論　文

双方向探索による効率的な畳込み符号の重み分布の計算法

笹野　博[†a)]　小上　祐輝[†]　曽根　直人[††]　毛利　公美[†††]
西村　卓也[†]　森井　昌克[††††]

An Efficient Calculating Method for Weight Spectrum of Convolutional Codes by a Bidirectional Search

Hiroshi SASANO[†a)], Yuuki OGAMI[†], Naoto SONE[††], Masami MOHRI[†††], Takuya NISHIMURA[†], and Masakatu MORII[††††]

あらまし　畳込み符号の性能は重み分布によって評価できる．畳込み符号の重み分布は入力と出力の対応関係を表す符号木を探索することで導出される．ある畳込み符号の符号語を考えたとき，その符号の生成多項式の係数を逆順にした符号（逆符号）の中に対応する逆順の符号語が存在する．個々の符号語について，計算量がより少ない符号木で重みの計算をすることによって重み分布を求める双方向走査法を提案する．いくつかの符号に対する重み分布の計算時間を従来法と比較し，本手法が優れていることを示す．また，双方向走査法の効果の要因について明らかにし，本手法は拘束長が大きく計算がより困難となる符号に対して効果的であることを示す．

キーワード　畳込み符号，重み分布，双方向探索

1. まえがき

畳込み符号は比較的単純な構成の符号，復号器によって優れた誤り訂正能力が得られることから，衛星通信，深宇宙通信，地上波デジタル放送などに広く実用化されている．

畳込み符号の性能は最ゆう復号を適用した際の復号誤り確率によって評価することが望ましい．このためにはその符号の重み分布を知る必要がある [1]．

畳込み符号の重み分布は，拘束長がごく小さい場合を除き解析的に計算することは困難であり，符号の入力系列と出力系列の関係を記述する符号木を探索することにより導出するのが一般的である．符号木の規模，そしてそれに伴う計算労力は木の深さに対し，指数関数的に増大する．したがって重み分布を高速に計算するためには不要な部分木への探索をできる限り早く打ち切ることが不可欠の条件となる．

符号木による重み分布の計算手法として，Cedervall らの高速木探索アルゴリズム (FAST) [2] がよく知られている．ある畳込み符号と，その生成多項式の係数を逆順にした符号（逆符号）の重み分布は同一である．FAST では不要な探索を打ち切るためのしきい値として，逆符号の列距離関数を効果的に用いている．このしきい値の値が大きいほど探索の深さは抑制され，計算量を減少させることができる．森井らの改良 FAST (IFAST) [3], [4] は逆符号の列距離関数の値に適当な整数値を加算することでより早い段階で探索を打ち切り，計算量を抑える工夫を行っている．しかし，このことによって，必要な符号語を含む部分木までをも棄却してしまう可能性が生じる．そこで，逆符号の符号木を探索することにより計数からもれた符号語の補償を行っている．このような改良により，IFAST は FAST に比べ計算量を 1/2 にまで減少させることができる．しかし，最適なパラメータを事前に決定することが困難であるという問題を有する．

FAST 及び IFAST ではいずれも符号の列距離関数

† 近畿大学理工学部，東大阪市
School of Science and Engineering, Kinki University, Higashi-Osaka-shi, 577–8502 Japan
†† 鳴門教育大学情報処理センター，鳴門市
Information Processing Center, Naruto University of Education, Naruto-shi, 772–8502 Japan
††† 岐阜大学総合情報メディアセンター，岐阜市
Information and Multimedia Center, Gifu University, Gifu-shi, 501–1193 Japan
†††† 神戸大学大学院工学研究科，神戸市
Graduate School of Engineering, Kobe University, Kobe-shi, 657–8501 Japan
a) E-mail: sasano@info.kindai.ac.jp

を探索を打ち切るしきい値として用いている．しかし，これは符号木におけるある深さの枝の重みの最小値であり，しきい値として用いることが必ずしも最適であるとはいえない．ある符号 C とその逆符号 $\tilde{C}$ を考える．C の任意の符号語にはこれに対応した，逆順の符号語が $\tilde{C}$ に存在する．符号 C の重み分布を求めるためには，そのいずれか一方の符号語の重みを計算すればよい．ある符号語の重みを計算するための計算量は，符号語の末尾部分の重みに強く依存する．この重みを探索を打ち切るしきい値として用いた場合，重みが大きいほど計算量は減少する傾向にある．したがって，符号語の先頭部分の重みと同じ長さの末尾部分の重みを比べ，後者の方が大きい場合は C の符号木で，逆の場合は $\tilde{C}$ のそれで計算すれば全体の計算量を小さくすることができる．このように個々の符号語について，C 及び $\tilde{C}$ のそれぞれの符号木を探索するという原理に基づき，効率良く重み分布を計算できる双方向走査法を提案する．本手法の原理が重み分布を計算する手法として非常に合理的であることを明らかにする．そしてこれにより FAST 及び IFAST と比較して計算時間を大幅に削減できることを示す．

本手法は任意の符号化率に適用可能であるが，説明の簡単のため $R=1/n$ の符号について論じる．

本論文の構成を以下に示す．**2.** で畳込み符号，及び従来の重み分布の計算法について述べ，従来法の問題点を明らかにする．**3.** で，双方向走査法を提案する．また，従来法に比べて計算時間が短縮できる要因について考察し，実際の計算結果から効果を確かめる．

