The 27th Symposium on Information Theory
and Its Applications (SITA2004)
Gero, Gifu, Japan, Dec. 14–17, 2004

Small-Loop を含む低密度パリティ検査 (LDPC) 符号の復号に関する研究
A Note on Decoding of Low Density Parity Check Codes with Small-Loop

小林 直人* 松嶋 敏泰* 平澤 茂一*
Naoto Kobayashi Toshiyasu Matsushima Shigeichi Hirasawa

Abstract— Low Density Parity Check(LDPC) codes decoded by Sum-Product algorithm(SPA) have good performances.If graphical model of LDPC codes include loop structure, especially loop size of 4 structures(Small-Loop), performance of SPA gets worse. Cluster method[2] or algorithm with junction graph[3] can reduce loop structure from graphical model,but these method are more complicated and requires more memory than SPA. We propose a new method of clustering Small-Loop. This method requires same memory size as SPA. And we show the usefulness of this method by computer simulation.

Keywords— LDPC codes , Sum-Product Algorithm , small-loop , clustering

1 はじめに

疎な検査行列を持つ低密度パリティチェック (LDPC) 符号は，復号アルゴリズムとして Sum-Product アルゴリズム (SPA) を用いると非常に良い復号特性を持つため，近年活発に研究がなされている [1].

SPA は確率モデルをグラフィカルモデルで表現し，ノード間でメッセージ伝播を行うことで周辺事後確率を計算するアルゴリズムである．グラフィカルモデルにループ構造が存在しない場合は，SPA は正確な事後確率を計算することができる．ループ構造が存在する場合は，特に理論的保証は無いが，その計算結果が事後確率の近似となることが実験的に確かめられている．LDPC 符号は，一般的にグラフィカルモデルにループ構造が存在するため，SPA の計算結果は近似事後確率となるが，特に長さが 4 のループ (Small-Loop と呼ぶ) は SPA の近似性能を悪くする [5]．その対処法として，小さな Loop を含まないように符号を構成する方法や，Loop 構造に含まれる複数の符号語ビットを 1 つの変数として扱い（クラスタリング）復号する方法が行われている．後者の研究としては，[2] のクラスター BP や [3] の Junction Graph を用いたアルゴリズム等が存在するが，共に計算式が SPA より複雑となり，メッセージ伝播に必要なメモリ量が増加する．

本研究では，2 元符号の性質を利用することで，SPA のメッセージ伝播に必要なメモリ量を増加させずに，Small-Loop をクラスタリングした場合と同じ結果を求めるアルゴリズムを提案する．さらに，Small-Loop を含む比較的符号長の短い符号でシミュレーションを行いクラスタリングの有用性を検証した．なお，本稿では Small-Loop のクラスタリングについてのみ取り扱っているが，これ以外のループ構造についても同様のアルゴリズムが実現可能である（この場合，クラスタリングする変数の組み合わせにより，同一の符号で複数のグラフ表現が可能となるため，その考慮が必要となる）.

*〒 169–8555 東京都新宿区大久保 3–4–1 早稲田大学理工学部経営システム工学科 Dept. of Industrial & Management Systems Engineering, School of Science and Engineering, Ookubo 3-4-1, Shinjyukuku, Tokyo, 169-8555 Japan. E-mail: kobayashi@matsu.mgmt.waseda.ac.jp

2 準備

2.1 低密度パリティチェック符号

(2 元) 低密度パリティチェック (LDPC) 符号 [1] とは，パリティ検査行列 H に含まれるシンボル 1 の個数が，符号長に対して非常に少ない符号である．シンボル 1 の配置は基本的にランダムに決定される.

2.2 Sum-Product アルゴリズム

Sum-Product アルゴリズム (SPA) は，確率モデルをグラフィカルモデルで表現し，そのノード間でメッセージ伝播を行うことで事後周辺確率を計算するアルゴリズムである．グラフィカルモデルにループ構造が存在する場合は，正確な事後確率は計算できないが，符号の復号のように，グラフ構造に特殊な条件を持たせたり，また終了条件を定めた下で用いることにより，その計算結果が事後確率の近似となることが知られている.

以下に LDPC 符号の復号に用いる際のアルゴリズムを示す.

$\mathcal{N}(m) \equiv \{n : H_{mn} = 1\}$，$\mathcal{M}(n) \equiv \{m : H_{mn} = 1\}$ とする．H_{mn} はパリティ検査行列の m 行 n 列目の値である $(0 \le m < M, 0 \le n < N)$．また，無記憶通信路を仮定し，符号語 $(x_0, x_1, ..., x_{N-1}) \in \{0,1\}^N$ を送信，受信語 $(y_0, y_1, ..., y_{N-1}) \in \mathcal{R}^N$ を受信した元での，各符号語ビット x_i の尤度を $f_i^a = \Pr\{y_i | x_i = a\}$ とする.

