信州大学
大学院総合理工学研究科

修士論文

A^* アルゴリズムを用いた最適な AIFV 符号の探索アルゴリズム

指導教員　西新 幹彦 准教授

専攻　工学専攻
分野　電子情報システム学分野
学籍番号　16W2073G
氏名　山川　誠史

2018 年 2 月 20 日

目次

1　はじめに

ハフマン符号は 1 つの符号木を用いて符号化レートを最小にする FV 符号として知られているが，文献 [1] では，複数の符号木を用い，ハフマン符号よりも符号化レートが小さい AIFV 符号が提案されている．符号語を受け取ってから対応する情報源シンボルを復号するのに必要な先読みする符号シンボルの数を復号遅延という．ハフマン符号は瞬時符号であるため復号遅延の値は 0 となるが，AIFV 符号は復号遅延に対して正の上界を設けることでハフマン符号よりも小さい符号化レートを達成している．このことは言い換えると，ハフマン符号は符号木で表現するとすべての情報源シンボルは葉に割り当てられているが，AIFV 符号では情報源シンボルを各符号木の葉だけではなく内部ノードにも割り当てるということである．

最適な AIFV 符号は整数計画法により見つけることができるが [1]，最近他の探索の方法として動的計画法に基づく方法 [3, 4] が提案されている．本論文ではさらに別の方法として A^* アルゴリズムを用いた AIFV 符号の探索アルゴリズムを提案する．ただし本論文では従来法との優劣は論じない．ちなみに，A^* アルゴリズムは最適なリバーシブル符号の探索に適用することもできる [2].

本論文では 2 章で 2 元 AIFV 符号について説明し，3 章で最適な 2 元 AIFV 符号を探索する探索アルゴリズムを提案し，その最適性を証明する．次に 4 章で 3 元 AIFV 符号の探索へと提案アルゴリズムを拡張する．次に 5 章で最適な AIFV 符号の例をいくつか示し，6 章では本論文で提案するアルゴリズムの計算量を見積もる．最後に 7 章で本論文をまとめる．

2　2 元 AIFV 符号

この章では，2 元 AIFV 符号の定義と例を示す．2 元 AIFV 符号の定義と符号化と復号化の手続きは以下のとおりである．

定義　（2 元 AIFV 符号）[1]

1. 2 元 AIFV 符号は 2 個の符号木からなる．それらを T_0, T_1 と表す．
2. 1 つだけの子をもつ不完全な内部ノードは主ノードと副ノードに分けられる．主ノードの子は副ノードでなくてはならない．主ノードはその孫へラベル 00 で繋がっている．
3. T_1 のルートノードは 2 つの子をもち，00 で繋がる孫をもたない．T_1 のルートノードからラベル 0 で繋がる子ノードは副ノードである．
4. 情報源シンボルは符号木の葉または主ノードに割り当てられる．

2 元 AIFV 符号による符号化の手続きは次のようになる．

手続き　（2 元 AIFV 符号による符号化）

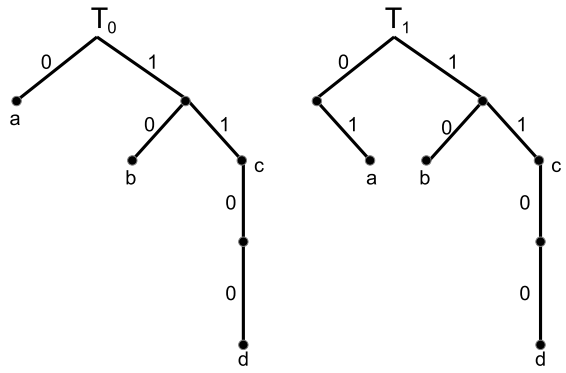

図 1　2 元 AIFV 符号

1. 最初の情報源シンボルは T_0 によって符号化する．
2. 情報源シンボルが葉 (または主ノード) によって符号化されたら，次のシンボルの符号化には T_0 (または T_1) を用いる．

2 元 AIFV 符号による復号化の手続きは次のようになる．

手続き　(2 元 AIFV 符号による復号化)

1. 符号化と同様に最初の符号語系列は T_0 によって復号化する．
2. 復号に用いる符号木のルートから観測した符号シンボルの順番に枝をたどり，次の符号シンボルが割り当てられた枝をたどる方法がない場合は，今指しているノードに割り当てられた情報源シンボルを復号する．今指しているノードに情報源シンボルが割り当てられていない場合は親ノードに割り当てられた情報源シンボルを復号する．
3. 符号化と同様に符号語系列が葉 (または主ノード) によって復号化されたら，次の符号語系列の復号化には T_0 (または T_1) を用いる．

次に 2 元 AIFV 符号の例として情報源アルファベット $\mathcal{X} = \{a, b, c, d\}$ をもつ情報源に対する 2 元 AIFV 符号を表す符号木を文献 [1] から引用して図 1 に示す．情報源系列を長さ 3 の aca とし，図 1 を用いてこの情報源系列を符号化する．まず，1 つ目の情報源シンボル a は T_0 を用いて 0 と符号化される．次に a が T_0 の葉で符号化されたため次の情報源シンボル c も T_0 を用いて 11 と符号化される．最後に c が T_0 の主ノードで符号化されたため次の情報源シンボル a は T_1 を用いて 01 と符号化される．

続いて符号化により得られた符号語系列 01101 が aca に正しく復号されるか確認する．まず，復号器に 0 が入力されて T_0 のルートから 0 が割り当てられた枝をたどる．T_0 ではこれ以上枝をたどることが出来ないため今指しているノードに割り当てられた情報源シンボル a が

復号される．次に復号された情報源シンボル a が T_0 の葉に割り当てられているため次の 1 から始まる符号語系列は T_0 を用いて復号される．まず 1 が入力されて T_0 のルートから 1 が割り当てられた枝をたどる．同様に 1 が入力されて 1 が割り当てられた枝をたどり次に 0 が入力され 0 が割り当てられた枝をたどる．その次に 1 が入力されるが今指しているノードから 1 が割り当てられた枝をたどることが出来ないため c が割り当てられたノードまで戻り情報源シンボル c を復号し，次の符号語系列 01 は T_1 を用いて情報源シンボル a が復号される．これで，情報源系列 aca が正しく復号されることが確認できた．

3　最適な 2 元 AIFV 符号

AIFV 符号では 2 つの符号木を使用する．したがって，AIFV 符号の平均符号語長は各符号木の平均符号語長を符号木の定常分布で重み付けしたものになる．現在知られている最適符号探索アルゴリズム [1,3,4] は各符号木を独立に最適化する方法に基づいており，一方の符号木の最適化の際には他方の符号木の平均符号語長を与えなければならない．すなわち，一方の最適化の際には他方の最適化の結果が必要になる．そこで，これらのアルゴリズムでは，最適化のパラメータを導入することで個別の最適化を行い，そのパラメータを更新しながら最適な AIFV 符号を探索する．したがって，探索アルゴリズムは与えられたパラメータのもとでの符号木の最適化の部分とパラメータの更新の部分に大きく分けられる．

続く節では，まずパラメータの更新部分のアルゴリズムを説明し，その次の節でパラメータに基づく符号木の最適化のアルゴリズムを説明する．後半の符号木の最適化アルゴリズムが本研究の提案である．

3.1　最適化のパラメータとその更新方法

この節では，最適な AIFV 符号を探索する際に用いられるパラメータとその更新方法について文献 [1] に基づいて説明する．必要なパラメータは 1 つの実数値であり，本論文ではこれをロスと呼ぶ．ロスは，それぞれの符号木を最適化する際に用いられ，2 つの符号木の性能の差を表している．この節で説明するアルゴリズムはロスの値を更新しながら最適なロスの値を求めるアルゴリズムとも言える．ロスの初期値を $s^{(0)}$ と表す．原理的にはロスの初期値は任意だが，0.405 が最適値のよい近似であり [1]，これを初期値とすることで更新回数を節約することができる．次に自然数 m(最初は $m = 1$) と与えられたロス $s^{(m-1)}$(最初は $s^{(0)}$) に対しコストと呼ばれる値を最小にする AIFV 符号の符号木 $T_0^{(m)}, T_1^{(m)}$ をそれぞれ求める．コストは平均符号語長とほぼ同じ概念だが，全体の符号の最適性がロスの値によって考慮されているところが異なる．ロスとコストの関係は次節で明記する．次に求めた $T_0^{(m)}, T_1^{(m)}$ から新しいロ

Algorithm 1 最適な 2 元 AIFV 符号の探索アルゴリズム

入力:各情報源シンボルの出現確率 $p_i, (i = 0, 1, \ldots, d), s^{(0)} = 0.405$
出力:最適な AIFV 符号 $\boldsymbol{T} = (T_0, T_1)$

1:$m = 0$ とする.
2:**do**
3: $m \longleftarrow m + 1$
4: $s^{(m-1)}$ に対してコストが最小となる $T_0^{(m)}, T_1^{(m)}$ をそれぞれ求める.
5: 式 (1) を用いて $s^{(m)}$ の値を求める.
6:**while** $s^{(m-1)} \neq s^{(m)}$
7:$T_0^{(m)}, T_1^{(m)}$ を T_0, T_1 として出力.

図 2　最適な 2 元 AIFV 符号を構成するアルゴリズム

スの値を

$$s^{(m)} = \frac{l_1^{(m)} - l_0^{(m)}}{q_0^{(m)} + q_1^{(m)}} \tag{1}$$

のように求める [1]. ここで $l_0^{(m)}, l_1^{(m)}$ はそれぞれ $T_0^{(m)}, T_1^{(m)}$ に対する平均符号語長とし, $q_0^{(m)}, q_1^{(0)}$ は符号木の遷移確率であり $q_0^{(0)} = Q(T_1^{(m)}|T_0^{(m)}), q_1 = Q(T_0^{(m)}|T_0^{(0)})$ とする. 次に $s^{(m-1)}$ と s^m を比べてみて値が異なっていれば, m を $m + 1$ に置き換えて上記の手順を繰り返す. $s^{(m-1)}$ と s^m の値が同じであった場合はそのときに求まっている $T_0^{(m)}, T_1^{(m)}$ を最適な AIFV 符号として出力する. このアルゴリズムをまとめたものを図 2 に Algorithm1 として示す.

3.2　2 元 AIFV 符号の各符号木に対しコストを最小にするアルゴリズム

この節では与えられたロス s に対してコストを最小にする符号木 T_0, T_1 の探索方法を提案する. その探索方法には A^* アルゴリズムを用いる. 一般に A^* アルゴリズムとは, 広い意味で最短経路を探索するアルゴリズムである. A^* アルゴリズムでは目的地に至る途中の経路が最短経路の候補として複数保持されており, 各径路には目的地までの距離の推定値が割り当てられている. 複数ある経路の候補の中から推定値が最も短いものを優先的に探索することで最短経路を早く見つけることができる. 特に探索を一歩進める時に距離の推定値が単調増加になる場合は, 最初に見つけた経路が最短経路となる [5].