2. 畳込み符号

2.1 畳込み符号と重み分布

符号化率 $R=1/n$，拘束長 m の畳込み符号 C の生成多項式を

$$\mathbf{g}^{(i)}=(g_0^{(i)}g_1^{(i)}\cdots g_m^{(i)}) \quad (i=1,2,\cdots,n)$$

とする．拘束長 m は符号器のシフトレジスタの段数とする．シフトレジスタの内部状態 S_t は符号器の状態と呼ばれ，入力系列を $\mathbf{u}=(u_0\ u_1\ \cdots)$ とすれば，u_t が入力されるときの状態は $S_t=(u_{t-1}\ u_{t-2}\ \cdots\ u_{t-m})$ で表される．符号語 $\mathbf{v}$ は状態が全零である符号器に $u_0=1$ なる系列 $\mathbf{u}$ が入力され，再び符号器の状態が全零に戻るまでに出力される系列とする．$u_0=u_\Lambda=1$ である系列を $\mathbf{u}_{[0,\Lambda]}=(u_0\ u_1\ \cdots\ u_\Lambda)$ とすれば，符号語 $\mathbf{v}$ は長さ $\Lambda+m+1$ の入力系列 $\mathbf{u}=(\mathbf{u}_{[0,\Lambda]},0\cdots0)$ により生成される．

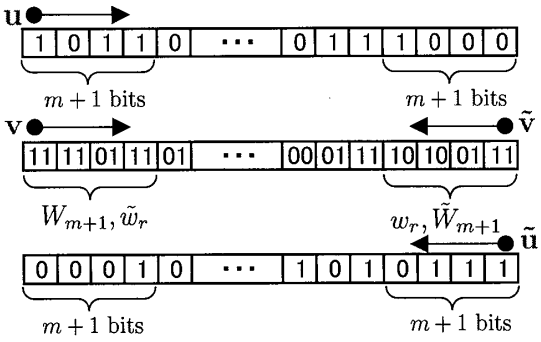


図 1　符号 C と逆符号 $\tilde{C}$ の入力系列と出力系列の対応 ($R=1/2$, $m=3$, $(\mathbf{g}^{(1)},\ \mathbf{g}^{(2)})=(17,15)$)

Fig. 1　Input and output sequences of a code C and its reverse code $\tilde{C}$.

ここで，$\mathbf{g}^{(i)}$ の並びを逆順にした生成多項式

$$\tilde{\mathbf{g}}^{(i)}=(g_m^{(i)}\ g_{m-1}^{(i)}\ \cdots\ g_0^{(i)}) \quad (i=1,2,\ldots,n)$$

を考える．$\tilde{\mathbf{g}}^{(i)}$ によって生成される符号を逆符号，その符号語を $\tilde{\mathbf{v}}$ とする．$\mathbf{u}_{[0,\Lambda]}$ の逆順を $\tilde{\mathbf{u}}_{[0,\Lambda]}$ としたとき，$\tilde{\mathbf{v}}$ は，$\tilde{\mathbf{u}}=(\tilde{\mathbf{u}}_{[0,\Lambda]},0\cdots0)$ により生成される．

図 1 に，$R=1/2$，$m=3$，$(\mathbf{g}^{(1)},\mathbf{g}^{(2)})=(17,15)$ の畳込み符号 C とその逆符号 $\tilde{C}$ に対する入力系列と出力系列の例を示す．生成多項式 $\mathbf{g}^{(1)}$，$\mathbf{g}^{(2)}$ は 8 進数で表記する．図中の系列は上からそれぞれ $\mathbf{u}$，$\mathbf{v}$，$\tilde{\mathbf{v}}$，$\tilde{\mathbf{u}}$ を表しており，黒丸は始点を，矢印は系列の向きを表す．$\mathbf{v}$ と $\tilde{\mathbf{v}}$ のハミング重み（以下，単に重みと記す）は明らかに同一である．$\mathbf{u}_{[\Lambda,\Lambda+m]}=(10\cdots0)$ に対する出力系列の重みを w_r とし，$\mathbf{u}_{[0,m]}$ に対する出力系列の重みを W_{m+1} とする．$\tilde{C}$ に対しても同様に，$\tilde{w}_r$，$\tilde{W}_{m+1}$ を定義すると，ある符号語 $\mathbf{v}$ の w_r，W_{m+1} とその逆順 $\tilde{\mathbf{v}}$ の $\tilde{w}_r$，$\tilde{W}_{m+1}$ には次の関係が成り立つ．

$$\begin{cases} W_{m+1}=\tilde{w}_r \\ w_r\ \ \ =\tilde{W}_{m+1} \end{cases} \tag{1}$$

畳込み符号の性能は重み分布によって評価できる．重み d の符号語の数を A_d，重み d の符号語を生成する入力系列の重みの総和を B_d として，重み分布を $\{A_d\},\{B_d\},d=d_f,d_f+1,\cdots$ で表す．ここで，d_f は自由距離である．求めるべき重み分布の次数はそれを適用する対象によってほぼ決定される．例えばある畳込み符号の性能を評価するため，復号誤り確率の上界を計算する場合，d_f とそれより高次の数項を必要とする．本論文では必要とされる重み分布の最高次数を d_L とする．

