報源が一様に分布する情報源空間を考え，そこに有限個の符号器を置いて符号化するときの最適な符号器の配置を探索した．その目的は符号器の数を増やしていった時の最適な配置の規則を調べることである．その実現のため，k-means アルゴリズムに基づくプログラムを作成し，最適な配置を実験的に求めてその結果を検証した．

## 3.1 ボロノイ図とボロノイ分割

情報源空間における符号器の配置はボロノイ図を用いてわかりやすく図示することができる．そこでまず，ボロノイ図の基本について簡単に説明する．

距離空間の中に母点と呼ばれる点がいくつか配置されていると仮定する．このとき，空間内の任意の点に対して最も近い母点を考えることができる．（最も近い母点は複数存在することもある.）各点に対して最も近い母点にしたがって空間全体を分割するとき，この分割を母点によるボロノイ分割という．また，各母点に対して，その母点が最も近い母点であるような領域をその母点のボロノイ領域という．ボロノイ領域の境界をボロノイ境界という．距離空間と

してユークリッド空間を考える場合が多いが，このときボロノイ境界は超平面となる．特に 2 次元ユークリッド空間ではボロノイ境界は直線である．2 次元空間におけるボロノイ分割は図で表すことができ，その図はボロノイ図と呼ばれる．図 1 は 2 次元ユークリッド空間におけるボロノイ図の例である．

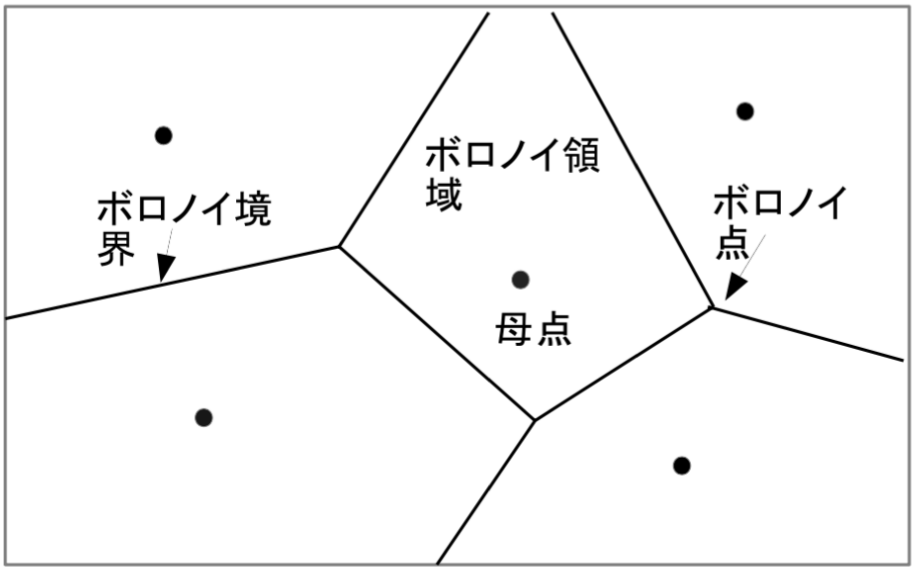


図 1　ボロノイ図の例

## 3.2　一様分布の場合

通常のボロノイ分割では，ユークリッド空間をユークリッド距離にしたがって分割するが，本研究では，情報源空間 $\mathcal{P}$ をダイバージェンスにしたがって分割する．つまり，母点 $Q$ に対し，情報源空間の点 $P$ の距離は $D(P||Q)$ と表される．これは先に述べた通り，情報源 $P$ を符号化確率 $Q$ で符号化した時の冗長度である．したがって母点 $Q$ は符号化確率 $Q$ をもつ符号器と同一視される．与えられた $n$ 個の母点の集合 $\mathcal{Q} = \{Q_1, Q_2, Q_3, \cdots, Q_n\}$ に対し，母点 $Q_i$ のボロノイ領域 $V(Q_i)$ は形式的には

$$V(Q_i) \triangleq \{P \in \mathcal{P} | D(P\|Q_i) \leq D(P\|Q_j), j = 1, \ldots, n\}$$

と定義される．アルファベットサイズが 3 の情報源に限定して，$V(Q_i)$ で構成された領域の境界線は通常のボロノイ領域の境界線と同様に直線によって形成されることを付録 $A$.1 より

示すことができる．

母点の集合として $\mathcal{Q}$ が与えられたとき，一様分布に対する平均冗長度は

$$\overline{D} = \sum_{Q \in \mathcal{Q}} \int_{P \in V(Q)} D\left(P \| Q\right) \mu(dP) \tag{1}$$

となる．上の式の $\mu$ は情報源空間上の一様分布である．母点の数が決められたときに，式 (1) の平均冗長度 (平均ダイバージェンス) を最小化する母点集合 $\mathcal{Q}$ を決めるのが基礎的検証の課題である．

## 3.3 k-means アルゴリズム

情報源空間における符号器の最適な配置を見つけるために k-means アルゴリズムを用いた．k-means 法とは，非階層型クラスタリングの一種であり，クラスタリング手法として最も広く使われる手法の一つである．この手法は，評価関数を最小化するクラスタ中心を見つけることによって，サンプルの集合を指定された個数のクラスタに分割する．情報源空間 $\mathcal{P}$ を $n$ 個の領域に分割する最適な母点の配置を求めるために，このアルゴリズムを適用する．すなわち，与えられた $n$ 個の母点にしたがってボロノイ領域を構成した後，各ボロノイ領域で平均冗長度が最小となるように母点の位置を更新する．新しい母点の位置の計算方法は付録 A.2 に補題 2 として示した．詳しいアルゴリズムは以下の通りである．

1. 母点の数 $n$ を入力する．
2. 母点に基づき情報源空間をボロノイ領域に分割する．
3. 各ボロノイ領域に対して，平均冗長度が最小となるように母点の位置を更新する．
4. 全ての母点に変化がなくなるまで，ステップ 2, 3 を繰り返す．
5. $n$ 個の母点の最適な位置を出力する．

この方法は，適切なクラスタ中心を求めて分割するという簡単なアルゴリズムであり，計算コストが小さいというメリットがある．しかし，この方法には，ランダムに決定される母点の初期値にクラスタリング結果が依存してしまう問題があるので，基礎的検証の実験では，全ての初期値をランダムで定め，母点の初期配置が特殊なものにならないようにした．

## 3.4 考察

基礎的検証ではアルファベットサイズが 3 の情報源に限定して実験した．その理由は次である．アルファベットサイズが 2 の情報源の場合は情報源空間は 1 次元なので得られる知見が少ない．アルファベットサイズが 4 以上の場合はボロノイ分割が図面に書けないので知見を把握しづらい．アルファベットサイズが 3 の場合はボロノイ分割を図面に書くことができるので

基礎的検証として最適である．

非可算無限個の一様分布はパソコン上で実現できないので，実験は $x$ 軸と $y$ 軸を均等に 100 分割し，多数の有限の点からなる情報源空間に対して，試験をした．結果は以下の図 2 にようになる．