最短経路探索を最適符号探索に適用する方法の概略を説明する. 最適符号探索の目的は平均符号語長が最小になるようにすべての情報源シンボルに符号語を割り当てることである. 符号

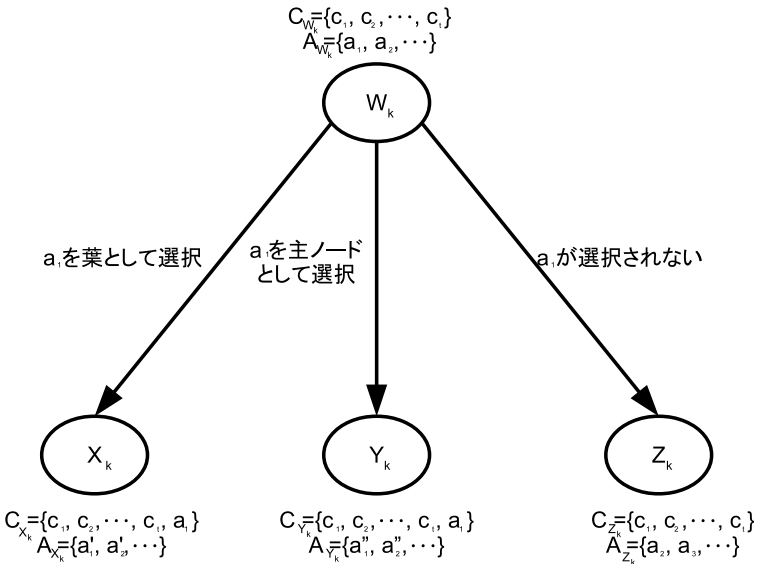

図 3　A^* アルゴリズムによって生成される状態の関係

探索の出発点は符号語が割り当てられた情報源シンボルが存在しない状態であり，探索を一歩進めることはあるひとつの情報源シンボルにある文字列を符号語として割り当てるか割り当てないかを決定することに対応する．また，経路の距離の推定値は平均符号語長の推定値に対応しており，さらに以下で提案する手順で探索をすると平均符号語長の推定値は探索を進めるごとに単調増加になるため最初に見つかった符号が最適である．このことについては後で証明する．

情報源アルファベットを $\mathcal{X} = \{1, 2, \ldots, d\}$ とし，それぞれの出現確率が $p_i, i \in \mathcal{X}$ で定まるとする．ただし，$i < j$ のとき $p_i \geq p_j$ とする．また，文字列 c の長さを $l(c)$ と表す．ロス s に対しコストを最小にする符号木 $T_k(k = 0, 1)$ を探索する A^* アルゴリズムによって生成される状態の関係を図 3 に示す．本論文で提案するアルゴリズムではシンボルへの符号語の割り当て方が 2 種類あるので，割り当てない場合と合わせて 3 つの子状態を作ることで 3 分木を作りながら探索を進める．

各状態 S は，コスト $e(S)$ と，符号語として割り当てられた文字列の並び C_S と，符号語の候補の並び A_S により構成される．ここで，A_S の中身の文字列は文字列長が短い順で辞書的順序に並べるものとする．

一般に状態 W から 3 つの子状態 X, Y, Z を作るには次の手順を行う．X は A_W の a_1 を葉として選択することで作成される．つまり，a_1 を C_W の最後に加え C_X とし，A_W から a_1 の子孫となる符号語を消去し A_X とする．次に Y は A_W の a_1 を主ノードとして選択すること

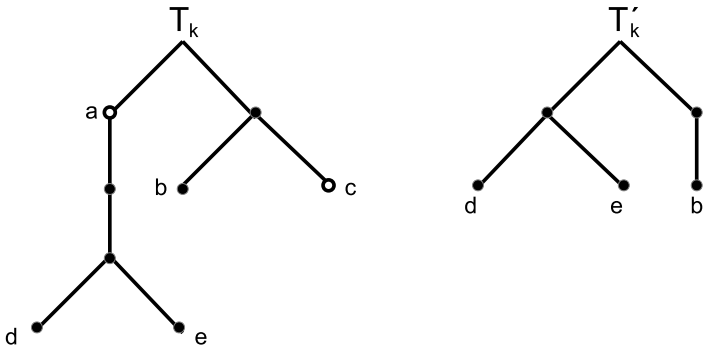

図 4　T_k と T'_k の例

で作成される．つまり，a_1 を C_W の最後に加え C_Y とし，A_W から a_1 の子孫のうち a_100 を祖先に持たない符号語を消去し A_Y とする．最後に Z は A_W の a_1 を符号語として選択しないことで作成される．つまり，A_W から a_1 を消去し，A_Z とし，C_Z はそのままの状態で C_Z とする．

符号語の候補の文字列を葉として選択する時，符号語の数が情報源シンボルの数に達しておらず，符号語の候補の数が 0 個であるような状態が生成された場合は，その状態ではこれ以上探索することが出来ないためその状態を消去する．候補の数が 0 個である確認にはクラフトの不等式を次のように用いる．まず，すでに符号語として選択されている文字列のみで符号木 T_k を作る．次に符号木 T_k を元に以下の手順に従い符号木 T'_k を作る．T_k と T'_k の例を図 4 に示す．図 4 では主ノードを ◯ と表した．T'_k は，まず T_k から主ノードに割り当てられた情報源シンボルをすべて消去し，次に主ノードが子孫を持たない場合はその主ノードとその主ノードの親ノードに繋がる枝を消去し，さらに主ノードが孫を持つ場合はその主ノードとその孫へ繋がるまでの枝とノードも消去して，消去した主ノードの孫をルートとする T_k の部分木を消去した主ノードの親ノードに枝でつなげることで生成される．符号語の候補の数が 0 個であることは T'_k に対しクラフトの不等式を適用することで確認できる．

まず $k = 0$ の時 (T_0 を構成する場合) に，ある文字列を葉として選択した場合を考える．このとき T'_0 に対しクラフトの不等式が 1 以上の時符号語の候補が 0 個になるため，

$$\sum_{i=1}^{t} u_i 2^{-(l(c_i)-2w_i)} \geq 1 \tag{2}$$

を満たし符号語の数が情報源シンボルの数に達していない場合その状態を削除する．ここで，c_i は情報シンボル i に割り当てられた符号語であり，u_i は c_i が葉のときに 1 でそれ以外なら 0 となる値とし，w_i は c_i の祖先の内，主ノードであるノードの数とし，t はすでに選択されている符号語の数とする．

次に $k=1$ の時 (T_1 を構成する場合) に，ある文字列を葉として選択した場合を考える．T_1 ではルートが 00 で繋がる孫を持つことが出来ないため，クラフトの不等式が $\frac{3}{4}$ 以上の時，符号語の候補の数が 0 個になる．したがって，

$$\sum_{i=1}^{t} u_i 2^{-(l(c_i)-2w_i)} \geq \frac{3}{4} \tag{3}$$

を満たし符号語の数が情報源シンボルの数に達していない場合その状態を消去する．また，主ノードは必ず子孫を持つため，C_X（もしくは C_Y）の要素の中に現在子孫を持たない主ノードの数が $d-t$ より多い場合 X(もしくは Y) を消去する．ここで t は符号語として選択されている文字列の数とする．

次に子を持たない状態を各々に割り当てられたコストの小さい順に並べ，最もコストの小さい状態を取り出し，上の手順を繰り返す．

ここでコストの計算方法を説明する．まず，W_k において C_{W_k} のコスト $g(W_k)$ を，

$$g(W_k) = \sum_{i=1}^{t} \{u_i p_i l(c_i) + v_i p_i (l(c_i)+s)\} \tag{4}$$

とする．ここで，v_i は c_i が主ノードの時に 1 でそれ以外なら 0 となる値とする．したがって $v_i = 1-u_i$ と書ける．次に A_{W_k} のコスト $h(W_k)$ を，

$$h(W_k) = \sum_{i=t+1}^{d} p_i l(a_{i-t}) \tag{5}$$

とし，状態 W_k のコスト $e(W_k)$ を，

$$e(W_k) = g(W_k) + h(W_k) \tag{6}$$

$$= \sum_{i=1}^{t} \{u_i p_i l(c_i) + v_i p_i (l(c_i)+s)\} + \sum_{i=t+1}^{d} p_i l(a_{i-t}) \tag{7}$$

とする．つまり状態のコストとは仮の平均符号語長にロスと呼ばれる重みを加えたものであるといえる．このように定義されたコストを計算し上記の手順を繰り返し，d 個の符号語が選択された状態が 1 つ出来たところでその状態での選択された符号語の集まりを T_k として出力する．

図 5 にこれまでに説明した探索アルゴリズムの基本構造を示す．ここで，Algorithm2 中の λ は空列とする．

Algorithm2 の出力は次のような最適性をもつ．

定理 1　情報源アルファベットを $\mathcal{X} = \{1, 2, \ldots, d\}$ とし，それぞれの出現確率を $p_i (p_1 \leq p_2 \leq \cdots \leq p_d)$ とする．2 元 AIFV 符号の符号木 $T_k (k = 0, 1)$ をロスが s の時に Algorithm2 で定まる符号木とする．このとき T_k のコストの値は最小となる．

Algorithm 2 $T_k, (k = 0, 1)$

入力:各情報源シンボルの出現確率 $p_i, (i = 0, 1, \ldots, d), s$
出力:$T_k, (k = 0, 1)$

1:$k = 0$ の時 $C = \phi, A = \{\lambda, 0, 1, 00, \ldots\}$ という状態
をリストの先頭に入れる.
$k = 1$ の時 $C = \phi, A = \{1, 01, 10, 11, \ldots\}$
(= 全通りの文字列から$\lambda, 0, 00$ の子孫を取り除いた集合)
という状態をリストの先頭に入れる.
2:**do**
3: リストの先頭の状態 W_k(始めは 1 で作った状態) を取り出す.
4: 状態 W_k から 3 つの子状態 X_k, Y_k, Z_k を作る.
$k = 0$ の時式 (2) を満たし $|C_{W_k}| < d$ ならば X_0 を消去する.
$k = 1$ の時式 (3) を満たし $|C_{W_k}| < d$ ならば X_1 を消去する.
C_{X_k}(もしくは C_{Y_k}) の要素の中の子孫を持たない主ノードの数
が $d - t$ より多い場合は X_k(もしくは Y_k) を消去する.
消去されなかった子状態のコストを式 (7) を用いて計算する.
5: 消去されなかった状態をリストに入れてコストの小さい
順に並び替える.
6:**while** リストの先頭の状態 W_k で $|C_{W_k}| \neq d$.
7:C_{W_k} の要素を T_k の符号語として出力

図 5 2 元 AIFV 符号の各符号木 $T_k(k = 0, 1)$ を探索するアルゴリズム

(証明) まず，どの状態を見ても子状態のコストは親状態のコストを下回ることは無いことを示す．図 3 を見て，状態 W_k から，子状態 X_k を作るとする．このとき，

$$\begin{aligned} m(W_k) &= \sum_{i=1}^{t} \{u_i p_i l(c_i) + v_i p_i (l(c_i) + s)\} + \sum_{i=t+1}^{d} p_i l(a_{i-t}) \\ &\leq \left(\sum_{i=1}^{t} \{u_i p_i l(c_i) + v_i p_i (l(c_i) + s)\} + p_{t+1} l(a_1) \right) + \sum_{i=t+2}^{d} p_i l(a'_{i-t-1}) \\ &= m(X_k) \end{aligned}$$

となる．このとき 1 行目と 2 行目の式の関係が不等式になるのは，状態 W_k から状態 X_k を構成する時，符号語の候補が更新されるが，符号語の候補は文字列長が短い順で並んでいるため $l(a_1) \leq l(a'_2), l(a_2) \leq l(a'_3), \ldots, l(a_{d-t}) \leq l(a_{d-t-1})$ となるからである．同様に状態 W_k か

ら，子状態 Y_k を作るとすると，

$$\begin{aligned} m(W_k) &= \sum_{i=1}^{t}\{u_i p_i l(c_i) + v_i p_i (l(c_i)+s)\} + \sum_{i=t+1}^{d} p_i l(a_{i-t}) \\ &\leq \left(\sum_{i=1}^{t}\{u_i p_i l(c_i) + v_i p_i (l(c_i)+s)\} + p_{t+1}(l(a_1)+s)\right) + \sum_{i=t+2}^{d} p_i l(a''_{i-t-1}) \\ &= m(Y_k) \end{aligned}$$