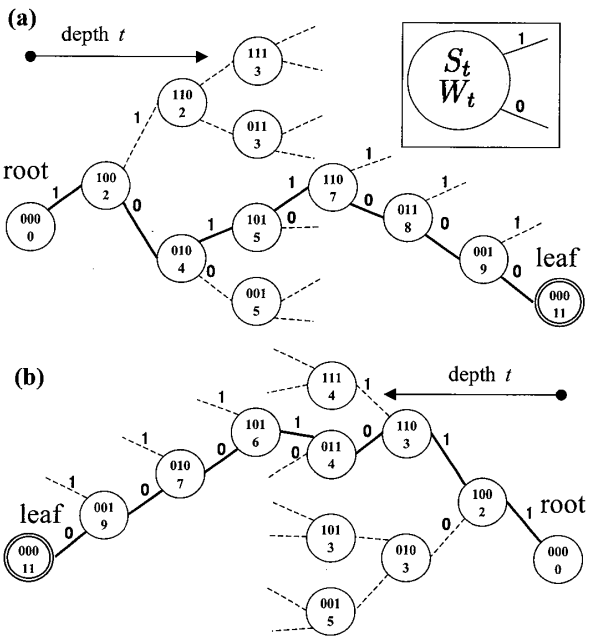


図 2 (a) C の符号の木 (b) $\tilde{C}$ の符号の木
Fig. 2 (a) Code tree of C. (b) Code tree of $\tilde{C}$.

畳込み符号の符号器の動作は，符号の木によって表現できる．図 2 は図 1 の符号 C 及び逆符号 $\tilde{C}$ の木表現である．深さ t のノードには状態 S_t 及び W_t を対応づけ，各ノードから派生する枝をそれぞれ入力 1 の枝及び入力 0 の枝と表す．ただし，W_t は $\mathbf{u}_{[0,t-1]}$ に対する出力系列の重みである．根の深さは 0 であり，二重丸で示した深さ 7 のノードは葉である．根から葉に至るパスが符号語に対応し，葉の重み 11 がその符号語の重みとなる．図中に実線で示したパスは，それぞれ $\mathbf{u}=(1011000)$，$\tilde{\mathbf{u}}=(1101000)$ により生成される符号語 $\mathbf{v}$，$\tilde{\mathbf{v}}$ を表す．重み d_L 以下の葉をすべて走査することで重み分布が計算できる．任意の $\mathbf{v}$ に重みの等しい $\tilde{\mathbf{v}}$ が対応するため，C と $\tilde{C}$ の重み分布は等しい．本論文では，C 及び $\tilde{C}$ の木に対する走査をそれぞれ順走査及び逆走査と呼ぶ．また符号語に対応するパスの走査が葉に至ることを「符号語を確定する」と表す．

2.2 重み分布計算

重み分布の計算量は走査されるノード数によって決定される．走査されるノード数を少なくするためには，あるノードを根とする部分木に重み d_L 以下の葉が存在しなければ，直ちにその部分木への走査を打ち切り，他のノードへ移行することが必要である．走査を打ち切るための条件を「打切り条件」と表記する．

入力 1 の枝に対する打切り条件を考える．深さ t で，

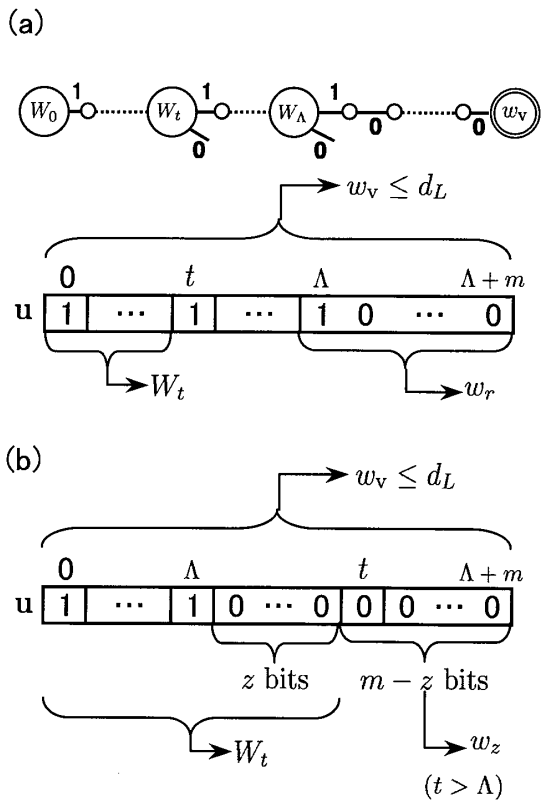


図 3 (a) 式 (2) の説明図 (b) 式 (3) の説明図
Fig. 3 (a) Explanation of Eq. (2). (b) Explanation of Eq. (3).

重みが W_t のノードから伸びた入力 1 の子ノードを根とする部分木に重み w_v の葉があるとする．符号語を生成する入力系列の末尾 $\mathbf{u}_{[\Lambda,\Lambda+m]}=(10\cdots0)$ に対する出力系列の重みは w_r であるので，$W_t+w_r \le w_\mathrm{V}$ となる（図 3 (a) 参照）．走査の対象は $w_\mathrm{V} \le d_L$ であるので，$W_t+w_r \le d_L$ が成り立つ．ゆえに，$W_t+w_r > d_L$ を満たせば，入力 1 の子ノードを根とする部分木には，対象とする符号語は存在しない．したがって，次の式 (2) を打切り条件とすることができる．

$$W_t > d_L - w_r \tag{2}$$

W_t はノードの深さ t に対して単調に増加する．式 (2) の右辺が増加すると，打切り条件を満足するには，より深い走査が必要となり，走査するノード数は式 (2) の右辺に対して指数関数的に増加する．