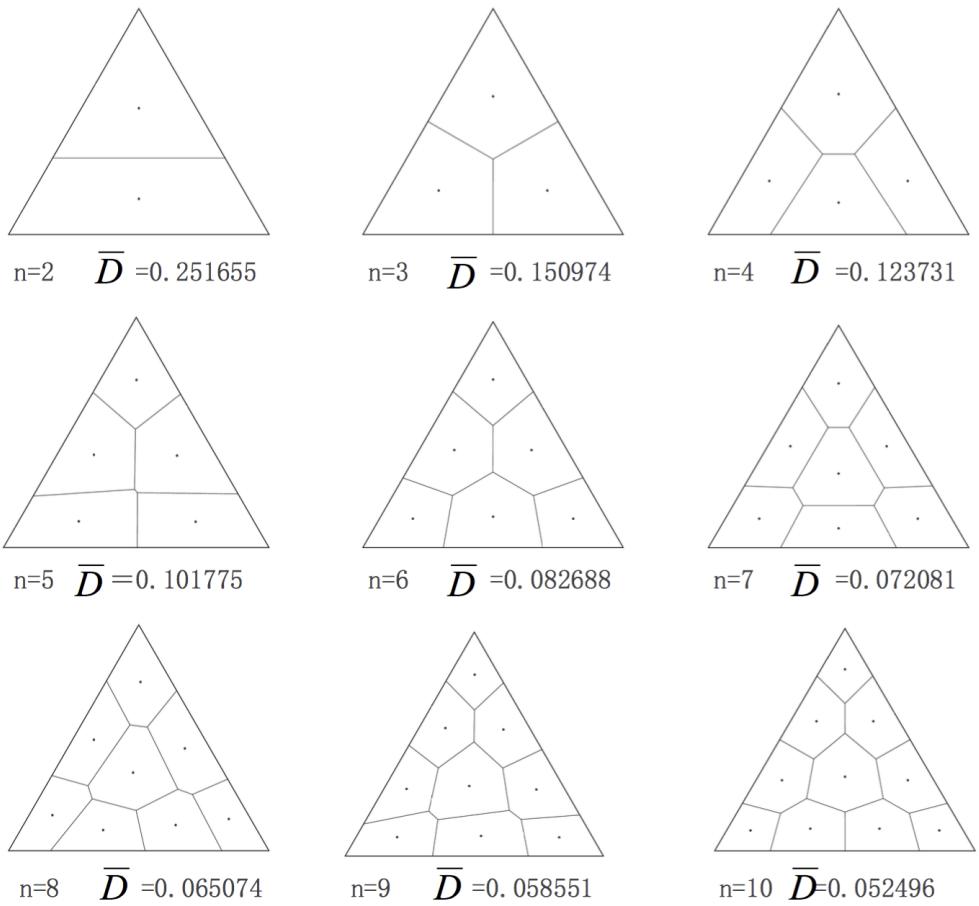


図 2　実験結果

表 1　実験結果の分類

| 符号器の数 | 線対称 | 点対象 |
|---|---|---|
| 2 | ○ | × |
| 3 | ○ | ○ |
| 4 | ○ | × |
| 5 | △ | × |
| 6 | ○ | ○ |
| 7 | ○ | ○ |
| 8 | × | × |
| 9 | × | × |
| 10 | ○ | ○ |

表 1 は実験結果の分類を示す (「△」は厳密な対称性ではないことを表す．)，$n = 8, 9$ の場合，線対称でも点対称でもない配置が最適となった．そこで情報源空間の分割数を大きくした場合もいくつか検証してみたが，やはり非対称な配置が最適となった．したがって，連続の情報源空間においても非対称な配置が最適になると予想される．

本段階最初の予想は符号器の数が $n^2$ なる時，構成されるボロノイ領域は図 3 にようになると予想していた．一方，最低一つの符号器が三角形の角の 2 等分線の上に配置されると予想していた．しかし，実験結果と予想は一致しなかった．残念ながら法則等は発見できなかった．

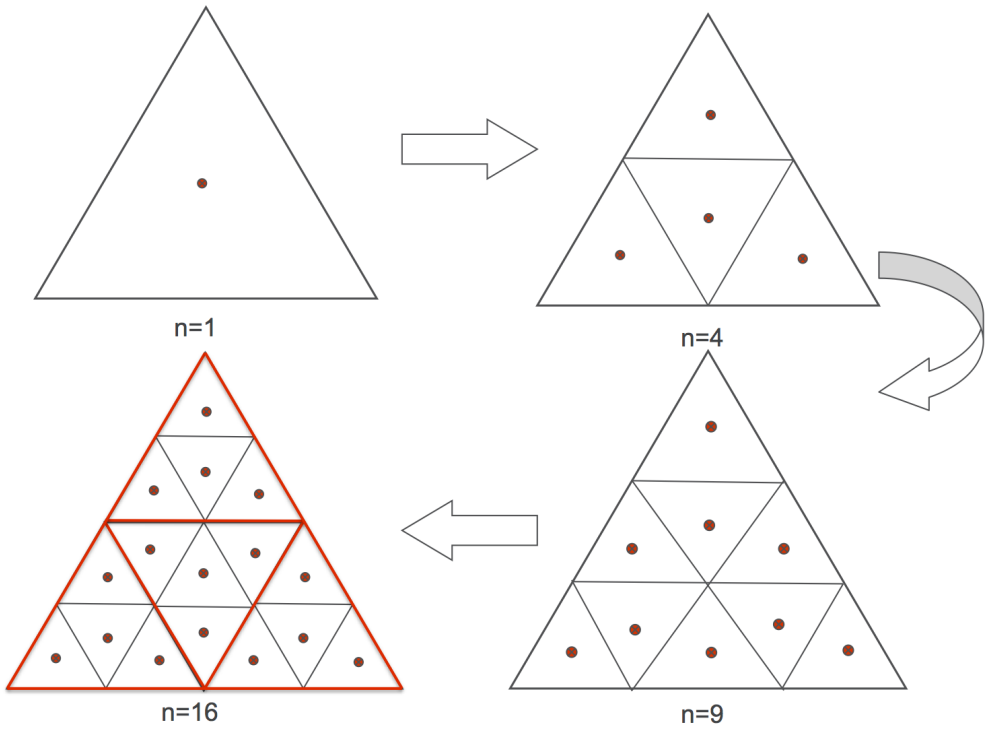


図 3　予想の配置

## 4　マルコフ情報源の符号化

ここから考察対象を有限個の情報源からなる情報源空間に変更する．特に，エルゴード的な遷移確率をもつマルコフ情報源を考えていく．エルゴード的な遷移確率をもつマルコフ情報源では，はじめにどんな状態分布が与えられても，十分な時間が経過すれば，状態の相対頻度は初期分布によらずに一つの分布に収束する．この分布を定常分布といい，定常分布を初期分布とすると情報源は定常となる．したがって十分長い時間の後ではこの情報源は定常情報源とみなすことができる．