$H_{mn} = 1$ を満たす全ての組 (m, n) に対し計算を行う.

Initialization.

$$\begin{cases} q_{mn}^0 = f_n^0 \\ q_{mn}^1 = f_n^1 \end{cases} \tag{1}$$

と初期化する.

Horizontal step.

$$\begin{cases} r_{mn}^0 = ((1 + \delta r_{mn})/2) \\ r_{mn}^1 = ((1 - \delta r_{mn})/2) \end{cases} \tag{2}$$

但し

$$\delta r_{mn} = \prod_{n' \in \mathcal{N}(m) \backslash n} (q_{mn'}^0 - q_{mn'}^1) \tag{3}$$

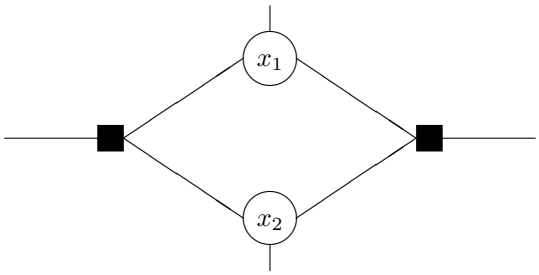

図 1: Factor Graph の例

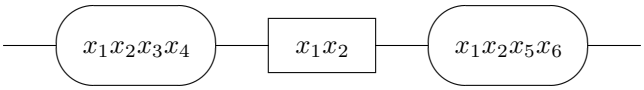

図 2: Junction Graph の例

Vertical step.

$$\begin{cases} q^0_{mn} = \alpha(f^0_n \prod_{m' \in \mathcal{M}(n)\setminus m} r^0_{m'n}) \\ q^1_{mn} = \alpha(f^1_n \prod_{m' \in \mathcal{M}(n)\setminus m} r^1_{m'n}) \end{cases} \tag{4}$$

(α は正規化定数)

Pseudoposterior probabilities.

$$\begin{cases} q^0_n = \alpha(f^0_n \prod_{m' \in \mathcal{M}(n)} r^0_{m'n}) \\ q^1_n = \alpha(f^1_n \prod_{m' \in \mathcal{M}(n)} r^1_{m'n}) \end{cases} \tag{5}$$

Decoding.

$$C_n = \begin{cases} 0 & (q^0_n \geq q^1_n) \\ 1 & (q^0_n < q^1_n) \end{cases} \tag{6}$$

として

$$(C_0, ..., C_{N-1})H^t = 0 \tag{7}$$

を満たしていれば，$(C_0, ..., C_{N-1})$ を推定符号語として終了．そうでなければ，**Horizontal step.** へ戻る (指定回数反復しても，(7) を満たさない場合，その時点の $(C_0, ..., C_{N-1})$ を推定符号語として終了 (もしくは推定語非発見として終了)).

2.3 符号のグラフ表現

線形符号は Tanner Graph[4] や Junction Graph[3] と呼ばれるグラフィカルモデルで表現することができる．Tanner Graph は符号語の各ビットに対応する変数ノードと，検査行列の各行（線形制約）に対応するチェックノードから構成される．Junction Graph は検査行列の各行（線形制約）に対応する Cluster ノードと，符号語の各ビットの集合に対応する Intersection ノードから構成される．

パリティ検査行列 H に，式 (8) のようなシンボル 1 の配置があった場合にタナーグラフで表現すると，長さ 4 のループ構造 (Small-Loop) となる (図 1)．Small-Loop が存在すると，SPA の復号性能が悪くなることが実験的に確かめられている [5].

$$H = \begin{pmatrix} & \vdots & & \vdots & \\ \cdots & 1 & \cdots & 1 & \cdots \\ & \vdots & & \vdots & \\ \cdots & 1 & \cdots & 1 & \cdots \\ & \vdots & & \vdots & \end{pmatrix} \tag{8}$$

同様に Junction Graph で表現すると，図 2 のように Loop 構造にならない．この場合，対象となる符号語 2 ビット (図 2 における x_1, x_2) が，4 元に拡大されたノードとして扱われる（「クラスタリング」される）．従って，このグラフィカルモデルに SPA を適用すると，このノードに対するメッセージは，$\{q^{00}_{mn}, q^{01}_{mn}, q^{10}_{mn}, q^{11}_{mn}\}$ のように 4 元となり，このメッセージを用いる計算を他の計算と場合分けする必要が生じる．

なお本稿では，式 (8) のようなシンボル 1 が 2 つ重複するような箇所が，3 行又は 3 列以上存在するような符号を除外して考える．

3 提案法

3.1 クラスタリングされたノードの計算

本章では符号を Junction Graph で表現し，クラスタリングされた符号語ビットが存在する場合の SPA による復号方法を考える．簡単のため 2 つの符号語ビットがクラスタリングされた場合のみ説明するが，それ以上の符号語ビットがクラスタリングされた場合でも同様のことが言える．