次に入力 0 の枝に対する打切り条件を考える．直前に z 回 0 が入力された重み W_t のノードに対して，$m-z$ 回連続して 0 を入力すると重み w_V の葉に至る．このとき，$W_t+w_z \le w_\mathrm{V} \le d_L$ が成り立つ（図 3 (b) 参照）．ただし，等号は $t > \Lambda$ のときに成立する．また w_z は入力系列の末尾 $m-z$ ビット $\mathbf{u}_{[\Lambda+z+1,\Lambda+m]}$

による出力系列の重みである．ゆえに $W_t + w_z > d_L$ を満たせば，入力 0 の子ノードを根とする部分木には，対象とする符号語は存在しない．したがって，次の式 (3) を打切り条件とすることができる．

$$W_t > d_L - w_z \tag{3}$$

入力 1 の場合と同様に式 (3) の右辺の増加は走査するノード数の指数関数的増大につながる．

2.3 重み分布計算の従来法

本節では，重み分布計算の従来法である FAST [2], IFAST [3] について述べる．

以下のように定義される列距離関数 $\mathbf{c}$ を考える．

$$\mathbf{c} = (c_0, c_1, \cdots, c_l, \cdots, c_m)$$
$$c_l = \min_{u_0 \neq 0}\{W_{l+1}\}$$

式 (1) より $\tilde{W}_{m+1} = w_r$ であるから，逆符号の $\tilde{c}_m$ は w_r の最小値となり，$\tilde{c}_{m-z-1}$ は w_z の最小値となる．これを式 (2), (3) に適用し，FAST では入力 1 及び入力 0 の枝に対して，以下の打切り条件を用いる．

[FAST の打切り条件]

$$\text{入力 } 1 : W_t > d_L - \tilde{c}_m \tag{4}$$
$$\text{入力 } 0 : W_t > d_L - \tilde{c}_{m-z-1} \tag{5}$$

走査されるノード数の主要部分は式 (4) によって決定され，右辺の値とともに指数関数的に増加する．

IFAST では式 (4) の $\tilde{c}_m$ を任意の整数 α だけ増加させて，打切り条件を式 (6) 及び式 (7) を用いて順走査する．

[IFAST の打切り条件（順走査）]

$$\text{入力 } 1 : W_t > d_L - (\tilde{c}_m + \alpha) \tag{6}$$
$$\text{入力 } 0 : W_t > d_L - \tilde{c}_{m-z-1} \tag{7}$$

式 (6) により打ち切られる部分木には，$w_r < \tilde{c}_m + \alpha$ である符号語 $\mathbf{v}$ が存在する可能性があるため，その補償として逆走査を行い，符号語 $\tilde{\mathbf{v}}$ として確定する．式 (1) より $w_r = \tilde{W}_{m+1}$ であるから，$\tilde{W}_{m+1} < \tilde{c}_m + \alpha$ であるノードを根とする部分木に対して，以下の打切り条件で逆走査する．

[IFAST の打切り条件（逆走査）]

$$\text{入力 } 1 : \tilde{W}_t > d_L - c_m \tag{8}$$
$$\text{入力 } 0 : \tilde{W}_t > d_L - c_{m-z-1} \tag{9}$$

α が大きいほど，順走査のノード数は指数関数的に減少する．一方，逆走査されるノード数は指数関数的に増加する．走査ノードの総計を最小とする α を走査前に決定することは一般に困難である．

FAST はすべての符号語を順走査で確定するために，打切り条件として $\tilde{\mathbf{c}}$ を用いている．しかし，$\tilde{c}_m$ は w_r の最小値であるため，式 (4) の右辺を減少させる効果はさほど大きくない．IFAST では，$\tilde{c}_m$ を α 増加させることにより，この効果を上げているが α の最適値は不明であり，また逆走査においては FAST と同様の問題を残している．このように従来法の打切り条件として用いられている c_m, $\tilde{c}_m$ は必ずしも最適な基準とはいえないことが分かる．

3. 重み分布計算法の提案

3.1 双方向走査法

2.2 で述べたように，順走査及び逆走査のそれぞれに対し，入力 1 の子ノードを根とする部分木への走査の打切り条件として次式を用いる．

$$W_t > d_L - w_r$$
$$\tilde{W}_t > d_L - \tilde{w}_r$$

d_L が与えられたとき，走査するノード数は w_r, $\tilde{w}_r$ の増加に対して指数関数的に減少する．$w_r \geq \tilde{w}_r$ である符号語に対しては順走査，$w_r < \tilde{w}_r$ である符号語に対しては逆走査で確定することにより，走査ノード数を削減できると考えられる．この原理に基づいた双方向走査法の手順について述べる．

[手順 1] $\mathbf{c}$ 及び $\tilde{\mathbf{c}}$ の導出

深さ優先で $t \leq m+1$ のすべてのノードを順走査して $\mathbf{c}$ を求める．$\mathbf{c}$ の初期値をすべて ∞ として，各ノードにおいて $W_t < c_{t-1}$ を満たせば $c_{t-1} \leftarrow W_t$ とする．$W_t \geq c_m$ が成り立てば，走査を打ち切る．$\tilde{\mathbf{c}}$ は同様に逆走査で導出する．

[手順 2] $w_r \geq \tilde{w}_r$ の符号語を順走査で確定する

入力 1 の場合，走査の打切り条件は $W_t > d_L - w_r$ である．$t \geq m+1$ のとき，$w_r < \tilde{w}_r$ なる符号語に対し，この条件は $d_L < W_t + \tilde{w}_r = W_t + W_{m+1}$ すなわち，$W_t > d_L - W_{m+1}$ と表される．$t < m+1$ に対しては W_{m+1} は不明であるが，$W_t \leq W_{m+1} = \tilde{w}_r$ である．任意の符号語について $w_r + W_t \leq w_{\mathrm{v}} \leq d_L$ が成立するから，$w_r \geq \tilde{w}_r$ である符号語については $2W_t \leq d_L$ となる．したがって $2W_t > d_L$ であればこのような符号語は存在せず，走査を打ち切ることがで