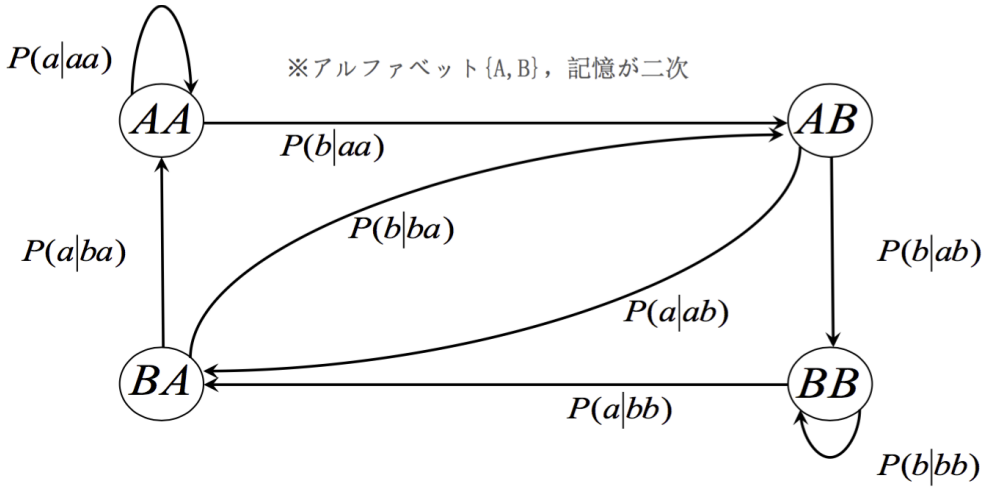


図 4　二次マルコフ過程

### 4.1　定常分布

状態遷移を十分長く繰り返すと，各ブロックを通過する頻度を計測することができる．このようにして具体的に計測した頻度は，各ブロックの出現確率 $P(AA), P(AB), P(BA), P(BB)$ を表している．したがって，最初の遷移によるブロックの出現確率は

$$
\begin{aligned}
P_1(AA) &= P(AA \mid AA)P_0(AA) + P(AA \mid BA)P_0(AA) \\
P_1(AB) &= P(AB \mid AA)P_0(AB) + P(AB \mid BA)P_0(AB) \\
P_1(BA) &= P(BA \mid BB)P_0(BA) + P(BA \mid AB)P_0(BA) \\
P_1(BB) &= P(BB \mid BB)P_0(BB) + P(BB \mid AB)P_0(BB)
\end{aligned}
$$

のようになる．以下，情報源からシンボルが出力されるたびに，ブロックの出現確率は下の漸化式にしたがって変化する．

$$
\begin{aligned}
P_{k+1}(AA) &= P(AA \mid AA)P_k(AA) + P(AA \mid BA)P_k(AA) \\
P_{k+1}(AB) &= P(AB \mid AA)P_k(AB) + P(AB \mid BA)P_k(AB) \\
P_{k+1}(BA) &= P(BA \mid BB)P_k(BA) + P(BA \mid AB)P_k(BA) \\
P_{k+1}(BB) &= P(BB \mid BB)P_k(BB) + P(BB \mid AB)P_k(BB)
\end{aligned}
$$

もし情報源がエルゴード的ならば，この遷移を無限に繰り返すと，ブロック $AA, AB, BA, BB$ が出現する確率は一定の値に収束する．言いかえれば, 十分長い時間さまよった後では，振りだしのブロックに関係せず，各ブロックを通過する頻度が一定の値に落ち着く．この状態のこ

とを定常状態といい，このときの確率分布をブロックの定常分布という．上の漸化式を無限に繰り返すと，実は一意な定常分布に収束することがいえる．したがって，定常分布は

$$\pi = \begin{pmatrix} P_\infty(AA) \\ P_\infty(AB) \\ P_\infty(BA) \\ P_\infty(BB) \end{pmatrix}$$

$$\mathcal{T} = \begin{pmatrix} P(AA \mid AA) & 0 & P(AA \mid BA) & 0 \\ P(AB \mid AA) & 0 & P(AB \mid BA) & 0 \\ 0 & P(BA \mid AB) & 0 & P(BA \mid BB) \\ 0 & P(BB \mid AB) & 0 & P(BB \mid BB) \end{pmatrix}$$

$$\pi = \mathcal{T}\pi$$

を満たす．この連立一次方程式を

$$P_\infty(AA) + P_\infty(AB) + P_\infty(BA) + P_\infty(BB) = 1$$

の条件下で解けば，定常分布を求めることができる．

本研究では，$n$ 個の状態を持つ情報源に対して，$k$ 個 $(k < n)$ の符号器を用いて符号化したとき，平均冗長度が最小となるような状態のグループを見つける問題を考える．

## 4.2 例題

アルファベット { 0,1,2 } の二次マルコフ情報源を考える．状態数は 9 である．まず例題として，表 2 のような遷移確率をもつ情報源を考える．

表 2　遷移確率

| | $S_{00}$ | $S_{01}$ | $S_{02}$ | $S_{10}$ | $S_{11}$ | $S_{12}$ | $S_{20}$ | $S_{21}$ | $S_{22}$ |
|---|---|---|---|---|---|---|---|---|---|
| $P(0\|S)$ | $\frac{1}{5}$ | $\frac{4}{7}$ | $\frac{1}{4}$ | $\frac{1}{3}$ | $\frac{2}{3}$ | $\frac{1}{6}$ | $\frac{5}{7}$ | $\frac{3}{4}$ | $\frac{2}{5}$ |
| $P(1\|S)$ | $\frac{2}{5}$ | $\frac{1}{7}$ | $\frac{1}{2}$ | $\frac{1}{3}$ | $\frac{1}{6}$ | $\frac{1}{2}$ | $\frac{1}{7}$ | $\frac{1}{8}$ | $\frac{2}{5}$ |
| $P(2\|S)$ | $\frac{2}{5}$ | $\frac{2}{7}$ | $\frac{1}{4}$ | $\frac{1}{3}$ | $\frac{1}{6}$ | $\frac{1}{3}$ | $\frac{1}{7}$ | $\frac{1}{8}$ | $\frac{1}{5}$ |

この遷移確率によって情報源の定常分布 $\pi = [\pi_{00}, \pi_{01}, \pi_{02}, ..., \pi_{22}]$ を計算することができる．状態 $S_{00}$, $S_{01}$, $S_{02}, ...$, $S_{22}$ 全てを $\mathcal{X}$ 上の分布とする．状態数より少ない符号器を用いた可能な配置は 21147 通り存在する．全ての可能な配置に対して平均冗長度を計算すると，限られた数の符号器を用いて最適な配置方法は表 3 のようになる．