となり，状態 W_k から，子状態 Z_k を作るとすると，

$$\begin{aligned} m(W_k) &= \sum_{i=1}^{t}\{u_i p_i l(c_i) + v_i p_i (l(c_i)+s)\} + \sum_{i=t+1}^{d} p_i l(a_{i-t}) \\ &\leq \sum_{i=1}^{t}\{u_i p_i l(c_i) + v_i p_i (l(c_i)+s)\} + \sum_{i=t+1}^{d} p_i l(a_{i-t+1}) \\ &= m(Z_k) \end{aligned}$$

となる．したがってどの状態でも子状態のコストが親状態のコストを下回ることは無い．

Algorithm2 が止まる時は，$d-1$ 個の符号語が選択されている状態 W がすべての状態の中で一番コストが小さいときである．このとき状態 W がコストが一番小さいため W から子状態を作るが，a_1 を葉として選んだ場合この子状態 X は親状態である W と同じコストになる．親状態のコストは子状態のコストを超えることは無いのだから，そのときできた状態 X のコストより低いコストを持つ状態は今後現れないことになる．□

4　3 元 AIFV 符号

前章までに最適な 2 元 AIFV 符号の探索アルゴリズムについて説明してきたが，$K(K \geq 3)$ 元 AIFV 符号についても同じ様にロスの値を更新しながら A^* アルゴリズムを繰り返すことで最適な符号を見つけることが出来る．この章では例として最適な 3 元 AIFV 符号の探索アルゴリズムを説明する．

4.1　3 元 AIFV 符号の定義

この節では 3 元 AIFV 符号の定義とその符号化と復号化の方法を説明する．3 元 AIFV 符号の定義を以下に示す．

定義　（3 元 AIFV 符号）[1]

1. 3 元 AIFV 符号は 2 個の符号木からなる．それらを T_0, T_1 と表す．

2. ひとつだけの子ノードを持つ不完全な内部ノードは子ノードと 0 で繋がっている．
3. 符号木 T_1 のルートは $1, 2$ で繋がる二つの子ノードを持つ．
4. 情報源シンボルは符号木の葉または不完全な内部ノードに割り当てられる．

3 元 AIFV 符号による符号化の手続きは次のようになる．

手続き （3 元 AIFV 符号による符号化）

1. 最初の情報源シンボルは T_0 によって符号化する．
2. 情報源シンボルが葉 (または内部ノード) 符号化されたら，次のシンボルの符号化には T_0(または T_1) を用いるわけではない．

3 元 AIFV 符号の復号化の手続きは次のようになる．

手続き （3 元 AIFV 符号による復号化）

1. 符号化と同様に最初の符号語系列は T_0 によって復号化する．
2. 復号に用いる符号木のルートから観測した符号シンボルの順番に枝をたどり，次の符号シンボルが割り当てられた枝をたどる方法がない場合は，今指しているノードに割り当てられた情報源シンボルを復号する．
3. 符号化と同様に符号語系列が葉 (または内部ノード) によって復号化されたら，次の符号語系列の復号化には T_0 (または T_1) を用いる．

4.2 最適な 3 元 AIFV 符号の探索アルゴリズム

この節では 3 章で説明した 2 元 AIFV 符号の探索アルゴリズムに，前の節で示した 3 元 AIFV 符号の定義を適用することで，最適な 3 元 AIFV 符号の探索アルゴリズムを提案する．

3 元の場合はロスの値として 0.369 が最適値のよい近似である [1]．また，3 元 AIFV 符号では不完全な内部ノードとその子は符号シンボル 0 のみで繋がっているため，図 3 で親状態 W から子状態 Y を作る時，A_{W_k} から a_1 の子孫のうち a_10 を祖先に持たない符号語を消去し A_{Y_k} とする．さらに親から子状態を作る時，作られた子状態が符号語の数が情報源シンボルの数に達しておらず符号語の候補が 0 個となる場合その状態を消去する．その確認には 2 元のときと同じ手法でクラフトの不等式を用いる．まず今の状態で選択されている符号語のみで符号木を作りその符号木から内部ノードに割り当てられたシンボルを消去し，次にその内部ノードが子を持たない場合はその内部ノードとその親ノードに繋がる枝を消去し，さらにその内部ノードが子を持つ場合その内部ノードとその子へ繋がる枝を消去し，その子ノードをルートとする部分木を消去した内部ノードの親ノードに枝でつなげる．次に完成した符号木に対しクラフトの不等式を解く．$k = 0$ のとき (T_0 を構成するとき) は計算したクラフトの式が 1 を超え

ると符号語の候補の数が 0 個になるため，

$$\sum_{i=1}^{t} u_i 3^{-(l(c_i)-w_i)} \geq 1 \tag{8}$$

を満たし $t < d$ ならば X_0 を削除する．また T_1 のルートは 0 でつながる子はないため $k = 1$ のとき (T_1 を構成するとき) はクラフトの式が $\frac{2}{3}$ を超えると符号語の候補が 0 個になるため

$$\sum_{i=1}^{t} u_i 3^{-(l(c_i)-w_i)} \geq \frac{2}{3} \tag{9}$$

を満たし $t < d$ ならば X_1 を消去する．また，不完全な内部ノードは必ず子孫を持つため，C_{X_k}（もしくは C_{Y_k}）の要素の中に現在子孫を持たない不完全な内部ノードの数が $d - t$ より多い場合 X_k(もしくは Y_k) を消去する．また，ロスを更新する式はエンコーダーの状態数によって変わるが，3 元 AIFV 符号では符号木の数が 2 元 AIFV 符号と同様に 2 つであるため，式 (1) を用いることが出来る [1]．まとめると Algorithm3 を用いれば最適な 3 元 AIFV 符号を見つけることが出来る．また Algorithm3 の最適性の証明は，2 元 AIFV 符号の探索アルゴリズムの最適性の証明と同じであるためここでは省略する．

5　最適な AIFV 符号の例

この章では本論文で提案した Algorithm1，Algorithm2，Algorithm3 を用いて見つかった，AIFV 符号の例をいくつか示す．

$P(a) = 0.45$，$P(b) = 0.3$，$P(c) = 0.2$，$P(d) = 0.05$ の分布を持つ情報源 $\mathcal{X} = \{a, b, c, d\}$ に対し最適な 2 元 AIFV 符号を求める．まず Algorithm1 を用いて見つけた AIFV 符号を図 7 に示す．

図 7 の AIFV 符号の平均符号語長を求める．状態遷移確率はそれぞれ $Q(T_1|T_0) = 0.2, Q(T_0|T_1) = 0.25$ となるため，T_0 と T_1 の定常確率はそれぞれ $Q(T_0) = \frac{5}{9}, Q(T_1) = \frac{4}{9}$ となる．また，T_0 と T_1 の平均符号語長はそれぞれ $l_0 = 1.65, l_1 = 1.85$ であるから，図 7 の平均符号語長は $L_{AIFV} = Q(T_0)l_0 + Q(T_1)l_1 = 1.73\dot{8}$ となる．ハフマン符号の平均符号語長は 1.8 であるため，ハフマン符号の平均符号語長を下回ることを確認できた．ちなみに，情報源のエントロピーは約 1.720 である．

2 元 AIFV 符号では分布が極端に偏っている時に T_0 のルートにシンボルが割り当てられることがある．その例を見るために，$P(a) = 0.9$，$P(b) = P(c) = 0.05$ の分布を持つ情報源 $\mathcal{X} = \{a, b, c\}$ に対し最適な 2 元 AIFV 符号を求める．Algorithm1 を用いて見つけた AIFV 符号を図 8 に示した．この符号の平均符号語長を求めると，$L_{AIFV} = Q(T_0)l_0 + Q(T_1)l_1 = \frac{10}{19} \times 0.3 + \frac{9}{19} \times 1.2 \simeq 0.726$ となる．またこの分布のハフマン符号の平均符号語長とエントロピーはそれぞれ 1.1 と約 0.569 である．

Algorithm 3 最適な 3 元 AIFV 符号の探索アルゴリズム

入力:各情報源シンボルの出現確率 $p_i, (i = 0, 1, \ldots, d), s^{(0)} = 0.369$
出力:最適な AIFV 符号 $\boldsymbol{T} = (T_0, T_1)$

1:$m = 0$ とする.
2:**do**
3: $m \longleftarrow m + 1$
4: **for** $k \leftarrow 0, 1$ **do**
5: リストを空にする.
6: $k = 0$ の時 $C = \phi, A = \{\lambda, 0, 1, 2, \ldots\}$ という状態をリストの先頭に入れる.
$k = 1$ の時 $C = \phi, A = \{1, 2, 10, 11, \ldots\}$
(= 全通りの文字列からλと 0 の子孫を取り除いた集合) という状態を
リストの先頭に入れる.
7: **do**
8: リストの先頭の状態 W_k(始めは 1 で作った状態) を取り出す.
9: 状態 W_k から 3 つの子状態 X_k, Y_k, Z_k を作る.
$k = 0$ の時式 (8) を満し $|C_{W_k}| < d$ ならば X_0 を消去する.
$k = 1$ の時式 (9) を満し $|C_{W_k}| < d$ ならば X_1 を消去する.
C_{X_k}(もしくは C_{Y_k}) の要素の中の子孫を持たない不完全な内部ノードの数
が $d - t$ より多い場合は X_k(もしくは Y_k) を消去する.
消去されなかった子状態のコストを式 (7) を用いて計算する.
10: 消去されなかった状態をリストに入れてコストの小さい
順に並び替える.
11: **while** リスト k の先頭の状態 W_k で $|C_{W_k}| \neq d$.
12: C_{W_k} の要素を $T_k^{(m)}$ の符号語とする
13: **end for**
14: 式 (1) を用いて $s^{(m)}$ の値を求める.
15:**while** $s^{(m-1)} \neq s^{(m)}$
16:$T_0^{(m)}, T_1^{(m)}$ を T_0, T_1 として出力.

図 6 最適な 3 元 AIFV 符号を構成するアルゴリズム

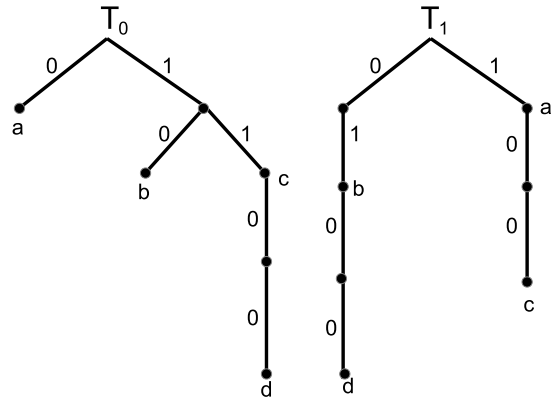

図 7　Algorithm1 を用いて見つけた $\mathcal{X} = \{a, b, c, d\}$ の最適 2 元 AIFV 符号

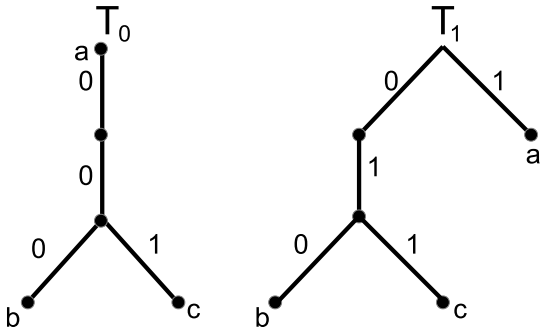

図 8　Algorithm1 を用いて見つけた $\mathcal{X} = \{a, b, c\}$ の最適 2 元 AIFV 符号

次に 3 元 AIFV 符号に対し，$P(a) = P(b) = P(c) = P(d) = P(e) = 0.2$ の分布を持つ情報源 $\mathcal{X} = \{a, b, c, d, e\}$ と，$P(a) = 0.8, P(b) = 0.1, P(c) = P(d) = 0.05$ の分布を持つ情報源 $\mathcal{X} = \{a, b, c, d\}$ について最適な符号を Algorithm3 を用いて見つけそれぞれ図 9,10 に示した．また，各分布をそれぞれ分布 1，分布 2 として，各分布の最適な 3 元 AIFV 符号の平均符号語長とハフマン符号の平均符号語長とエントロピーを表 1 に示す． 表 1 から 3 元 AIFV 符号に対してもどちらの分布でもハフマン符号よりも平均符号語長がよりエントロピーに近づいていることが確認できる．