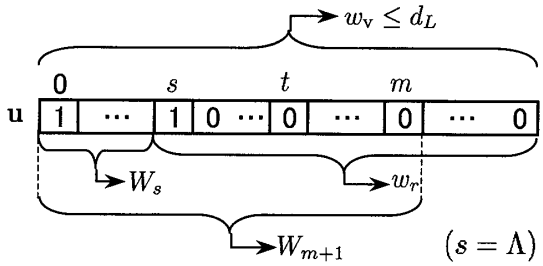


図 4 式 (12) 及び式 (13) の説明図
Fig. 4 Explanation of Eq. (12) and Eq. (13).

きる．以下に入力 1 の枝に対する打切り条件を示す．
[双方向走査法の打切り条件 1（順走査）]

$$\text{入力 } 1 : W_t > d_L - W_{m+1} \quad (t \ge m+1) \quad (10)$$
$$2W_t > d_L \quad (t < m+1) \quad (11)$$

入力 0 の場合の条件は FAST と同様である．ただし，$t < m+1$ についてはもう一つの条件が考えられる．$s < t$ で最後の入力 1 を u_s，ノードの重みを W_s とする．このとき，$W_s + w_r \le w_v$ すなわち $d_L - W_s \ge w_r$ が成り立つ（図 4 参照）．ただし，等号は $s = \Lambda$ のとき成立する．$W_{m+1} = \tilde{w}_r \ge W_t$ であるから $W_t > d_L - W_s$ が成り立てば $\tilde{w}_r > w_r$ となり，走査を打ち切ることができる．以下に入力 0 の枝に対する打切り条件を示す．
[双方走査法の打切り条件 2（順走査）]

$$\text{入力 } 0 : W_t > d_L - \tilde{c}_{m-z-1} \quad (t \ge m+1) \quad (12)$$
$$\begin{cases} W_t > d_L - \tilde{c}_{m-z-1} \\ \quad \text{または} \quad (t < m+1) \\ W_t > d_L - W_s \end{cases} \quad (13)$$

A_d，B_d $(d \le d_L)$ の初期値を 0 として，上記の打切り条件 1 及び条件 2 を用いて深さ優先で順走査を行う．$S_t = \mathbf{0}$ となれば符号語が確定される．ただし，逆走査で同じ符号語を二重に計数してしまうことを防ぐために $w_r \ge W_{m+1}(= \tilde{w}_r)$ を満たす場合にのみ A_d，B_d を更新する．

[手順 3] $\tilde{w}_r > w_r$ の符号語を逆走査で確定する

打切り条件は順走査と同様，以下のとおりである．
[双方走査法の打切り条件（逆走査）]

$$\text{入力 } 1 : \tilde{W}_t \ge d_L - \tilde{W}_{m+1} \quad (t \ge m+1) \quad (14)$$
$$2\tilde{W}_t \ge d_L \quad (t < m+1) \quad (15)$$
$$\text{入力 } 0 : \tilde{W}_t > d_L - c_{m-z-1} \quad (t \ge m+1) \quad (16)$$

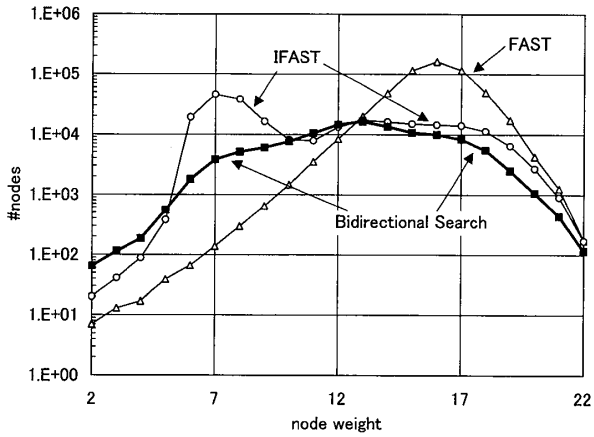


図 5 それぞれの方法におけるノードの重みとノード数
Fig. 5 Node weight and the number of nodes in each method.

$$\begin{cases} \tilde{W}_t > d_L - c_{m-z-1} \\ \quad \text{または} \quad (t < m+1) \\ \tilde{W}_t \ge d_L - \tilde{W}_s \end{cases} \quad (17)$$

同じ符号語を二重に計数することを防ぐため，$\tilde{w}_r > \tilde{W}_{m+1}$ を満たす場合にのみ A_d，B_d の更新を行う．すべてのノードが走査されれば，符号の重み分布が得られる．

3.2 双方向走査法の効果

本節では，双方向走査法の効果を考察する．図 5 は $R = 1/2$，$(g^{(1)}, g^{(2)}) = (5056615, 6717423)$ なる符号について次数 $d_L = d_f = 24$ 以下の重み分布を FAST，IFAST，及び双方向走査法により計算した際の，走査したノードの重みとノード数を示している．ただし，順走査及び逆走査の特徴を明確にするためノードの重みは順走査の場合は W_t を，逆走査の場合は左右を逆転させた $d_L - \tilde{W}_t$ を W_t として示している．また，IFAST についてはノードの総数が最小となる $\alpha = 3$ の場合について示した．