表 3　最適配置

| 符号器の数 | 最適な配置方法 | 平均冗長度 |
|---|---|---|
| 8 | $\{ S_{11}, S_{20} \}, \{ S_{00} \}, \{ S_{01} \}, \{ S_{02} \},$ $\{ S_{10} \}, \{ S_{12} \}, \{ S_{21} \}, \{ S_{22} \}$ | 0.000142999 |
| 7 | $\{ S_{11}, S_{20}, S_{21} \}, \{ S_{00} \}, \{ S_{01} \},$ $\{ S_{02} \}, \{ S_{10} \}, \{ S_{12} \}, \{ S_{22} \}$ | 0.000574361 |
| 6 | $\{ S_{11}, S_{20}, S_{21} \}, \{ S_{00}, S_{12} \},$ $\{ S_{01} \}, \{ S_{02} \}, \{ S_{10} \}, \{ S_{22} \}$ | 0.00146615 |
| 5 | $\{ S_{11}, S_{20}, S_{21} \}, \{ S_{00}, S_{12} \},$ $\{ S_{10}, S_{22} \}, \{ S_{01} \}, \{ S_{02} \}$ | 0.00376444 |
| 4 | $\{ S_{11}, S_{20}, S_{21} \}, \{ S_{00}, S_{12}, S_{02} \},$ $\{ S_{10}, S_{22} \}, \{ S_{01} \}$ | 0.00712636 |
| 3 | $\{ S_{11}, S_{20}, S_{21}, S_{22} \}, \{ S_{00}, S_{02}, S_{10}, S_{12} \}, \{ S_{01} \}$ | 0.0135089 |
| 2 | $\{ S_{11}, S_{20}, S_{21} \}, \{ S_{00}, S_{01}, S_{02}, S_{10}, S_{12}, S_{22} \}$ | 0.0210248 |
| 1 | $\{ S_{11}, S_{20}, S_{21}, S_{00}, S_{01}, S_{02}, S_{10}, S_{12}, S_{22} \}$ | 0.101609 |

1 つの括弧は 1 つのクラスタ（状態の集合）を表しており，1 つの符号器で符号化することを意味している．その符号化確率はクラスタに対して最適なものを使用した．

# 5　符号器の配置を効率よく見つける方法

例題の結果を見ると，合わせたシンボルが多数の場合はずっと合わせることが分かる．例えば $S_{11}$ と $S_{20}$ は符号器の数が 8 個のときにひとつのクラスタを構成し，さらに符号器の数が減っても符号器の数が 2 個になるまで同じクラスタに属し続けている．最適な配置の結果はハフマン符号の符号木と似ているので，マルコフ情報源に対して，状態数より少ない符号器を用いた配置をハフマン符号の作り方と同様の手順で見つける方法を提案し，次の章で性能を解析する．

ハフマン符号の作り方はもっとも確率の小さい二つ状態を選び，選んだ二つの確率の和を割り付け，二つ状態を 1 つ状態として確率が 1 になるまで繰り返す．ハフマン符号の作り方を基礎として，本研究の課題に対して次のような手順で符号器（符号化確率）を決定する方法を提案する．

1.$n$ 個のクラスタの中から，付録 A.2 の補題 2 により，全体的に平均冗長度を最小化するように 2 つのクラスタを選び，これらを合わせて新たな 1 つのクラスタとして用いる．新たなクラスタに対して 1 つの符号器が割り当てられ，符号器の数は 1 つ減る．

2. 次に，手順 1 の操作をクラスタ数が 1 個になるまで繰り返す．

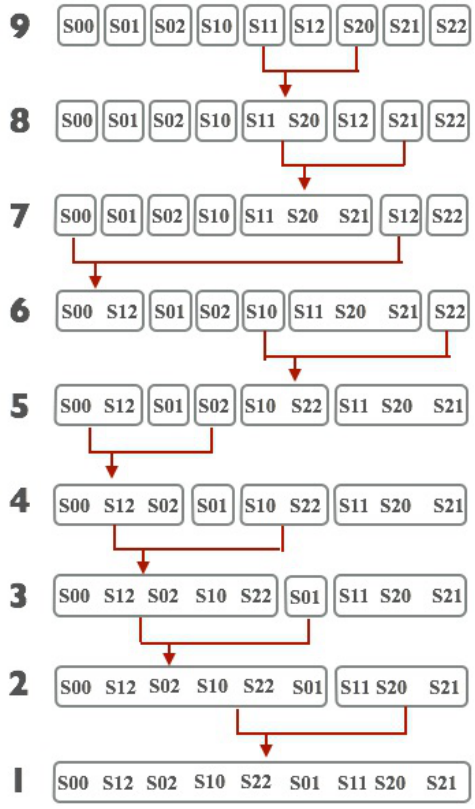


図 5　提案法の手続き

ここで表 2 によるアルファベット 3 つの二次マルコフ情報源の場合には，表 4 のように 8 回のステップを経てアルゴリズムが終了する．

表 4　提案法の結果

| | |
|---|---|
| 8 | $\{ S_{11}, S_{20} \}, \{ S_{00} \}, \{ S_{01} \}, \{ S_{02} \}, \{ S_{10} \}, \{ S_{12} \}, \{ S_{21} \}, \{ S_{22} \}$ |
| 7 | $\{ S_{11}, S_{20}, S_{21} \}, \{ S_{00} \}, \{ S_{01} \}, \{ S_{02} \}, \{ S_{10} \}, \{ S_{12} \}, \{ S_{22} \}$ |
| 6 | $\{ S_{11}, S_{20}, S_{21} \}, \{ S_{00}, S_{12} \}, \{ S_{01} \}, \{ S_{02} \}, \{ S_{10} \}, \{ S_{22} \}$ |
| 5 | $\{ S_{11}, S_{20}, S_{21} \}, \{ S_{00}, S_{12} \}, \{ S_{10}, S_{22} \}, \{ S_{01} \}, \{ S_{02} \}$ |
| 4 | $\{ S_{11}, S_{20}, S_{21} \}, \{ S_{00}, S_{12}, S_{02} \}, \{ S_{10}, S_{22} \}, \{ S_{01} \}$ |
| 3 | $\{ S_{11}, S_{20}, S_{21} \}, \{ S_{00}, S_{02}, S_{10}, S_{12}, S_{22} \}, \{ S_{01} \}$ |
| 2 | $\{ S_{11}, S_{20}, S_{21} \}, \{ S_{00}, S_{01}, S_{02}, S_{10}, S_{12}, S_{22} \}$ |
| 1 | $\{ S_{11}, S_{20}, S_{21}, S_{00}, S_{01}, S_{02}, S_{10}, S_{12}, S_{22} \}$ |

# 6　提案法の評価

提案法はすべての組み合わせのうち一部しか探索しないので，図 6 にような全ての最適な配置を見つけることができない．しかし，最適ではない配置になっている場合の平均冗長度を見ると，提案法は決して悪いとは言えない．本研究の提案を確認するため，さらに 100 個の例題をランダムに生成して提案法を検証した．