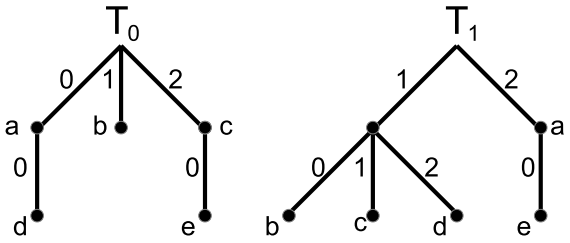

図 9　Algorithm3 を用いて見つけた $\mathcal{X} = \{a, b, c, d, e\}$ の最適 3 元 AIFV 符号

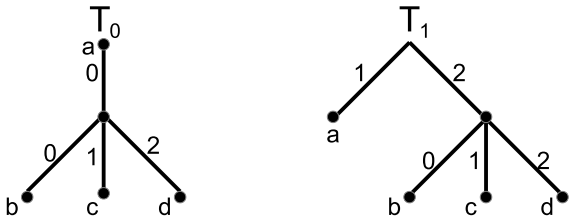

図 10　Algorithm3 を用いて見つけた $\mathcal{X} = \{a, b, c, d\}$ の最適 3 元 AIFV 符号

表 1　3 元 AIFV 符号とハフマン符号とエントロピーの比較

	分布 1	分布 2
3 元 AIFV 符号の平均符号語長	1.533	0.756
ハフマン符号の平均符号語長	1.6	1.1
エントロピー	1.465	0.645

6　最適な AIFV 符号を探索するための計算量

この章では本論文で提案した AIFV 符号の探索アルゴリズムの計算量を考える．Algorithm1 で親が子状態を 3 つ作るという作業の回数と，最大でいくつの状態が存在していたかを，ロスの値を更新しながら数え，その数値をロスの更新回数で割ってグラフに表したものを図 11 に示した．またそのときの情報源分布は一様分布とし，アルファベットサイズを変えながら計測した．さらに同じグラフを 3 元の場合でもつくり図 12 に示した． 図 11,12 では状態の分岐数と状態の最大数が同じグラフに記されているが両者を表す単位は互いに異なっている．また，二つのグラフを見るとどちらも最大状態数が分岐の回数を上回っているが，これは一回分岐をするとひとつの状態から三つの状態に増えるからである．

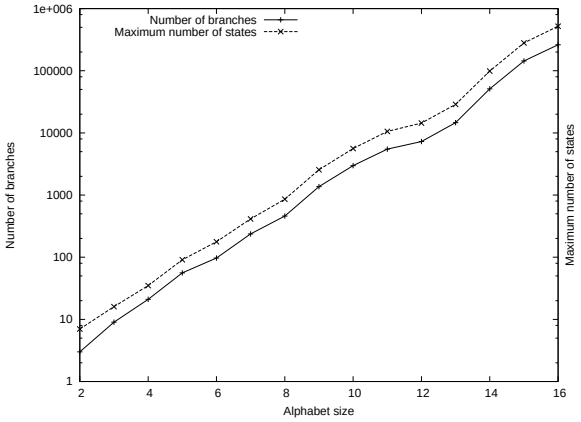

図 11　2 元 AIFV 符号の計算量

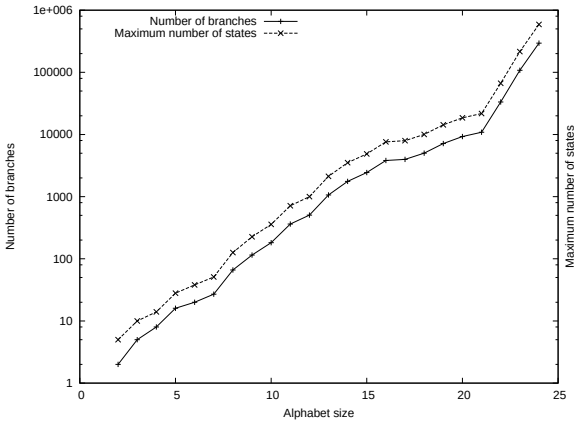

図 12　3 元 AIFV 符号の計算量

図 11,12 から分岐数と状態最大数がどちらもアルファベットサイズに関して指数的に増加しているため本論文で提案した探索アルゴリズムの計算時間と使用メモリは共に指数オーダーであると考えられる．

7　まとめ

本論文では AIFV 符号において平均符号語長を最小にする符号の探索アルゴリズムを提案した．提案したアルゴリズムは，A^* アルゴリズムに新しいパラメータであるロスを付け加え，ロスの値を更新しながら A^* アルゴリズムを繰り返すことで得られたものである．また提案したアルゴリズムの計算量と使用メモリは共に指数オーダーであると考えられる．

本論文では 2 つの符号木からなる最適な AIFV 符号を見つけるアルゴリズムを説明したが，3 つ以上の符号木からなる AIFV 符号についてはロスを更新する式を文献 [4] に掲載されている式に適切に置き換えることで同様のアルゴリズムで見つけることが出来る．

謝辞

本研究を行うにあたって，細かく指導してくださった指導教員の西新幹彦准教授に感謝の意を表する．

参考文献

[1] Hirosuke Yamamoto, Masato Tsuchihashi, and Junya Honda, “Almost Instantaneous Fixed-to-Variable Length Codes,” IEEE Transactions on Infomation Theory, vol.61, no.12, pp.6432–6443, December 2015.

[2] Yuh-Ming, Ting-Yi, Yunghsiang S.Han, “An A^*-Based Algorithm for Constructing Reversible Variable Length Codes with Minimum Average codeword Length,” IEEE Transactions on Communication, vol.58, no.11, pp.3175–3185, November 2010.

[3] Ken-ichi Iwata, Hirosuke Yamamoto, “A Dynamic Programming Algorithm to Construct Optimal Code Tree of AIFV Codes,” ISITA2016, Monterey, California, USA, pp.672–676, 2016.

[4] 河井貴哉，岩田賢一，山本博資，「2 元 AIFV-m 符号の最適な符号木を構成する動的計画法」，信学技報，IT2017-14， pp.79–84，2017 年 5 月．

[5] E.Rich and K.Knight, *Artificial Intelligence*, New York: McGraw-Hill, 1991.