FAST は，順走査のみで重み分布を計算する．この符号は $\tilde{c}_m = 9$ であり，$W_t \le d_L - \tilde{c}_m = 15$ では入力 1 の打切り条件を満たさないため，ノード数が指数関数的に増加している．最大値は $W_t = 16$ で 163,044 個となり，$W_t = 15 \sim 17$ におけるノード数が総数の約 7 割を占めている．

IFAST では順走査及び逆走査が行われる．図にはそれぞれの走査におけるノード数の和を示している．ノード数は $W_t = 7$ 及び 13 で極大となる．前者は逆走査に，後者は順走査に起因する．$\alpha = 3$ が順走査において FAST の最大値 $W_t = 16$ との差に現れている．

α の効果により，順走査においてノード数の増加は穏やかであるが，逆走査の $W_t = 9$ 付近は指数関数的増加が見られる．最大値は $W_t = 7$ で 45,675 個となる．

双方向走査法では順走査及び逆走査が行われる．$W_t = 7$〜18 においてノード数に急激な変化は見られず，全体的に他の方法より少数であることが分かる．また，走査の方向による極大はなく，$W_t = 13$ で最大となる．

表 1 に FAST，IFAST 及び双方向走査法のノード数の比較を示す．ノード数は総数とともに深さ $t \geq m+1$ と $t < m+1$ 及び列距離関数導出の場合に分類して示している．括弧内は，それぞれの場合の FAST に対する比率を表す．列距離関数の項は他の項に比べて，はるかに影響が少ないことは明らかである．表より，双方向走査法では FAST，IFAST に比べて大幅にノード数が削減されていることが分かる．その中で，$t \geq m+1$ の場合の削減が最も支配的であることが分かる．

表 1　FAST, IFAST 及び双方向走査法の走査ノード数
Table 1　Number of searching nodes in FAST, IFAST, and bidirectional search.

	ノード数			
	総数	$t \geq m+1$	$t < m+1$	$\mathbf{c}$, $\tilde{\mathbf{c}}$ 導出
FAST	551,942	387,645	162,276	2,021
IFAST	248,026	181,971	62,313	3,742
(比率 [%])	(45)	(47)	(38)	−
双方向走査法	118,917	67,490	47,685	3,742
(比率 [%])	(22)	(17)	(29)	−

この原因について以下に考察する．

順走査 ($t \geq m+1$) におけるすべてのノードに対し，二つの値 $W_{m+1} = \tilde{w}_r$ 及び $w_x = d_f - W_s$ を割り当てる．式 (13) で用いた W_s は $u_s = 1$ のときに更新され，単調に増加する重みである．w_x は w_r の上界値であり，符号語が確定されれば $w_x = w_r$ となる．すなわち，これらの値は各ノードをいずれの方向に走査すれば有利かを示す指標となる．逆走査 ($t \geq m+1$) においても同様にすべてのノードに対し，$\tilde{W}_{m+1} = w_r$ と $\tilde{w}_x$ を割り当てる．

表 2 は図 5 と同じ符号に対して重み分布を計算したときの，$t \geq m+1$ におけるノードに割り当てられた値とノード数の関係を表している．各項の上段は順

表 2　順走査及び逆走査における各ノードの w_x，$\tilde{w}_x$，W_{m+1}，$\tilde{W}_{m+1}$ とノード数
Table 2　w_x, $\tilde{w}_x$, W_{m+1}, and $\tilde{W}_{m+1}$ of each node and the number of nodes by the forward and the backward search.

W_{m+1}(Forward) / $\tilde{w}_x$(Backward)	w_x(Forward) / $\tilde{W}_{m+1}$(Backward) 8	9	10	11	12	13	14	15	16	17	18	19	20
7	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0
8	0 0	1011 4109	931 18181	630 27566	287 39314	189 51261	62 57653	53 53255	20 67354	0 11066	0 5479	0 0	0 0
9	0 0	1690 2836	1[illegible]4 1[illegible]93	805 19151	461 26938	211 35526	[illegible]08 [illegible]90	101 54883	0 17703	0 6355	0 1647	0 1	0 0
10	0 0	4411 1648	3696 7897	2355 11798	1403 16464	535 21455	488 34706	66 12387	27 9744	0 2744	0 934	0 2	0 0
11	0 0	11257 926	9534 4209	5758 6116	3020 8831	2587 16848	306 7032	137 6375	56 4359	0 1356	0 436	0 7	0 0
12	0 0	19057 550	15380 2181	9797 3358	7846 7219	1338 3566	728 3515	203 2636	50 2019	12 729	0 212	0 14	0 0
13	0 0	26137 252	20878 1092	19772 2491	4250 1181	1907 1532	508 1515	282 1447	36 958	32 458	0 129	0 22	0 0
14	0 0	28542 86	36655 799	8889 428	4422 477	1902 629	864 608	251 604	111 432	36 252	0 76	0 29	0 2
15	0 0	45395 126	14727 [illegible]7	8120 162	3625 262	1698 266	716 253	262 173	154 206	87 169	0 58	0 16	0 1
16	0 0	9380 20	[illegible]255 [illegible]5	5757 57	3015 49	1399 71	[illegible]95 124	326 67	193 150	64 135	0 48	22 33	0 1
17	0 0	4219 0	3388 15	2979 24	1942 38	982 8	535 47	249 30	113 45	74 73	38 38	0 3	0 2
18	0 0	409 0	927 0	868 0	539 0	385 14	260 13	140 0	109 43	52 49	4 11	15 16	11 13
19	0 0	53 0	144 0	200 0	174 0	134 10	117 0	85 5	48 0	40 8	35 23	10 11	0 4
20	0 0	0 0	1 0	5 0	11 0	21 0	22 0	21 0	10 0	7 0	9 5	2 0	10 10

表 3 表 2 の各領域におけるノード数
Table 3 Number of searching nodes in the regions of Table 2.