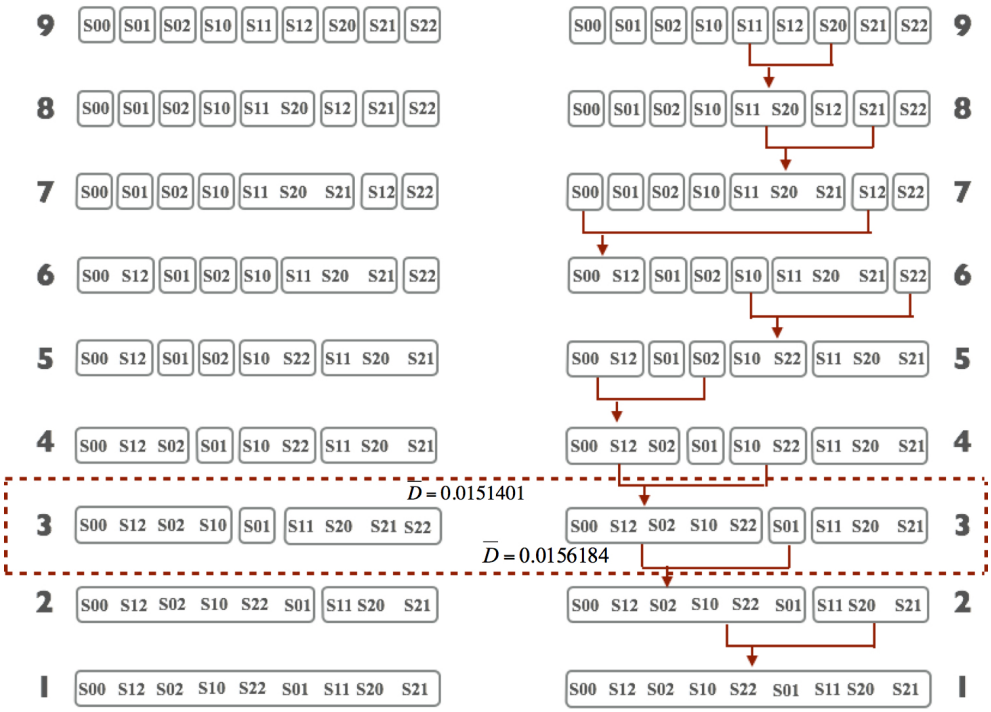


図 6　従来法（左）と提案法（右）の比較

## 6.1　計算量

提案法と従来法の計算時間はどちらもその調べる配置の組み合わせの数に比例すると考えられる．したがって，調べる配置の数を計算量とみなして両者を比較する．

符号器は少なくても 1 つ状態を代表されるので，従来法の計算量は第 2 種のスターリング数にしたがっている．第 2 種のスターリング数は，$n$ 個の要素からなる集合を $k$ 個の空でない部分集合に分ける場合の数であり，記号で $\left\{ {n \atop k} \right\}$ と表す [2]．例えば，4 個の要素からなる集合を

ふたつに分割する方法は，次の 7 通りが存在している．

$$\{1,2,3\}\bigcup\{4\}$$
$$\{1,2,4\}\bigcup\{3\}$$
$$\{1,3,4\}\bigcup\{2\}$$
$$\{2,3,4\}\bigcup\{1\}$$
$$\{1,2\}\bigcup\{3,4\}$$
$$\{1,3\}\bigcup\{2,4\}$$
$$\{1,4\}\bigcup\{2,3\}$$

したがって，$\left\{ {4 \atop 2} \right\} = 7$ である．漸化式を用いて

$$\left\{ {n \atop k} \right\} = k \left\{ {n-1 \atop k} \right\} + \left\{ {n-1 \atop k-1} \right\} \tag{2}$$

と表すこともできる．

一方，提案法の計算量は基本的な二項係数にしたがっている．基本的な二項係数は，$n$ 個の要素からなる集合を $k$ 個の部分集合に選ぶ場合の数であり，記号で $\binom{n}{k}$ と表す [2]．漸化式は

$$\binom{n}{k} = \frac{n(n-1)\cdots(n-k+1)}{k(k-1)\cdots(1)} \tag{3}$$

となる．

状態数が 9 個の場合に二つの方法に対して，必要な計算量は図 7 に示した．図を見ると，従来法の計算量は提案法の計算量と比べて，従来法の方が全て多いと分かる．3, 4, 5, 6 個符号器を用いる場合は従来法の計算量が特に多い．状態数が増えると，従来法の計算量は指数的に増加するので，本研究の提案は計算量の削減に有効である．

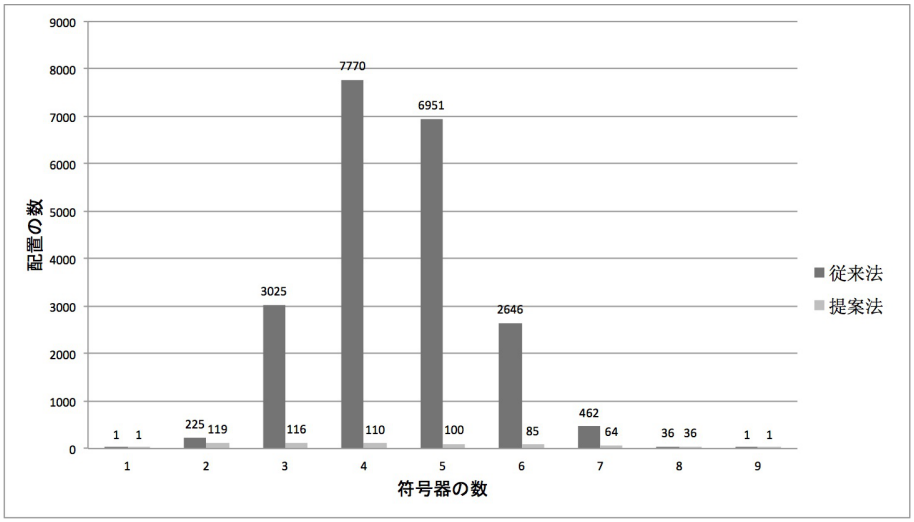


図 7　n=9 の場合

## 6.2　結果の最適性

提案法は必ずしも最適な配置を見つけるわけではないので，最適な配置を見つけていない場合の結果を最適な結果と比較する．100 個の例題に対する結果を以下の図 8 に示した．

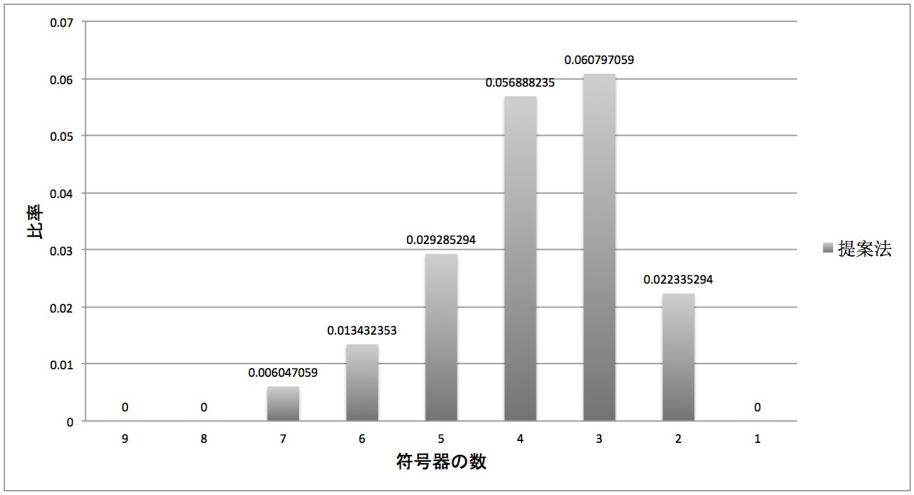


図 8　提案法の非最適性

図 8 の横軸は符号器を用いる数，縦軸は

$$\frac{1}{100}\sum_{100}(\frac{\text{提案法の冗長度}}{\text{従来法の冗長度}}-1)$$

である．図 8 を見て分かるように，最悪の場合で提案法の平均冗長度が 3.8 パーセント大きくなっている．また，最適な配置を見つけた数を表 5 に示した．符号器の数は 1 個から 8 個まで 8 通りの場合があるが，符号器の数が 1 個と 8 個は提案法と従来法の計算量は同じなので，一つの例題の中で最適でない配置の数の最大は 6 個である．表 6 の左側はそのうち最適でない配置の数で，右側はそうなるような例題の数である．表 5 と表 6 を見て分かるように，提案法を用いて最適な配置を見つけることが多いとが分かった．