付録 A　最適な AIFV 符号を探索するプログラム

A.1　2 元 AIFV 符号

```
#define _CRT_NONSTDC_NO_DEPRECATE
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>

#define MAX_CODEWORDLEN 32

struct state {
    int fixed;//選択済みの数
    char **codeword;
    char *arraycheck;//葉なら 1、主ノードなら 0
    double cost;
};

int alphabet_size;
double *prob;

void
printstate(struct state *p)
{
    int i;
    for (i = 0; i < p->fixed; i++){
        printf("%s ", (p->codeword[i]) ? p->codeword[i] : "(NULL)");
    }
    printf("|");
    for (; i < alphabet_size; i++){
        printf(" %s", (p->codeword[i]) ? p->codeword[i] : "(NULL)");
    }
    printf("\ncost: %f\n", p->cost);
    return;
}

/*****************************次の文字列を生成*********************************************************************/
char *nextword(char *can, int tree_name)
{
    int i;
NEXT:
    for (i = 0; can[i] == '1'; i++){
        ; //何も入れないのもあり
    }
    if (can[i] == '\0'){
        if (i + 1 >= MAX_CODEWORDLEN){
            printf("error in %d\n", __LINE__);
            exit(1);
        }
        can[i + 1] = '\0'; /* 1 文字長くなる*/
        for (; i >= 0; i--){
            can[i] = '0';
        }
        if (tree_name == 1 && strlen(can) >= 2 && can[0] == '0' && can[1] == '0'){
            goto NEXT;
        }
        return(can);
    }
    for (; can[i] != '\0'; i++){ /* 終端を探す*/
        ;
    }
```

```
    for (i--; can[i] == '1'; i--){
        can[i] = '0';
    }
    can[i] = '1';
    if (tree_name == 1 && strlen(can) >= 2 && can[0] == '0' && can[1] == '0'){
        goto NEXT;
    }
    return(can);
}

int
isprefix(char *word1, char *word2){
    int len1, len2, i;
    len1 = strlen(word1);
    len2 = strlen(word2);
    if (len1 > len2){
        return(0);
    }

    for (i = 0; i < len1; i++){
        if (word1[i] != word2[i]){
            return(0);
        }
    }
    return(1);
}

int
ismasternode(char *word1, char *word2){
    int len1, len2, i;
    len1 = strlen(word1);
    len2 = strlen(word2);

    if (len1 > len2){
        return(1);
    }
    for (i = 0; i < len1; i++){
        if (word1[i] != word2[i]){
            return(1);
        }
    }
    if (len1 + 2 <= len2){
        if (word2[len1] == '0' && word2[len1 + 1] == '0'){
            return(1);
        }
    }
    return(0);
}

void
replace_codeword(struct state state, int tree_name)//仮の符号語の補充
{
    int i, j;
    char can[MAX_CODEWORDLEN];

    for (i = alphabet_size - 1; state.codeword[i] == NULL ; i--){
        ;
    }
    if (i == alphabet_size - 1){
        return;
    }
    strcpy(can, state.codeword[i]);
    for (i++; i < alphabet_size; i++){
    NEXT_CANDIDATE:
        switch (tree_name){
        case 0:
            nextword(can, 0);
            break;
        case 1:
```

```
            nextword(can, 1);
            break;
        default:
            printf("error in %d\n", __LINE__);
            exit(1);
            break;
        }
        for (j = 0; j < state.fixed; j++){//選択済みの符号語と作成した符号語の比較
            switch (state.arraycheck[j]){
            case '0':
                if (!ismasternode(state.codeword[j], can)){
                    goto NEXT_CANDIDATE;
                }
                break;
            case '1':
                if (isprefix(state.codeword[j], can)){
                    goto NEXT_CANDIDATE;
                }
                break;
            default:
                printf("error in %d\n", __LINE__);
                exit(1);
                break;
            }
        }
        state.codeword[i] = strdup(can);
        if (state.codeword[i] == NULL){
            exit(1);
        }
    }

    return;
}

void
confirm_descendant_masternode(struct state *state){//全ての主ノードが子孫を持てるか確認
    int i, j,l, m;
    l = 0;//葉の数
    m = 0;//子孫を持つ主ノードの数
    for (i = 0; i < state->fixed; i++){
        if (state->arraycheck[i] == '1'){
            l++;
            continue;
        }
        if (i == alphabet_size - 1){
            state->fixed += alphabet_size + 1;
            return;
        }
        for (j = i + 1; j < state->fixed; j++){
            if (isprefix(state->codeword[i], state->codeword[j])){
                m++;
                break;
            }
        }
    }
    if (alphabet_size - state->fixed < state->fixed - m - 1){//仮の符号語の数 < 子孫がない主ノードの数
        state->fixed += alphabet_size + 1;
        return;
    }

    return;
}
/*****************************T0 で選択しない場合*****************************************************************/
struct state
removecandidate_t0(struct state *pstate)
{
    struct state nstate;
    int i;
```

```
    if (pstate == NULL){
        printf("[%d]", __LINE__);
        exit(1);
    }
    nstate.codeword = (char **)malloc(sizeof(char*) * alphabet_size);
    if (nstate.codeword == NULL){//エラーチェック
        printf("error in %d\n", __LINE__);
        exit(1);
    }
    nstate.fixed = pstate->fixed;//コピーの作成
    nstate.arraycheck = (char *)malloc(sizeof(char) * nstate.fixed);
    if (nstate.arraycheck == NULL){//エラーチェック
        printf("error in %d\n", __LINE__);
        exit(1);
    }
    for (i = 0; i < nstate.fixed; i++){
        nstate.arraycheck[i] = pstate->arraycheck[i];
    }
    for (i = 0; i < nstate.fixed; i++){
        nstate.codeword[i] = strdup(pstate->codeword[i]);
        if (nstate.codeword[i] == NULL){
            printf("error in %d\n", __LINE__);
            exit(1);
        }
    }
    for (i = nstate.fixed; i < alphabet_size - 1; i++){
        nstate.codeword[i] = strdup(pstate->codeword[i + 1]);
        if (nstate.codeword[i] == NULL){
            printf("error in %d\n", __LINE__);
            exit(1);
        }
    }
    nstate.codeword[alphabet_size - 1] = NULL;
    replace_codeword(nstate, 0);

    return(nstate);
}
/*****************************************************************************************/

/*****************************T0 で葉として選択した場合*************************************************************/
struct state
selected_word_leaf_t0(struct state *pstate)
{
    struct state nstate;
    int i, j, k;
    int len1, len2, power, kraft;
    nstate.codeword = (char **)malloc(sizeof(char*) * alphabet_size);
    if (nstate.codeword == NULL){//エラーチェック
        printf("error in %d\n", __LINE__);
        exit(1);
    }
    nstate.fixed = pstate->fixed + 1;//コピーの作成
    nstate.arraycheck = (char *)malloc(sizeof(char) * nstate.fixed);
    if (nstate.arraycheck == NULL){//エラーチェック
        printf("error in %d\n", __LINE__);
        exit(1);
    }
    for (i = 0; i < nstate.fixed; i++){
        nstate.codeword[i] = strdup(pstate->codeword[i]);
        if (nstate.codeword[i] == NULL){
            printf("error in %d\n", __LINE__);
            exit(1);
        }
    }
    for (; i < alphabet_size; i++){
        nstate.codeword[i] = NULL;
    }
    for (i = 0; i < pstate->fixed; i++){
        nstate.arraycheck[i] = pstate->arraycheck[i];
```

```
    }
    nstate.arraycheck[nstate.fixed - 1] = '1';//葉として選択

    j = nstate.fixed;
    for (i = nstate.fixed; i < alphabet_size; i++){
        if (isprefix(nstate.codeword[nstate.fixed - 1], pstate->codeword[i])){
            continue;
        }
        nstate.codeword[j] = strdup(pstate->codeword[i]);
        if (nstate.codeword[j] == NULL){
            printf("error in %d\n", __LINE__);
            exit(1);
        }
        j++;
    }

    if (nstate.fixed < alphabet_size){//クラフト和
        kraft = 0;
        len1 = strlen(nstate.codeword[nstate.fixed - 1]);//符号語長上界（最大値とは限らない）
        for (i = 0; i < nstate.fixed; i++){
            if (nstate.arraycheck[i] == '0'){
                continue;
            }
            len2 = strlen(nstate.codeword[i]);
            for (k = 0; k < i; k++){
                if (nstate.arraycheck[k] == '0' && isprefix(nstate.codeword[k], nstate.codeword[i])){
                    len2 -= 2;
                }
            }
            power = 1 << (len1 - len2);
            kraft += power;
        }
        if (kraft >= 1 << (len1)){
            nstate.fixed += alphabet_size + 1;
            //free(nstate.codeword);
            return(nstate);
        }
    }
    replace_codeword(nstate, 0);
    confirm_descendant_masternode(&nstate);

    return(nstate);
}
/*********************************************************************************************/

/*****************************T0で主ノードとして選択した場合*************************************************************/
struct state
selected_word_masternode_t0(struct state *pstate)
{
    struct state nstate;
    //char can[MAX_CODEWORDLEN];
    int i, j;
    nstate.codeword = (char **)malloc(sizeof(char*) * alphabet_size);
    if (nstate.codeword == NULL){//エラーチェック
        printf("error in %d\n", __LINE__);
        exit(1);
    }
    nstate.fixed = pstate->fixed + 1;//コピーの作成
    nstate.arraycheck = (char *)malloc(sizeof(char) * nstate.fixed);
    if (nstate.arraycheck == NULL){//エラーチェック
        printf("error in %d\n", __LINE__);
        exit(1);
    }
    for (i = 0; i < nstate.fixed; i++){
        nstate.codeword[i] = strdup(pstate->codeword[i]);
        if (nstate.codeword[i] == NULL){
            printf("error in %d\n", __LINE__);
            exit(1);
        }
```

```
    }
    for (; i < alphabet_size; i++){
        nstate.codeword[i] = NULL;
    }
    for (i = 0; i < pstate->fixed; i++){
        nstate.arraycheck[i] = pstate->arraycheck[i];
    }
    nstate.arraycheck[nstate.fixed - 1] = '0';

    j = nstate.fixed;
    for (i = nstate.fixed; i < alphabet_size; i++){
        if (ismasternode(nstate.codeword[nstate.fixed - 1], pstate->codeword[i])){
            nstate.codeword[j] = strdup(pstate->codeword[i]);
            if (nstate.codeword[j] == NULL){
                printf("error in %d\n", __LINE__);
                exit(1);
            }
            j++;
        }
    }
    replace_codeword(nstate, 0);
    confirm_descendant_masternode(&nstate);

    return(nstate);
}
/*****************************************************************************************/


/*****************************T1 で選択しない場合*****************************************************************/
struct state
removecandidate_t1(struct state *pstate)
{
    struct state nstate;

    int i;

    if (pstate == NULL){
        printf("[%d]", __LINE__);
        exit(1);
    }
    nstate.codeword = (char **)malloc(sizeof(char*) * alphabet_size);
    if (nstate.codeword == NULL){//エラーチェック
        printf("error in %d\n", __LINE__);
        exit(1);
    }
    nstate.fixed = pstate->fixed;//コピーの作成
    nstate.arraycheck = (char *)malloc(sizeof(char) * nstate.fixed);
    if (nstate.arraycheck == NULL){//エラーチェック
        printf("error in %d\n", __LINE__);
        exit(1);
    }
    for (i = 0; i < nstate.fixed; i++){
        nstate.arraycheck[i] = pstate->arraycheck[i];
    }
    for (i = 0; i < nstate.fixed; i++){
        nstate.codeword[i] = strdup(pstate->codeword[i]);
        if (nstate.codeword[i] == NULL){
            printf("error in %d\n", __LINE__);
            exit(1);
        }
    }
    for (i = nstate.fixed; i < alphabet_size - 1; i++){
        nstate.codeword[i] = strdup(pstate->codeword[i + 1]);
        if (nstate.codeword[i] == NULL){
            printf("error in %d\n", __LINE__);
            exit(1);
        }
    }
```

```
    nstate.codeword[alphabet_size - 1] = NULL;
    replace_codeword(nstate, 1);

    return(nstate);
}
/*********************************************************************************************/

/*****************************T1で葉として選択した場合*****************************************************************/
struct state
selected_word_leaf_t1(struct state *pstate)
{
    struct state nstate;
    //char can[MAX_CODEWORDLEN];
    int i, j, k;
    int len1, len2, power, kraft;
    nstate.codeword = (char **)malloc(sizeof(char*) * alphabet_size);
    if (nstate.codeword == NULL){//エラーチェック
        printf("error in %d\n", __LINE__);
        exit(1);
    }
    nstate.fixed = pstate->fixed + 1;//コピーの作成
    nstate.arraycheck = (char *)malloc(sizeof(char) * nstate.fixed);
    if (nstate.arraycheck == NULL){//エラーチェック
        printf("error in %d\n", __LINE__);
        exit(1);
    }
    for (i = 0; i < nstate.fixed; i++){
        nstate.codeword[i] = strdup(pstate->codeword[i]);
        if (nstate.codeword[i] == NULL){
            printf("error in %d\n", __LINE__);
            exit(1);
        }
    }