A	18,130	25,271	B
	110,147	640,157	
C	316,136	28,108	D
	18,636	3,717	

表 4 IFAST 及び双方向走査法の走査方向
Table 4 Direction of the search in IFAST and bidirectional search in the regions of Table 2.

(1) IFAST

A	Backward	Forward	B
C	Backward	Forward	D

(2) Bidirectional Search

A	Forward	Forward	B
C	Backward	Backward	D

走査による，下段は逆走査によるノード数を示している．例えば，順走査において，$w_x = 9$，$W_{m+1} = 8$ となるノードは 1,011 個存在することを示している．重み分布を求めるためには，すべての項について，上段若しくは下段に示されている数のノードを走査する必要がある．表中では，順走査のノード数が逆走査のそれより少ない項を斜線で表している．すなわち，斜線で示した項には順方向で，それ以外の項には逆方向で走査すればノードの総数が削減できることが分かる．

FAST ではすべてのノードを順走査する．IFAST では $\alpha = 3$ としたとき $w_x \geq \tilde{c}_m + \alpha = 12$ であるノードを順走査，$w_x \leq 11$ であるノードを逆走査する．表 2 では，この境界を二重の縦線で示している．双方向走査法では $w_x \geq W_{m+1}$ であるノードを順走査，$\tilde{w}_x > \tilde{W}_{m+1}$ であるノードを逆走査する．表 2 ではこの境界を階段状の太線で示している．この太線及び二重線で 4 分割された領域をそれぞれ領域 A～D とする．表 3 に領域 A～D における順走査及び逆走査のノード数の総和を示す．FAST はすべての領域を順走査するため，特に領域 C で多数のノードを走査していることが分かる．

IFAST 及び双方向走査法のそれぞれの領域における走査方向を表 4 に示す．IFAST と双方向走査法では領域 A 及び領域 D で走査方向が逆転していることが分かる．双方向走査法はいずれの領域においても有利な方向の走査を行っており，この結果，最も少ないノード数を達成していることが分かる．

3.3 計算結果

$R = 1/2$，$m = 17, 19, 21, 23, 25$ の符号の集合に対する FAST，IFAST 及び双方向走査法による重み分布の計算時間の比較を表 5 に示す．重み分布は $d_f + 4$ 次まで計算した．ただし，IFAST の α は $m = 17, 19, 21, 23, 25$ の少数の符号に対してあらかじめ重み分布計算を行い，それぞれ走査時間が最も少なくなる $\alpha = 3, 3, 4, 4, 4$ を用いた．FAST の計算時間が拘束長に対しほぼ均一になるよう符号数を選んだ．また括弧内は FAST に対する計算時間の比率を表しており，この比率は重み分布の次数によらずほぼ一定であることが確かめられている．

表 5 FAST，IFAST 及び双方向走査法による計算時間 ($R = 1/2$)
Table 5 Search time of FAST, IFAST, and bidirectional search ($R = 1/2$).

拘束長	17	19	21	23	25
符号数 [$\times 10^3$]	390.0	100.0	30.0	10.0	2.0
FAST [min]	1093	1025	1122	1321	928
IFAST [min]	387	311	282	273	166
(比率 [%])	(35.4)	(30.3)	(25.1)	(20.7)	(17.9)
双方向走査法 [min]	187	130	106	87	47
(比率 [%])	(17.1)	(12.7)	(9.4)	(6.6)	(5.1)

IFAST では，FAST に比べ計算時間が大きく削減されているが，双方向走査法では，これが更に大幅に削減されていることが分かる．また，拘束長が大きくなるにつれて比率が低下しており，$m = 17$ では IFAST と比較して 1/2 程度であるのに対して，$m = 25$ では 1/3 以下になっている．これらのことから，拘束長が大きく重み分布の計算により多くの時間を要する符号に対して双方向走査法が効果的であるといえる．

次に，$R = 1/2$ に比べてより多くの時間を必要とする $R = 2/3$ の場合について考える．$m = 6, 7, \cdots, 15$ についてランダムに選んだ 3,000 個の符号の集合に対して，FAST，IFAST 及び双方向走査法により，d_f 次における重み分布を計算した際の計算時間を図 6 に示す．IFAST の α は，事前に少数の符号から求めた走査時間が最小となる $\alpha = 1$ ($m = 6, 7$)，$\alpha = 2$ ($m = 8, 9, \cdots, 15$) を用いた．計算時間は実線で，FAST に対する計算時間の比率は破線で示している．どの方法においても拘束長の増加に対して計算時間は指数関数的に増加するが，常に双方向走査法が最小であり，各方法による差は拡大する方向にあることが分かる．IFAST と比べ双方向走査法は拘束長が増加するほど計算時間の比率が低下しており $m \geq 9$ ではほぼ 1/2 になることが分かる．