表 5　100 個例題として実験した結果

| 符号器の数 | 最適な配置を見つけた数 |
|---|---|
| 8 | 100 |
| 7 | 96 |
| 6 | 94 |
| 5 | 86 |
| 4 | 75 |
| 3 | 65 |
| 2 | 64 |
| 1 | 100 |

表 6　100 個例題の考察

| 一つ例題の中で最適でない配置の数 | 最適でない配置の数の回数 |
|---|---|
| 0 個 | 36 |
| 1 個 | 29 |
| 2 個 | 19 |
| 3 個 | 11 |
| 4 個 | 4 |
| 5 個 | 0 |
| 6 個 | 1 |
| 合計 | 100 |

# 7　まとめ

本研究では，複数の状態をもつ情報源に対して，短い計算時間で準最適な配置を見つけることができる方法を提案して，提案法の評価を行った．提案法は最悪の場合で平均冗長度が平均約 4 パーセント以内を守ることができると分かった．

今までは状態の数が 9 個の場合しか検討してないので，今後の課題では，状態の数を増え，提案法の品質を検討したいと考えている．

# 謝辞

本研究をまとめるにあたり，指導教員の西新先生にいろいろご指導とご援助をいただき，心よりお礼申し上げる．

# 参考文献

[1] Thomas M. Cover and Joy A. Thomas, *Elements of Information Theory, Second Edition* , WILEY-INTERSCIENCE, 2006.

[2] GRAHAM. KNUTH. PATASHNIK, *Concrete Mathematics A Foundation For Computer Science* , 共立出版株式会社, 1993.

[3] Takuji Nishimura., Makoto Matsumoto, " A C-program for MT19937, " *http://www.math.sci.hiroshima-u.ac.jp/m-mat/MT/VERSIONS/C-LANG/mt19937-64.c*, 2014 年 4 月閲覧.

# 付録 A　補題の証明

## A.1　ボロノイ領域の境界線

与えられた $n$ 個の母点の集合 $\mathcal{Q} = \{Q_1, Q_2, Q_3, \cdots, Q_n\}$ に対し，母点 $Q_i$ のボロノイ領域 $V(Q_i)$ は形式的には

$$V(Q_i) \triangleq \{P \in \mathcal{P} | D(P\|Q_i) \leq D(P\|Q_j), j = 1, \ldots, n\}$$

と定義される．

**補題 1**　$V(Q_i)$ で構成された領域の境界線は通常のボロノイ領域の境界線と同様に直線によって形成される．

（証明）任意の隣母点 $Q \in \mathcal{P}$ と $S \in \mathcal{P}$ に対して，その 2 つ隣母点と冗長度が同じの点の集合 $P^* \in \mathcal{P}$ は

$$D\left(P^*\|Q\right) = D\left(P^*\|S\right) \tag{4}$$

と表される．したがって，式 (4) を満たす点の集合がボロノイ境界である．アルファベットサイズが 3 の情報源に限定して式展開すると，

$$P_1^* \log \frac{P_1^*}{Q_1} + P_2^* \log \frac{P_2^*}{Q_2} + P_3^* \log \frac{P_3^*}{Q_3} = P_1^* \log \frac{P_1^*}{S_1} + P_2^* \log \frac{P_2^*}{S_2} + P_3^* \log \frac{P_3^*}{S_3}$$

となる．整理すると，

$$P_1^* \log \frac{Q_1}{S_1} + P_2^* \log \frac{Q_2}{S_2} + P_3^* \log \frac{Q_3}{S_3} = 0 \tag{5}$$

となる．$\log \frac{Q_1}{S_1}$, $\log \frac{Q_2}{S_2}$, $\log \frac{Q_3}{S_3}$ は全部定数なので，平面の定義

一次方程式 $ax + by + cz + d = 0$ を満たす点 $(x, y, z)$ の軌跡は平面である．

より，式 (5) を満たす点の集合は平面となる．さらに，式 (4) を満たす集合は，情報源クラス $\mathcal{P}$ の部分集合であることから直線になる．アルファベットサイズが 3 の情報源に限定して冗長度で構成された領域の境界線は一般的なボロノイ領域の境界線と同様に直線によって形成されることをが示された．□

## A.2　平均冗長度の最小値

複数の情報源 $P_i$ がそれぞれ重み $\pi_i$ をもっているとする．このときすべての情報源をひとつの符号器 $Q$ で符号化するとその平均冗長度は

$$\overline{D}(Q) \triangleq \sum_i \pi_i D(P_i\|Q) \tag{6}$$

と表される．

**補題 2** $\overline{D}(Q)$ の最小値を与える $Q$ は

$$Q^* \triangleq \frac{\sum_i \pi_i P_i}{\sum_i \pi_i}$$

で与えられ，最小値は

$$\overline{D}(Q^*) = \sum_i \pi_i (H(Q^*) - H(P_i))$$

である．

（証明）式 (6) は

$$\begin{aligned}\overline{D}(Q) &= \sum_i \pi_i \sum_x P_i(x) \log \frac{P_i(x)}{Q(x)} \\ &= \sum_i \sum_x P_i(x) \log \frac{1}{Q(x)} - \sum_i \pi_i H(P_i)\end{aligned}$$

と書き直すことができるので，$\sum_i \sum_x P_i(x) \log Q(x)$ の最大化問題を考えればよい．そこで，ラグランジュの未定乗数法を用いてこの問題を解く．未定乗数 $\lambda$ を導入し，$Q$ の関数

$$\Phi(Q) \triangleq \sum_i \sum_x P_i(x) \log Q(x) + \lambda \left( \sum_x Q(x) - 1 \right)$$

を定義する．これを $Q(x)$ で偏微分して 0 とおくと

$$\frac{\partial \Phi(Q)}{\partial Q(x)} = \frac{\sum_i \pi_i P(x)}{Q(x)} + \lambda = 0$$