    for (; i < alphabet_size; i++){
        nstate.codeword[i] = NULL;
    }
    for (i = 0; i < pstate->fixed; i++){
        nstate.arraycheck[i] = pstate->arraycheck[i];
    }
    nstate.arraycheck[nstate.fixed - 1] = '1';

    j = nstate.fixed;
    for (i = nstate.fixed; i < alphabet_size; i++){
        if (isprefix(nstate.codeword[nstate.fixed - 1], pstate->codeword[i])){
            continue;
        }
        nstate.codeword[j] = strdup(pstate->codeword[i]);
        if (nstate.codeword[j] == NULL){
            printf("error in %d\n", __LINE__);
            exit(1);
        }
        j++;
    }
    if (nstate.fixed < alphabet_size){//クラフト和
        kraft = 0;
        len1 = strlen(nstate.codeword[nstate.fixed - 1]);//符号語長上界（最大値とは限らない）
        for (i = 0; i < nstate.fixed; i++){
            if (nstate.arraycheck[i] == '0'){
                continue;
            }
            len2 = strlen(nstate.codeword[i]);
            if (isprefix("0", nstate.codeword[i])){
                len2 -= 1;
            }
            for (k = 0; k < i; k++){
                if (nstate.arraycheck[k] == '0' && isprefix(nstate.codeword[k], nstate.codeword[i])){
                    len2 -= 2;
                }
            }
```

```
            power = 1 << (len1 - len2);
            kraft += power;
        }
        if (kraft >= 1 << (len1)){
            nstate.fixed += alphabet_size + 1;
            return(nstate);
        }
    }
    replace_codeword(nstate, 1);
    confirm_descendant_masternode(&nstate);

    return(nstate);
}
/*****************************************************************************************/

/*****************************T1で主ノードとして選択した場合*****************************************************************/
struct state
selected_word_masternode_t1(struct state *pstate)
{
    struct state nstate;
    int i, j;
    nstate.codeword = (char **)malloc(sizeof(char*) * alphabet_size);
    if (nstate.codeword == NULL){//エラーチェック
        printf("error in %d\n", __LINE__);
        exit(1);
    }
    nstate.fixed = pstate->fixed + 1;//コピーの作成
    nstate.arraycheck = (char *)malloc(sizeof(char) * nstate.fixed);
    if (nstate.arraycheck == NULL){//エラーチェック
        printf("error in %d\n", __LINE__);
        exit(1);
    }
    for (i = 0; i < nstate.fixed; i++){
        nstate.codeword[i] = strdup(pstate->codeword[i]);
        if (nstate.codeword[i] == NULL){
            printf("error in %d\n", __LINE__);
            exit(1);
        }
    }
    for (; i < alphabet_size; i++){
        nstate.codeword[i] = NULL;
    }
    for (i = 0; i < pstate->fixed; i++){
        nstate.arraycheck[i] = pstate->arraycheck[i];
    }
    nstate.arraycheck[nstate.fixed - 1] = '0';

    j = nstate.fixed;
    for (i = nstate.fixed; i < alphabet_size; i++){
        if (ismasternode(nstate.codeword[nstate.fixed - 1], pstate->codeword[i])){
            nstate.codeword[j] = strdup(pstate->codeword[i]);
            if (nstate.codeword[j] == NULL){
                printf("error in %d\n", __LINE__);
                exit(1);
            }
            j++;
        }
    }
    replace_codeword(nstate, 1);
    confirm_descendant_masternode(&nstate);

    return(nstate);
}
/*****************************************************************************************/

double c_cost(struct state *pstate, double loss){
    int i;
    double cost;
```

```
    cost = 0;
    for (i = 0; i < alphabet_size; i++){
        cost += strlen(pstate->codeword[i]) * prob[i];
    }
    for (i = 0; i < pstate->fixed; i++){
        if (pstate->arraycheck[i] == '0'){
            cost += loss * prob[i];
        }
    }
    return(cost);
}

/*****************************************************************************************/
void
initlist_t0(struct state *p)//T0 用の初期設定
{
    int i;
    char can[MAX_CODEWORDLEN];

    p->fixed = 0;//一番最初の state に入力していく↓
    p->codeword = (char **)calloc(alphabet_size, sizeof(char *));
    if (p->codeword == NULL){
        exit(EXIT_FAILURE);
    }
    p->arraycheck = (char*)calloc(p->fixed, sizeof(char));
    if (p->codeword == NULL){
        exit(EXIT_FAILURE);
    }
    can[0] = '\0';
    p->codeword[0] = strdup(can);
    for (i = 1; i < alphabet_size; i++){
        nextword(can, 0);
        p->codeword[i] = strdup(can);
        if (p->codeword[i] == NULL){
            printf("error in %d\n", __LINE__);
            exit(1);
        }
    }
    return;
}

void
initlist_t1(struct state *p)//T1 用の初期設定
{
    int i;
    char can[MAX_CODEWORDLEN];

    p->fixed = 0;//一番最初の state に入力していく↓
    p->codeword = (char **)calloc(alphabet_size, sizeof(char *));
    if (p->codeword == NULL){
        exit(EXIT_FAILURE);
    }
    p->arraycheck = (char*)calloc(p->fixed, sizeof(char));
    if (p->codeword == NULL){
        exit(EXIT_FAILURE);
    }
    can[0] = '\0';
    nextword(can, 1);
    for (i = 0; i < alphabet_size; i++){
        nextword(can, 1);
        p->codeword[i] = strdup(can);
        if (p->codeword[i] == NULL){
            printf("error in %d\n", __LINE__);
            exit(1);
        }
    }
    return;
}
```

```
double Ubound;
#define LISTBOUND 1000000
struct state list[LISTBOUND];
struct state state, state1, state2_t0, state2_t1, state3;//state1,state2,state3 は子ノード

void
state_free(struct state state)
{
    int i;

    for (i = 0; i < alphabet_size; i++){
        free(state.codeword[i]);
    }
    free(state.codeword);
    free(state.arraycheck);
    return;
}

void
find_t0(double loss)
{
    int num_state;//state の数
    int i, k;
    initlist_t0(list);
    list[0].cost = c_cost(list, loss);
    num_state = 1;
    while (list[num_state - 1].fixed < alphabet_size){
        state = list[num_state - 1];
        num_state--;
        state1 = removecandidate_t0(&state);
        if (state1.fixed <= alphabet_size){
            if (num_state >= LISTBOUND){
                printf("error in %d\n", __LINE__);
                exit(1);
            }
            state1.cost = c_cost(&state1, loss);
            for (i = num_state; i > 0; i--){
                if (state1.cost <= list[i - 1].cost){
                    break;
                }
                list[i] = list[i - 1];
            }
            list[i] = state1;
            num_state++;
        } else {
            state_free(state1);
        }

        state2_t0 = selected_word_leaf_t0(&state);
        if (state2_t0.fixed == alphabet_size){
            state2_t0.cost = c_cost(&state2_t0, loss);
            return;
        }
        if (state2_t0.fixed < alphabet_size){
            if (num_state >= LISTBOUND){
                printf("error in %d\n", __LINE__);
                exit(1);
            }
            state2_t0.cost = c_cost(&state2_t0, loss);
            for (i = num_state; i > 0; i--){
                if (state2_t0.cost <= list[i - 1].cost){
                    break;
                }
                list[i] = list[i - 1];
            }
            list[i] = state2_t0;
            num_state++;
        } else {
            state_free(state2_t0);
```

```
        }

        state3 = selected_word_masternode_t0(&state);
        if (state3.fixed <= alphabet_size){
            if (num_state >= LISTBOUND){
                printf("error in %d\n", __LINE__);
                exit(1);
            }
            state3.cost = c_cost(&state3, loss);
            for (i = num_state; i > 0; i--){
                if (state3.cost <= list[i - 1].cost){
                    break;
                }
                list[i] = list[i - 1];
            }
            list[i] = state3;
            num_state++;
        } else {
            state_free(state3);
        }

        state_free(state);

        for (k = 0; k < num_state; k++){//Ubound と比較して大きいものは消去
            if (list[k].cost <= Ubound){
                break;
            }
        }
        if (k == num_state){
            printf("error in %d\n", __LINE__);
            exit(1);
        }
        if (k > 0){
            for (i = 0; i < k; i++){
                state_free(list[i]);
                list[i] = list[k + i];
            }
            for (num_state -= k; i < num_state; i++){
                list[i] = list[k + i];
            }
        }
    }
}

void
find_t1(double loss)
{
    int num_state;//state の数
    int i;
    initlist_t1(list);
    list[0].cost = c_cost(list, loss);
    num_state = 1;
    while (list[num_state - 1].fixed < alphabet_size){
        state = list[num_state - 1];
        num_state--;
        state1 = removecandidate_t1(&state);
        if (state1.fixed <= alphabet_size){
            if (num_state >= LISTBOUND){
                printf("error in %d\n", __LINE__);
                exit(1);
            }
            state1.cost = c_cost(&state1, loss);
            for (i = num_state; i > 0; i--){
                if (state1.cost <= list[i - 1].cost){
                    break;
                }
                list[i] = list[i - 1];
            }
            list[i] = state1;
```

```
            num_state++;
        } else {
            state_free(state1);
        }

        state2_t1 = selected_word_leaf_t1(&state);
        if (state2_t1.fixed == alphabet_size){
            state2_t1.cost = c_cost(&state2_t1, loss);
            return;
        }
        if (state2_t1.fixed < alphabet_size){
            if (num_state >= LISTBOUND){
                printf("error in %d\n", __LINE__);
                exit(1);
            }
            state2_t1.cost = c_cost(&state2_t1, loss);
            for (i = num_state; i > 0; i--){
                if (state2_t1.cost <= list[i - 1].cost){
                    break;
                }
                list[i] = list[i - 1];
            }
            list[i] = state2_t1;
            num_state++;
        } else {
            state_free(state2_t1);
        }

        state3 = selected_word_masternode_t1(&state);
        if (state3.fixed <= alphabet_size){
            if (num_state >= LISTBOUND){
                printf("error in %d\n", __LINE__);
                exit(1);
            }
            state3.cost = c_cost(&state3, loss);
            for (i = num_state; i > 0; i--){
                if (state3.cost <= list[i - 1].cost){
                    break;
                }
                list[i] = list[i - 1];
            }
            list[i] = state3;
            num_state++;
        } else {
            state_free(state3);
        }

        state_free(state);
    }
}
main(){
    int i, distribution;
    double loss, nextloss, l0, l1, l, q0, q1;//q0, q1 は Q(T1|T0), Q(T0|T1)
    loss = 0.405;

    printf("アルファベットサイズ=");
    scanf("%d", &alphabet_size);
    if (alphabet_size > 100){
        puts("要素数オーバー");
        return(0);
    }
    prob = (double*)calloc(alphabet_size, sizeof(double));
    for (i = 0; i<alphabet_size; i++){
        printf("確率 %d を入力", i + 1);
        scanf("%lf", &prob[i]);
        if (prob[i] > 1){
            printf("確率は 1 以下");
            printf("\n");
            return(0);
```

```
        }
        if (i > 0){
            if (prob[i] > prob[i - 1]){
                printf("確率は降順で");
                printf("\n");
                return(0);
            }
        }
    }
    for (i = 0; i < alphabet_size; i++){
            printf("確率 %d=%f\n", i, prob[i]);
        }
    Ubound = 0;
    for (i = 1; i <= alphabet_size; i++){
        Ubound += prob[i - 1] * i;
    }
FIND_AIFV:
    l0 = 0;
    l1 = 0;
    q0 = 0;
    q1 = 0;
    find_t0(loss);
    for (i = 0; i < alphabet_size; i++){
        l0 += prob[i] * strlen(state2_t0.codeword[i]);
        if (state2_t0.arraycheck[i] == '0'){
            q0 += prob[i];
        }
    }
    find_t1(loss);
    for (i = 0; i < alphabet_size; i++){
        l1 += prob[i] * strlen(state2_t1.codeword[i]);
        if (state2_t1.arraycheck[i] == '1'){
            q1 += prob[i];
        }
    }
    nextloss = (l1 - l0) / (q0 + q1);
    l = (l0 * (q1 / (q0 + q1))) + (l1 * (q0 / (q0 + q1)));
    if (loss != nextloss){
        state_free(state2_t0);
        state_free(state2_t1);
        loss = nextloss;
        goto FIND_AIFV;
    }

    printf("T0\n");
    printstate(&state2_t0);
    printf("T1\n");
    printstate(&state2_t1);
    l = (l0 * (q1 / (q0 + q1))) + (l1 * (q0 / (q0 + q1)));
    printf("Q(T0)=%f Q(T1)=%f 平均符号語長=%f\n", (q1 / (q0 + q1)), (q0 / (q0 + q1)), l);

    return(0);
}
```