表 5 及び図 6 の結果から，拘束長若しくは符号化

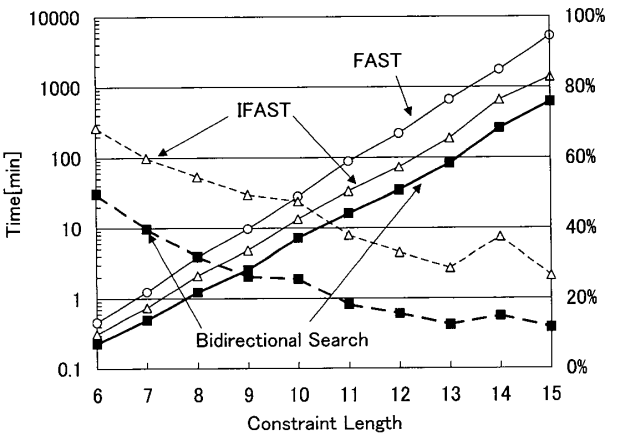


図 6　FAST，IFAST 及び双方向走査法による計算時間 ($R = 2/3$)

Fig. 6　Search time of FAST, IFAST, and bidirectional search ($R = 2/3$).

率と拘束長がともに大きく，計算労力が多大で重み分布計算が困難となる符号ほど本走査法の効果が大きい傾向が確かめられ，双方向走査法は従来法に比べて効率的に重み分布を計算するために有効であるといえる．

4. むすび

ある畳込み符号の個々の符号語に対して，その符号と逆符号のうち，より計算量が少ない符号木で重みを計算することによって重み分布を求める双方向走査法を提案した．いくつかの符号に対する重み分布の計算時間を従来法と比較し，本手法が優れていることを示した．また，双方向走査法の効果の要因について明らかにし，本手法は拘束長が大きく計算がより困難となる符号に対して効果的であることを示した．

文　献

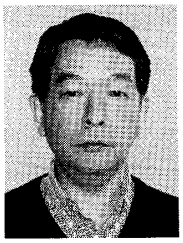
[1] L.H.C. Lee, Convolutional coding: Fundamentals and applications, Artech House, 1997.

[2] M. Cedervall and R. Johannesson, "A fast algorithm for computing distance spectrum of convolutional codes," IEEE Trans. Inf. Theory, vol.35, no.6, pp.1146–1159, Nov. 1989.

[3] 森井昌克，谷川晃一，笹野　博，"畳込み符号の重み分布を求める Cedervall-Johannesson アルゴリズム (FAST) の高速化について，" 信学論 (A)，vol.J78-A, no.3, pp.416–426, March 1995.

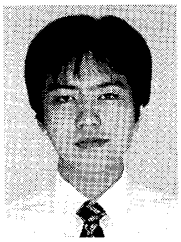
[4] N. Sone, M. Morii, and H. Sasano, "Recursive Cedervall-Jahannesson algorithm computing the distance spectrum of covolutional codes," Proc. ISITA'98, pp.14–16, Mexico City, Mexico, Oct. 1998.

（平成 20 年 5 月 7 日受付，8 月 5 日再受付）



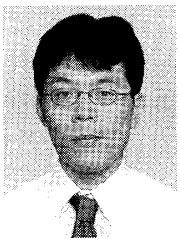
笹野　博　（正員）

昭 49 阪大・工・通信卒，昭 52 同大大学院工学研究科修士課程了．工博．同年近畿大学理工学部電気工学科助手，平 12 同助教授．主に情報理論，符号理論等の研究に従事．IEEE，情報処理学会，計測自動制御学会，システム制御情報学会，情報理論とその応用学会各会員．



小上　祐輝　（正員）

2000 近畿大・理工・電子卒．2002 同大大学院修士課程了．博士（工学）．符号理論，主に畳込み符号に関する研究に従事．

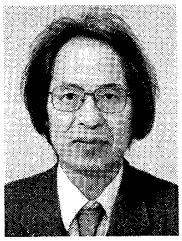


曽根　直人　（正員）

平 4 愛媛大・工・情報卒．平 6 同大大学院工学研究科情報工学専攻博士前期課程了．平 6 鳴門教育大学情報処理センター助手，現在に至る．主にコンピュータネットワーク，符号理論等の研究・教育に従事．

毛利　公美　（正員）

平 5 愛媛大・工・情報卒．平 7 同大大学院工学研究科情報工学専攻博士前期課程了．同年，香川短期大学助手．平 10 徳島大学工学部知能情報工学科助手，平 15 同講師．平 14 博士（工学）．平 19 岐阜大学総合情報メディアセンター准教授として現在に至る．主に符号理論，ネットワークセキュリティ等の研究・教育に従事．情報理論とその応用学会，IEEE 各会員．



西村　卓也　（正員）

1965 近畿大・理工・電気卒．同年同大電子工学科助手，現在，同大情報学科教授．低次元離散時間近似モデルの作成及び情報理論と制御理論の融合に関する研究に従事．IEEE，情報処理学会，計測自動制御学会及びシステム制御情報学会各会員．工博．

森井　昌克　（正員）

昭 58 佐賀大・理工・電気卒．昭 64 阪大大学院博士課程了．工博．同年京都工繊大・工芸・電気電子情報・助手．平 2 愛媛大・工・情報・講師，平 5 助教授．平 7 徳島大・工・知能情報・教授．平 17 神戸大・工・電気電子・教授．現在，同大大学院工学研究科教授．情報理論，暗号理論，ネットワークセキュリティ，代数的符号理論，インターネット応用などの研究に従事．IEEE，情報理論とその応用学会各会員．平 6 電気通信普及財団賞受賞．平 12 四国情報通信局局長表彰．平 19 本会基礎・境界ソサイエティ功労賞受賞．