となり

$$Q(x) = -\frac{\sum_i \pi_i P_i(x)}{\lambda}$$

を得るが，$\sum_x Q(x) = 1$ から $\lambda = -\sum_i \pi_i$ となる．以上より $\overline{D}(Q)$ は

$$Q = Q^* \triangleq \frac{\sum_i \pi_i P_i}{\sum_i \pi_i}$$

のとき極値をとる．この極値は実際最小値で，その値は

$$\begin{aligned}
\overline{D}(Q^*) &= \sum_i \sum_x P_i(x) \log \frac{1}{Q^a st(x)} - \sum_i \pi_i H(P_i) \\
&= \sum_x \left( \sum_i \pi_i P_i(x) \right) \log \frac{1}{Q^a st(x)} - \sum_i \pi_i H(P_i) \\
&= \sum_x \left( Q^a st(x) \sum_i \pi_i \right) \log \frac{1}{Q^a st(x)} - \sum_i \pi_i H(P_i) \\
&= \sum_i \pi_i H(Q^*) - \sum_i \pi_i H(P_i) \\
&= \sum_i \pi_i (H(Q^*) - H(P_i))
\end{aligned}$$

となる．□

# 付録 B　ソースコード

## B.1　基礎的検証プログラム

```
#include <stdio.h>
#include <math.h>

struct pos
{
    double x, y;
};
struct line
{
    double a, b, c;
};

#define MAX_SEEDS 100
int num_of_seeds;
struct pos seed[MAX_SEEDS];

struct pos polygon[MAX_SEEDS];
int n_vertices;
struct pos
intersection(struct line l1, struct pos p0, struct pos p1)
{
    struct line l2;
    struct pos intersect;

    l2.a = p1.y - p0.y;
    l2.b = p0.x - p1.x;
    l2.c = p1.x * p0.y - p0.x * p1.y;

    intersect.x = (l1.b * l2.c - l2.b * l1.c) / (l1.a * l2.b - l1.b * l2.a);
    intersect.y = (l2.a * l1.c - l1.a * l2.c) / (l1.a * l2.b - l1.b * l2.a);

    return(intersect);
}

void
polytrim(struct line line, struct pos seed)
{
```

```
    struct pos newpoly[MAX_SEEDS];
    int n_v;
    double seedparity = line.a * seed.x + line.b * seed.y + line.c;
    int i, j;

    if (seedparity == 0.0)
    {
        return;
    }
    n_v = 0;
    for (i = 0; i < n_vertices; i++)
    {
        double parity0 = (line.a * polygon[i].x + line.b * polygon[i].y + line.c) * seedparity;
        if (parity0 > 0.0)
        {
            newpoly[n_v++] = polygon[i];
            j = (i + 1) % n_vertices;
            if ((line.a * polygon[j].x + line.b * polygon[j].y + line.c) * seedparity < 0.0
            {
                newpoly[n_v++] = intersection(line, polygon[i], polygon[j]);
            }
        }
        else if (parity0 < 0.0)
        {
            j = (i + 1) % n_vertices;
            if ((line.a * polygon[j].x + line.b * polygon[j].y + line.c) * seedparity > 0.0
                {
                    newpoly[n_v++] = intersection(line, polygon[i], polygon[j]);
                }
        }
        else
        {
            newpoly[n_v++] = polygon[i];
        }
    }
    n_vertices = n_v;
    for (i = 0; i < n_vertices; i++)
    {
        polygon[i] = newpoly[i];
    }
    return;
}

void
bound(int n)
{
    int i;
    polygon[0].x = 0.0;
    polygon[0].y = 0.0;
    polygon[1].x = 1.0;
    polygon[1].y = 0.0;
    polygon[2].x = 0.0;
    polygon[2].y = 1.0;
    n_vertices = 3;

    for (i = 0; i < num_of_seeds; i++)
    {
        if (i == n)
        {
            continue;
        }
        double ra = log(seed[n].x / seed[i].x);
        double rb = log(seed[n].y / seed[i].y);
        double rc = log((1.0 - seed[n].x - seed[n].y) / (1.0 - seed[i].x - seed[i].y));
        struct line line = {ra - rc, rb -rc, rc};
        polytrim(line, seed[n]);
    }
    return;
}
```

```
struct pos
transform(struct pos p)
{
    struct pos pos;
    pos.x = 346.41016 * p.x + 173.20508 * p.y - 173.20508;
    pos.y = 300.0 * p.y - 100;
    return(pos);
}

void int
main(int argc, char** argv)
{
    char epsfilename[256];
    FILE *fp;
    FILE *eps;
    int index[2];
    if (argc < 2)
    {
        printf("Specify input file.\n");
        return(1);
    }
    if ((fp = fopen(argv[1], "r")) == NULL)
    {
        printf("open error: %s\n", argv[1]);
        return(1);
    }
    fscanf(fp, "%s", epsfilename);
    if ((eps = fopen(epsfilename, "w")) == NULL)
    {
        printf("open error: %s\n", epsfilename);
        return(1);
    }
    if (fscanf(fp, "%d%d", index, index + 1) != 2)
    {
        printf("syntax error: %s\n", argv[1]);
        return(1);
    }
    for (num_of_seeds = 0; num_of_seeds < MAX_SEEDS; num_of_seeds++)
    {
        double v[3];
        if (fscanf(fp, "%lf%lf%lf", v, v + 1, v + 2) != 3)
        {
            break;
        }
        seed[num_of_seeds].x = v[index[0]];
        seed[num_of_seeds].y = v[index[1]];
    }
    fclose(fp);
    draw(eps);
    fclose(eps);
    return(0);
}
```