A.2　3 元 AIFV 符号

```
#define _CRT_NONSTDC_NO_DEPRECATE
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>
```

```
#define MAX_CODEWORDLEN 32

struct state {
    int fixed;//選択済みの数
    char **codeword;
    char *arraycheck;//葉なら 1、内部ノードなら 0
    double cost;
};

int alphabet_size;
double *prob;

void
printstate(struct state *p)
{
    int i;
    for (i = 0; i < p->fixed; i++){
        printf("%s ", (p->codeword[i]) ? p->codeword[i] : "(NULL)");
    }
    printf("|");
    for (; i < alphabet_size; i++){
        printf(" %s", (p->codeword[i]) ? p->codeword[i] : "(NULL)");
    }
    printf("\ncost: %f\n", p->cost);
    return;
}

/*****************************次の文字列を生成*********************************************************************/

char *nextword(char *can, int tree_name)
{
    int i;

    if (tree_name != 0 && tree_name != 1){
        printf("error in %d\n", __LINE__);
        exit(1);
    }
NEXT:
    for (i = 0; can[i] == '2'; i++){
        ;
    }
    if (can[i] == '\0'){
        if (i + 1 >= MAX_CODEWORDLEN){
            printf("error in %d\n", __LINE__);
            exit(1);
        }
        can[i + 1] = '\0'; // 1 文字長くなる
        for (; i >= 0; i--){
            can[i] = '0';
        }
        if (tree_name == 1 && strlen(can) >= 1 && can[0] == '0'){
            goto NEXT;
        }
        return(can);
    }
    for (; can[i] != '\0'; i++){//終端を探す
        ;
    }
    for (i--; can[i] == '2'; i--){
        can[i] = '0';
    }
    if (can[i] == '1'){
        can[i] = '2';
    }
    if (can[i] == '0'){
        can[i] = '1';
    }
    if (tree_name == 1 && strlen(can) >= 1 && can[0] == '0'){
```

```
        goto NEXT;
    }
    return(can);
}

int
isprefix(char *word1, char *word2){
    int len1, len2, i;
    len1 = strlen(word1);
    len2 = strlen(word2);
    if (len1 > len2){
        return(0);
    }

    for (i = 0; i < len1; i++){
        if (word1[i] != word2[i]){
            return(0);
        }
    }
    return(1);
}

int
ismasternode(char *word1, char *word2){
    int len1, len2, i;
    len1 = strlen(word1);
    len2 = strlen(word2);

    if (len1 > len2){
        return(1);
    }
    for (i = 0; i < len1; i++){
        if (word1[i] != word2[i]){
            return(1);
        }
    }
    if (len1 + 1 <= len2){
        if (word2[len1] == '0'){
            return(1);
        }
    }
    return(0);
}

void
replace_codeword(struct state state, int tree_name)//仮の符号語の補充
{
    int i, j;
    char can[MAX_CODEWORDLEN];

    if (tree_name != 0 && tree_name != 1){
        printf("error in %d\n", __LINE__);
        exit(1);
    }
    for (i = alphabet_size - 1; state.codeword[i] == NULL; i--){
        ;
    }
    if (i == alphabet_size - 1){
        return;
    }
    strcpy(can, state.codeword[i]);
    for (i++; i < alphabet_size; i++){
    NEXT_CANDIDATE:
        /*
        switch (tree_name){
        case 0:
            nextword(can, 0);
            break;
        case 1:
```

```
            nextword(can, 1);
            break;
        default:
            printf("error in %d\n", __LINE__);
            exit(1);
            break;
        }
        */
        nextword(can, tree_name);
        for (j = 0; j < state.fixed; j++){//選択済みの符号語と作成した符号語の比較
            switch (state.arraycheck[j]){
            case '0':
                if (!ismasternode(state.codeword[j], can)){
                    goto NEXT_CANDIDATE;
                }
                break;
            case '1':
                if (isprefix(state.codeword[j], can)){
                    goto NEXT_CANDIDATE;
                }
                break;
            default:
                printf("error in %d\n", __LINE__);
                exit(1);
                break;
            }
        }
        state.codeword[i] = strdup(can);
        if (state.codeword[i] == NULL){
            exit(1);
        }
    }

    return;
}

void
confirm_descendant_masternode(struct state *state){//全ての不完全内部ノードが子孫を持てるか確認
    int i, j, l, m;
    l = 0;//葉の数
    m = 0;//子孫を持つ不完全内部ノードの数
    for (i = 0; i < state->fixed; i++){
        if (state->arraycheck[i] == '1'){
            l++;
            continue;
        }
        if (i == alphabet_size - 1){
            state->fixed += alphabet_size + 1;
            return;
        }
        for (j = i + 1; j < state->fixed; j++){
            if (isprefix(state->codeword[i], state->codeword[j])){
                m++;
                break;
            }
        }
    }
    if (alphabet_size - state->fixed < state->fixed - m - 1){//仮の符号語の数 < 子孫がない不完全内部ノードの数
        state->fixed += alphabet_size + 1;
        return;
    }

    return;
}

/*****************************T0で選択しない場合*********************************************************************/
struct state
    removecandidate_t0(struct state *pstate)
{
```

```
    struct state nstate;
    int i;

    if (pstate == NULL){
        printf("[%d]", __LINE__);
        exit(1);
    }
    nstate.codeword = (char **)malloc(sizeof(char*) * alphabet_size);
    if (nstate.codeword == NULL){//エラーチェック
        printf("error in %d\n", __LINE__);
        exit(1);
    }
    nstate.fixed = pstate->fixed;//コピーの作成
    nstate.arraycheck = (char *)malloc(sizeof(char) * nstate.fixed);
    if (nstate.arraycheck == NULL){//エラーチェック
        printf("error in %d\n", __LINE__);
        exit(1);
    }
    for (i = 0; i < nstate.fixed; i++){
        nstate.arraycheck[i] = pstate->arraycheck[i];
    }
    for (i = 0; i < nstate.fixed; i++){
        nstate.codeword[i] = strdup(pstate->codeword[i]);
        if (nstate.codeword[i] == NULL){
            printf("error in %d\n", __LINE__);
            exit(1);
        }
    }
    for (i = nstate.fixed; i < alphabet_size - 1; i++){
        nstate.codeword[i] = strdup(pstate->codeword[i + 1]);
        if (nstate.codeword[i] == NULL){
            printf("error in %d\n", __LINE__);
            exit(1);
        }
    }
    nstate.codeword[alphabet_size - 1] = NULL;
    replace_codeword(nstate, 0);

    return(nstate);
}
/*****************************************************************************************/

/*****************************T0で葉として選択した場合*************************************************************/
struct state
    selected_word_leaf_t0(struct state *pstate)
{
    struct state nstate;
    int i, j, k;
    int len1, len2, power, kraft_l, kraft_r;
    nstate.codeword = (char **)malloc(sizeof(char*) * alphabet_size);
    if (nstate.codeword == NULL){//エラーチェック
        printf("error in %d\n", __LINE__);
        exit(1);
    }
    nstate.fixed = pstate->fixed + 1;//コピーの作成
    nstate.arraycheck = (char *)malloc(sizeof(char) * nstate.fixed);
    if (nstate.arraycheck == NULL){//エラーチェック
        printf("error in %d\n", __LINE__);
        exit(1);
    }
    for (i = 0; i < nstate.fixed; i++){
        nstate.codeword[i] = strdup(pstate->codeword[i]);
        if (nstate.codeword[i] == NULL){
            printf("error in %d\n", __LINE__);
            exit(1);
        }
    }
    for (; i < alphabet_size; i++){
        nstate.codeword[i] = NULL;
```

```
    }
    for (i = 0; i < pstate->fixed; i++){
        nstate.arraycheck[i] = pstate->arraycheck[i];
    }
    nstate.arraycheck[nstate.fixed - 1] = '1';//葉として選択

    j = nstate.fixed;
    for (i = nstate.fixed; i < alphabet_size; i++){
        if (isprefix(nstate.codeword[nstate.fixed - 1], pstate->codeword[i])){
            continue;
        }
        nstate.codeword[j] = strdup(pstate->codeword[i]);
        if (nstate.codeword[j] == NULL){
            printf("error in %d\n", __LINE__);
            exit(1);
        }
        j++;
    }

    if (nstate.fixed < alphabet_size){//クラフト和
        kraft_l = 1;
        len1 = strlen(nstate.codeword[nstate.fixed - 1]);//符号語長上界（最大値とは限らない）
        for (i = 0; i < nstate.fixed; i++){
            if (nstate.arraycheck[i] == '0'){
                continue;
            }
            len2 = strlen(nstate.codeword[i]);
            for (k = 0; k < i; k++){
                if (nstate.arraycheck[k] == '0' && isprefix(nstate.codeword[k], nstate.codeword[i])){
                    len2 -= 1;
                }
            }
            power = 1;
            for (j = 0; j <= len1 - len2; j++){
                power *= 3;
            }
            kraft_l += power;
        }
        kraft_r = 1;
        for (j = 0; j <= len1; j++){
            kraft_r *= 3;
        }
        if (kraft_l >= kraft_r){
            nstate.fixed += alphabet_size + 1;
            //free(nstate.codeword);
            return(nstate);
        }
    }
    replace_codeword(nstate, 0);
    confirm_descendant_masternode(&nstate);

    return(nstate);
}
/*********************************************************************************************/

/*****************************T0で不完全内部ノードとして選択した場合*************************************************************
struct state
    selected_word_masternode_t0(struct state *pstate)
{
    struct state nstate;
    //char can[MAX_CODEWORDLEN];
    int i, j;
    nstate.codeword = (char **)malloc(sizeof(char*) * alphabet_size);
    if (nstate.codeword == NULL){//エラーチェック
        printf("error in %d\n", __LINE__);
        exit(1);
    }
    nstate.fixed = pstate->fixed + 1;//コピーの作成
    nstate.arraycheck = (char *)malloc(sizeof(char) * nstate.fixed);
```

```
    if (nstate.arraycheck == NULL){//エラーチェック
        printf("error in %d\n", __LINE__);
        exit(1);
    }
    for (i = 0; i < nstate.fixed; i++){
        nstate.codeword[i] = strdup(pstate->codeword[i]);
        if (nstate.codeword[i] == NULL){
            printf("error in %d\n", __LINE__);
            exit(1);
        }
    }
    for (; i < alphabet_size; i++){
        nstate.codeword[i] = NULL;
    }
    for (i = 0; i < pstate->fixed; i++){
        nstate.arraycheck[i] = pstate->arraycheck[i];
    }
    nstate.arraycheck[nstate.fixed - 1] = '0';

    j = nstate.fixed;
    for (i = nstate.fixed; i < alphabet_size; i++){
        if (ismasternode(nstate.codeword[nstate.fixed - 1], pstate->codeword[i])){
            nstate.codeword[j] = strdup(pstate->codeword[i]);
            if (nstate.codeword[j] == NULL){
                printf("error in %d\n", __LINE__);
                exit(1);
            }
            j++;
        }
    }
    replace_codeword(nstate, 0);
    confirm_descendant_masternode(&nstate);

    return(nstate);
}
/*****************************************************************************************/

/*****************************T1で選択しない場合*****************************************************************/
struct state
    removecandidate_t1(struct state *pstate)
{
    struct state nstate;

    int i;

    if (pstate == NULL){
        printf("[%d]", __LINE__);
        exit(1);
    }
    nstate.codeword = (char **)malloc(sizeof(char*) * alphabet_size);
    if (nstate.codeword == NULL){//エラーチェック
        printf("error in %d\n", __LINE__);
        exit(1);
    }
    nstate.fixed = pstate->fixed;//コピーの作成
    nstate.arraycheck = (char *)malloc(sizeof(char) * nstate.fixed);
    if (nstate.arraycheck == NULL){//エラーチェック
        printf("error in %d\n", __LINE__);
        exit(1);
    }
    for (i = 0; i < nstate.fixed; i++){
        nstate.arraycheck[i] = pstate->arraycheck[i];
    }
    for (i = 0; i < nstate.fixed; i++){
        nstate.codeword[i] = strdup(pstate->codeword[i]);
        if (nstate.codeword[i] == NULL){
            printf("error in %d\n", __LINE__);
            exit(1);
        }
```

```
    }
    for (i = nstate.fixed; i < alphabet_size - 1; i++){
        nstate.codeword[i] = strdup(pstate->codeword[i + 1]);
        if (nstate.codeword[i] == NULL){
            printf("error in %d\n", __LINE__);
            exit(1);
        }
    }

    nstate.codeword[alphabet_size - 1] = NULL;
    replace_codeword(nstate, 1);

    return(nstate);
}
/*****************************************************************************************/

/*****************************T1で葉として選択した場合*************************************************************/
struct state
    selected_word_leaf_t1(struct state *pstate)
{
    struct state nstate;
    //char can[MAX_CODEWORDLEN];
    int i, j, k;
    int len1, len2, power, kraft_l, kraft_r;
    nstate.codeword = (char **)malloc(sizeof(char*) * alphabet_size);
    if (nstate.codeword == NULL){//エラーチェック
        printf("error in %d\n", __LINE__);
        exit(1);
    }
    nstate.fixed = pstate->fixed + 1;//コピーの作成
    nstate.arraycheck = (char *)malloc(sizeof(char) * nstate.fixed);
    if (nstate.arraycheck == NULL){//エラーチェック
        printf("error in %d\n", __LINE__);
        exit(1);
    }
    for (i = 0; i < nstate.fixed; i++){
        nstate.codeword[i] = strdup(pstate->codeword[i]);
        if (nstate.codeword[i] == NULL){
            printf("error in %d\n", __LINE__);
            exit(1);
        }
    }
    for (; i < alphabet_size; i++){
        nstate.codeword[i] = NULL;
    }
    for (i = 0; i < pstate->fixed; i++){
        nstate.arraycheck[i] = pstate->arraycheck[i];
    }
    nstate.arraycheck[nstate.fixed - 1] = '1';

    j = nstate.fixed;
    for (i = nstate.fixed; i < alphabet_size; i++){
        if (isprefix(nstate.codeword[nstate.fixed - 1], pstate->codeword[i])){
            continue;
        }
        nstate.codeword[j] = strdup(pstate->codeword[i]);
        if (nstate.codeword[j] == NULL){
            printf("error in %d\n", __LINE__);
            exit(1);
        }
        j++;
    }
    if (nstate.fixed < alphabet_size){//クラフト和
        kraft_l = 0;
        len1 = strlen(nstate.codeword[nstate.fixed - 1]);//符号語長上界（最大値とは限らない）
        for (i = 0; i < nstate.fixed; i++){
            if (nstate.arraycheck[i] == '0'){
                continue;
            }
```

```
            len2 = strlen(nstate.codeword[i]);
            for (k = 0; k < i; k++){
                if (nstate.arraycheck[k] == '0' && isprefix(nstate.codeword[k], nstate.codeword[i])){
                    len2 -= 1;
                }
            }
            power = 1;
            for (j = 0; j <= len1 - len2; j++){
                power *= 3;
            }
            kraft_l += power;
        }
        kraft_r = 1;
        for (j = 0; j <= len1; j++){
            kraft_r *= 3;
        }
        if (3 * kraft_l >= 2 * kraft_r){
            nstate.fixed += alphabet_size + 1;
            return(nstate);
        }
    }
    replace_codeword(nstate, 1);
    confirm_descendant_masternode(&nstate);

    return(nstate);
}
/*********************************************************************************************/

/*****************************T1で不完全内部ノードとして選択した場合*************************************************************
struct state
    selected_word_masternode_t1(struct state *pstate)
{
    struct state nstate;
    int i, j;
    nstate.codeword = (char **)malloc(sizeof(char*) * alphabet_size);
    if (nstate.codeword == NULL){//エラーチェック
        printf("error in %d\n", __LINE__);
        exit(1);
    }
    nstate.fixed = pstate->fixed + 1;//コピーの作成
    nstate.arraycheck = (char *)malloc(sizeof(char) * nstate.fixed);
    if (nstate.arraycheck == NULL){//エラーチェック
        printf("error in %d\n", __LINE__);
        exit(1);
    }
    for (i = 0; i < nstate.fixed; i++){
        nstate.codeword[i] = strdup(pstate->codeword[i]);
        if (nstate.codeword[i] == NULL){
            printf("error in %d\n", __LINE__);
            exit(1);
        }
    }
    for (; i < alphabet_size; i++){
        nstate.codeword[i] = NULL;
    }
    for (i = 0; i < pstate->fixed; i++){
        nstate.arraycheck[i] = pstate->arraycheck[i];
    }
    nstate.arraycheck[nstate.fixed - 1] = '0';

    j = nstate.fixed;
    for (i = nstate.fixed; i < alphabet_size; i++){
        if (ismasternode(nstate.codeword[nstate.fixed - 1], pstate->codeword[i])){
            nstate.codeword[j] = strdup(pstate->codeword[i]);
            if (nstate.codeword[j] == NULL){
                printf("error in %d\n", __LINE__);
                exit(1);
            }
            j++;
```

```
        }
    }
    replace_codeword(nstate, 1);
    confirm_descendant_masternode(&nstate);

    return(nstate);
}
/*****************************************************************************************/

double c_cost(struct state *pstate, double loss){
    int i;
    double cost;

    cost = 0;
    for (i = 0; i < alphabet_size; i++){
        cost += strlen(pstate->codeword[i]) * prob[i];
    }
    for (i = 0; i < pstate->fixed; i++){
        if (pstate->arraycheck[i] == '0'){
            cost += loss * prob[i];
        }
    }
    return(cost);
}

/*****************************************************************************************/
void
initlist_t0(struct state *p)//T0 用の初期設定
{
    int i;
    char can[MAX_CODEWORDLEN];

    p->fixed = 0;//一番最初の state に入力していく↓
    p->codeword = (char **)calloc(alphabet_size, sizeof(char *));
    if (p->codeword == NULL){
        exit(EXIT_FAILURE);
    }
    p->arraycheck = (char*)calloc(p->fixed, sizeof(char));
    if (p->codeword == NULL){
        exit(EXIT_FAILURE);
    }
    can[0] = '\0';
    p->codeword[0] = strdup(can);
    for (i = 1; i < alphabet_size; i++){
        nextword(can, 0);
        p->codeword[i] = strdup(can);
        if (p->codeword[i] == NULL){
            printf("error in %d\n", __LINE__);
            exit(1);
        }
    }
    return;
}

void
initlist_t1(struct state *p)//T1 用の初期設定
{
    int i;
    char can[MAX_CODEWORDLEN];

    p->fixed = 0;//一番最初の state に入力していく↓
    p->codeword = (char **)calloc(alphabet_size, sizeof(char *));
    if (p->codeword == NULL){
        exit(EXIT_FAILURE);
    }
    p->arraycheck = (char*)calloc(p->fixed, sizeof(char));
    if (p->codeword == NULL){
        exit(EXIT_FAILURE);
    }
```

```
    can[0] = '\0';
    for (i = 0; i < alphabet_size; i++){
        nextword(can, 1);
        p->codeword[i] = strdup(can);
        if (p->codeword[i] == NULL){
            printf("error in %d\n", __LINE__);
            exit(1);
        }
    }
    return;
}

double Ubound;
#define LISTBOUND 1000000
struct state list[LISTBOUND];
struct state state, state1, state2_t0, state2_t1, state3;//state1,state2,state3 は子ノード

void
state_free(struct state state)
{
    int i;

    for (i = 0; i < alphabet_size; i++){
        free(state.codeword[i]);
    }
    free(state.codeword);
    free(state.arraycheck);
    return;
}
void
find_t0(double loss)
{
    int num_state;//state の数
    int i;
    initlist_t0(list);
    list[0].cost = c_cost(list, loss);
    num_state = 1;
    while (list[num_state - 1].fixed < alphabet_size){
        state = list[num_state - 1];
        num_state--;
        state1 = removecandidate_t0(&state);
        if (state1.fixed <= alphabet_size){
            if (num_state >= LISTBOUND){
                printf("error in %d\n", __LINE__);
                exit(1);
            }
            state1.cost = c_cost(&state1, loss);
            for (i = num_state; i > 0; i--){
                if (state1.cost <= list[i - 1].cost){
                    break;
                }
                list[i] = list[i - 1];
            }
            list[i] = state1;
            num_state++;
        }
        else {
            state_free(state1);
        }

        state2_t0 = selected_word_leaf_t0(&state);
        if (state2_t0.fixed == alphabet_size){
            state2_t0.cost = c_cost(&state2_t0, loss);
            return;
        }
        if (state2_t0.fixed < alphabet_size){
            if (num_state >= LISTBOUND){
                printf("error in %d\n", __LINE__);
                exit(1);
```

```
            }
            state2_t0.cost = c_cost(&state2_t0, loss);
            for (i = num_state; i > 0; i--){
                if (state2_t0.cost <= list[i - 1].cost){
                    break;
                }
                list[i] = list[i - 1];
            }
            list[i] = state2_t0;
            num_state++;
        }
        else {
            state_free(state2_t0);
        }

        state3 = selected_word_masternode_t0(&state);
        if (state3.fixed <= alphabet_size){
            if (num_state >= LISTBOUND){
                printf("error in %d\n", __LINE__);
                exit(1);
            }
            state3.cost = c_cost(&state3, loss);
            for (i = num_state; i > 0; i--){
                if (state3.cost <= list[i - 1].cost){
                    break;
                }
                list[i] = list[i - 1];
            }
            list[i] = state3;
            num_state++;
        }
        else {
            state_free(state3);
        }

        state_free(state);
    }
}

void
find_t1(double loss)
{
    int num_state;//state の数
    int i;
    initlist_t1(list);
    list[0].cost = c_cost(list, loss);
    num_state = 1;
    while (list[num_state - 1].fixed < alphabet_size){
        state = list[num_state - 1];
        num_state--;
        state1 = removecandidate_t1(&state);
        if (state1.fixed <= alphabet_size){
            if (num_state >= LISTBOUND){
                printf("error in %d\n", __LINE__);
                exit(1);
            }
            state1.cost = c_cost(&state1, loss);
            for (i = num_state; i > 0; i--){
                if (state1.cost <= list[i - 1].cost){
                    break;
                }
                list[i] = list[i - 1];
            }
            list[i] = state1;
            num_state++;
        }
        else {
            state_free(state1);
        }
```

```
        state2_t1 = selected_word_leaf_t1(&state);
        if (state2_t1.fixed == alphabet_size){
            state2_t1.cost = c_cost(&state2_t1, loss);
            return;
        }
        if (state2_t1.fixed < alphabet_size){
            if (num_state >= LISTBOUND){
                printf("error in %d\n", __LINE__);
                exit(1);
            }
            state2_t1.cost = c_cost(&state2_t1, loss);
            for (i = num_state; i > 0; i--){
                if (state2_t1.cost <= list[i - 1].cost){
                    break;
                }
                list[i] = list[i - 1];
            }
            list[i] = state2_t1;
            num_state++;
        }
        else {
            state_free(state2_t1);
        }

        state3 = selected_word_masternode_t1(&state);
        if (state3.fixed <= alphabet_size){
            if (num_state >= LISTBOUND){
                printf("error in %d\n", __LINE__);
                exit(1);
            }
            state3.cost = c_cost(&state3, loss);
            for (i = num_state; i > 0; i--){
                if (state3.cost <= list[i - 1].cost){
                    break;
                }
                list[i] = list[i - 1];
            }
            list[i] = state3;
            num_state++;
        }
        else {
            state_free(state3);
        }
        state_free(state);
    }
}


main(){
    int i;
    double loss, nextloss, l0, l1, l, q0, q1;//q0, q1 は Q(T1|T0), Q(T0|T1)
    loss = 0.369;

    printf("アルファベットサイズ=");
    scanf("%d", &alphabet_size);
    if (alphabet_size > 100){
        puts("要素数オーバー");
        return(0);
    }
    prob = (double*)calloc(alphabet_size, sizeof(double));
    for (i = 0; i < alphabet_size; i++){
        printf("確率 %d を入力", i + 1);
        scanf("%lf", &prob[i]);
        if (prob[i] > 1){
            printf("確率は 1 以下");
            printf("\n");
            return(0);
        }
        if (i > 0){
```

```
            if (prob[i] > prob[i - 1]){
                printf("確率は降順で");
                printf("\n");
                return(0);
            }
        }
    }
    for (i = 0; i < alphabet_size; i++){
        printf("確率 %d=%f\n", i, prob[i]);
    }
FIND_AIFV:
    l0 = 0;
    l1 = 0;
    q0 = 0;
    q1 = 0;
    find_t0(loss);
    for (i = 0; i < alphabet_size; i++){
        l0 += prob[i] * strlen(state2_t0.codeword[i]);
        if (state2_t0.arraycheck[i] == '0'){
            q0 += prob[i];
        }
    }
    find_t1(loss);
    for (i = 0; i < alphabet_size; i++){
        l1 += prob[i] * strlen(state2_t1.codeword[i]);
        if (state2_t1.arraycheck[i] == '1'){
            q1 += prob[i];
        }
    }
    nextloss = (l1 - l0) / (q0 + q1);
    l = (l0 * (q1 / (q0 + q1))) + (l1 * (q0 / (q0 + q1)));
    if (loss != nextloss){
        state_free(state2_t0);
        state_free(state2_t1);
        loss = nextloss;
        goto FIND_AIFV;
    }
    printf("T0\n");
    printstate(&state2_t0);
    printf("T1\n");
    printstate(&state2_t1);
    l = (l0 * (q1 / (q0 + q1))) + (l1 * (q0 / (q0 + q1)));
    printf("Q(T0)=%f Q(T1)=%f 平均符号語長=%f\n", (q1 / (q0 + q1)), (q0 / (q0 + q1)), l);

    return(0);
}
```