## B.2 符号器の配置を効率よく見つける方法プログラム

```
#include <iostream>
#include <vector>
#include <stdio.h>
#include <cmath>
#include <math.h>
#include <stdlib.h>
#include "random.h"
using namespace std;
```

```
const int MAX_SEEDS = 9;
vector<double>  stationary_distribution;
vector<double> optimal_D;
vector<vector<vector<int>>> optimal_Con;
double H_P =0.0;
vector<vector<double>> seed;
vector<vector<vector<int>>> proposal_Con;
vector<double> proposal_D;
double init_time=0;
double formerly_time=0;
double proposal_time=0;

void probability(vector<double> &stationary_distribution)
{
    double a = 1.0 / MAX_SEEDS;
    for (int i = 0; i < MAX_SEEDS; i++)
    {
        stationary_distribution.push_back(a);
    }
    for (int ii = 0; ii < 2000; ii++)
    {
        vector<double>temp;
        temp.clear();
        for (int i = 0; i < MAX_SEEDS / 3; i++)
        {
            for (int j = 0; j < 3; j++)
            {
                double temp_S = 0;
                temp_S = seed[i][j] * stationary_distribution[i]
                + seed[i + MAX_SEEDS / 3][j] * stationary_distribution[i + MAX_SEEDS / 3]
                + seed[i + 2 * MAX_SEEDS / 3][j] * stationary_distribution[i + 2 * MAX_SEEDS /3];
                temp.push_back(temp_S);
            }
        }
        if(temp == stationary_distribution)
        {
            cout <<" ok "<< ii <<endl;
            break;
        }
        stationary_distribution.clear();
        for (int i = 0; i < temp.size(); i++)
        {
            stationary_distribution.push_back(temp[i]);
        }
    }
}

void change_1(vector<double> &a)
{
    vector<double>b;
    for(int i = 0; i < a.size(); i++)
    {
        b.push_back(a[i]);
    }
    a.clear();
    for(int i = b.size() - 1; i >=0; i--)
    {
        a.push_back(b[i]);
    }
    return;
}

void change_3(vector<vector<vector<int>>> &a)
{
    vector<vector<vector<int>>> b;
    for(int i = 0; i < a.size(); i++)
    {
        b.push_back(a[i]);
```

```
    }
    a.clear();
    for(int i = b.size() - 1; i >=0; i--)
    {
        a.push_back(b[i]);
    }
    return;
}

double entropy()
{
    double temp = 0;
    for (int i = 0; i < seed.size(); i++)
    {
        temp += stationary_distribution[i]*(-seed[i][0] * log(seed[i][0])
        - seed[i][1] * log(seed[i][1])  - seed[i][2] * log(seed[i][2]));
    }
    return temp;
}

double entropy_Q(vector<vector<int>>a)
{
    double temp0 = 0;
    for (int i = 0; i < a.size(); i++)
    {
        if(!a[i].empty())
        {
            double temp1 = 0, temp2 = 0, temp3 = 0, temp4 = 0;
            for (int j = 0; j < a[i].size(); j++)
            {
                temp1 += stationary_distribution[a[i][j]];
                temp2 += seed[a[i][j]][0] * stationary_distribution[a[i][j]];
                temp3 += seed[a[i][j]][1] * stationary_distribution[a[i][j]];
                temp4 += seed[a[i][j]][2] * stationary_distribution[a[i][j]];
            }
            temp0 += temp1*(-(temp2 / temp1) * log(temp2 / temp1)
            - (temp3 / temp1) * log(temp3 / temp1) - (temp4 / temp1) * log(temp4 / temp1));
        }
    }
    return temp0;
}

void entr(vector<vector<int>>a, vector<double>b)
{
    for (int i = 0; i < a.size(); i++)
    {
        double temp1 = 0, temp2 = 0, temp3 = 0, temp4 = 0;
        for (int j = 0; j < a[i].size(); j++)
        {
            temp1 += b[a[i][j]];
            temp2 += seed[a[i][j]][0] * b[a[i][j]];
            temp3 += seed[a[i][j]][1] * b[a[i][j]];
            temp4 += seed[a[i][j]][2] * b[a[i][j]];
        }
        cout << "Distributed of Q   " << " ( " << temp2 / temp1 << " , "
        << temp3 / temp1 << " , " << temp4 / temp1 << " ) " << endl;
    }
}

void print_group(int n, int m, int digits[],vector<double> &optimal_D,vector<vector<vector<int>>> &optimal_Con)
{
    //printf("%d ", m);
    vector<int>temp_1;
    vector<vector<int>>temp_2;
    for (int d = 0; d < m; d++)
    {
        //printf("(");
        for (int i = 0; i < n; i++)
        {
```

```
            if (digits[i] == d)
            {
                //printf("%d", i);
                temp_1.push_back(i);
            }
        }
        //printf(")");
        temp_2.push_back(temp_1);
        temp_1.clear();
    }
    //printf("\n");
    double temp000=0;
    temp000=entropy_Q(temp_2)-H_P;
    if(temp000 < optimal_D[m-1])
    {
        optimal_D[m-1]=temp000;
        optimal_Con[m-1]=temp_2;
    }
    return;
}

void reference(int a,int b,vector<vector<int>> &cc )
{
    for(int i=0;i< cc[b].size();i++)
    {
        cc[a].push_back(cc[b][i]);
    }
    cc[b].clear();
}

int main()
{
    init_genrand((unsigned)time(NULL));
    for (int i = 0; i < MAX_SEEDS; i++)
    {
        vector<double> temp;
        double a=0;
        double b=0;
        double c=0;
        a=genrand_res53();
        b=genrand_res53();
        c=1-a-b;
        if(c<0)
        {
            a=1-a;
            b=1-b;
            c=1-a-b;
        }
        temp.push_back(a);
        temp.push_back(b);
        temp.push_back(c);
        seed.push_back(temp);
        temp.clear();
    }
    probability (stationary_distribution);
    H_P =entropy();
    for (int i = 0; i < MAX_SEEDS; i++)
    {
        double a_a = 10000;
        optimal_D.push_back(a_a);
        proposal_D.push_back(a_a);
    }
    for (int i = 0; i < MAX_SEEDS; i++)
    {
        vector<vector<int>>b_b;
        vector<int>c_c;
        for (int j = 0; j < MAX_SEEDS; j++)
        {
            c_c.push_back(j);
```

```
            b_b.push_back(c_c);
            c_c.clear();
        }
        optimal_Con.push_back(b_b);
        proposal_Con.push_back(b_b);
        b_b.clear();
    }
    init_time=(double)clock()/CLOCKS_PER_SEC;
    cout <<"     init time = "<<  (double)clock()/CLOCKS_PER_SEC << endl;

    int i, r = 0;
    int max[MAX_SEEDS + 1];
    int digits[MAX_SEEDS];

    max[0] = 0;
    for (i = 0; i < MAX_SEEDS; i++)
    {
        max[i + 1] = 1;
        digits[i] = 0;
    }
    for (;;)
    {
        print_group(MAX_SEEDS, max[MAX_SEEDS], digits,optimal_D,optimal_Con);
        r++;

        for (i = MAX_SEEDS - 1; i >= 0; i--)
        {
            if (digits[i] < max[i])
            {
                digits[i]++;
                max[i + 1] = (digits[i] < max[i]) ? max[i] : digits[i] + 1;
                while (++i < MAX_SEEDS){
                max[i + 1] = max[i];
            }
            break;
        }
        digits[i] = 0;
    }
    if (i < 0){
    break;
    }
}

formerly_time=(double)clock()/CLOCKS_PER_SEC;
cout <<" formerly time = "<<  (double)clock()/CLOCKS_PER_SEC << endl;

for(int r=0;r< MAX_SEEDS-1;r++)
{
    vector<vector<int>> temp_0_0_0;
    for(int i=0;i<MAX_SEEDS;i++)
    {
        vector<int>abc;
        abc.push_back(0);
        temp_0_0_0.push_back(abc);
    }
    for (int i=0;i < proposal_Con.size(); i++)
    {
        for(int j=i+1;j<proposal_Con.size();j++)
        {
            temp_0_0_0 = proposal_Con[r];
            if(!temp_0_0_0[i].empty() && !temp_0_0_0[j].empty())
                {
                    reference(i,j,temp_0_0_0);
                    double temp000=0;
                    temp000=entropy_Q(temp_0_0_0)-H_P;
                    if(temp000 < proposal_D[r+1])
                    {
                        proposal_D[r+1] = temp000;
                        proposal_Con[r+1] = temp_0_0_0;
```

```
                    }
                }
            }
        }
    }
    proposal_time=(double)clock()/CLOCKS_PER_SEC;
    cout <<" proposal time = "<<  (double)clock()/CLOCKS_PER_SEC << endl;
    return(0);
}
```