SPA の **Horizontal step.** は，

$$\begin{cases} r^0_{mn} = \sum_{\{x_{n'}: n' \in \mathcal{N}(m)\setminus n\}} \psi(x_n = 0, \{x_{n'}\}) \\ \qquad \times \prod_{n' \in \mathcal{N}(m)\setminus n} q^{x_{n'}}_{mn'} \\ r^1_{mn} = \sum_{\{x_{n'}: n' \in \mathcal{N}(m)\setminus n\}} \psi(x_n = 1, \{x_{n'}\}) \\ \qquad \times \prod_{n' \in \mathcal{N}(m)\setminus n} q^{x_{n'}}_{mn'} \end{cases} \tag{9}$$

($\psi(\{x_n\})$ は $\{x_n\}$ が偶重みの場合 1，奇重みの場合 0 を返す関数) の式変形となっている．

式 (9) の n がクラスタリングされた符号語ビットの場合，$\{r^{00}_{mn}, r^{01}_{mn}, r^{10}_{mn}, r^{11}_{mn}\}$ という 4 つのメッセージを計算・保持する必要があるが，$\psi(\{x_n\})$ の性質（線形符号の性質）により $r^{00}_{mn} = r^{11}_{mn}$，$r^{01}_{mn} = r^{10}_{mn}$ となるため，それぞれを r^0_{mn}，r^1_{mn} で代表させることができる．

また，式 (9) の n' がクラスタリングされた符号語ビット n^* を含む場合は，式 (3) を

$$\delta r_{mn} = \Big(\prod_{n' \in \mathcal{N}(m)\setminus\{n, n^*\}} (q^0_{mn'} - q^1_{mn'})\Big)(Q^{even}_{mn^*} - Q^{odd}_{mn^*}) \tag{10}$$

(Q^{even}_{mn} は変数ノード n がとり得る偶重みの元に対してのメッセージ q_{mn} の和，Q^{odd}_{mn} は奇重みの元に対してのメッセージ q_{mn} の和を表す) とすることで，同様の計算を行うことができる．従って，**Vertical step.** において，n がクラスタリングされた符号語ビット n^* の場合，

$\{q^{00}_{mn^*}, q^{01}_{mn^*}, q^{10}_{mn^*}, q^{11}_{mn^*}\}$ の 4 つを保持する必要はなく，$q^0_{mn^*} = Q^{even}_{mn^*}$，$q^1_{mn^*} = Q^{odd}_{mn^*}$ とすれば良い．

以上 2 つの性質を用いると，ループ構造に含まれる変数ノード，チェックノードを場合分けして計算することで，SPA のメッセージ伝播に必要なメモリ量を増加させずに，クラスタリングした場合の計算と同じ結果を求めることができる．

以上のことを Small-Loop 構造に適用したアルゴリズムを次節に示す．

3.2 提案アルゴリズム

$\mathscr{L}_A$ を Small-Loop に含まれる 2 つの変数ノードのうち任意の 1 つの変数ノードの集合とし，$\mathscr{L}_B$ を Small-Loop に含まれる変数ノードのうち $\mathscr{L}_A$ に含まれないノードの集合とする．Small-Loop の定義として，$\mathscr{A} = \{(m,n)|n$ と m が同一のSmall-Loop内のノード$, n \in \mathscr{L}_A\}$，$\mathscr{B} = \{(m,n)|n$ と m が同一のSmall-Loop内のノード$, n \in \mathscr{L}_B\}$ というチェックノード m と変数ノード n のペアの集合を用意する．

$\beta(n)$, $\gamma(n)$, $\zeta(n)$ は $n \in \mathscr{L}_A$ に対して定義され，$\beta(n) \in \mathscr{L}_B$ は n が含まれる Small-Loop の n では無い変数ノードを表し，$\gamma(n)$，$\zeta(n)$ は n が含まれる Small-Loop のチェックノードをそれぞれ表す $(\gamma(n) \neq \zeta(n))$．

$H_{mn} = 1$ を満たす全ての組 (m,n) に対し計算を行う．

Initialization.

$((m,n) \notin \mathscr{A}, (m,n) \notin \mathscr{B})$ の場合

式 (1) と同じ計算を行う．

$((m,n) \in \mathscr{A})$ の場合

$$\begin{cases} q^0_{mn} = f^0_n f^0_{\beta(n)} + f^1_n f^1_{\beta(n)} \\ q^1_{mn} = f^0_n f^1_{\beta(n)} + f^1_n f^0_{\beta(n)} \end{cases} \tag{11}$$

と初期化する．$((m,n) \in \mathscr{B})$ については計算しない（利用しない）．

Horizontal step.

$$\begin{cases} r^0_{mn} = ((1 + \delta r_{mn})/2) \\ r^1_{mn} = ((1 - \delta r_{mn})/2) \end{cases} \tag{12}$$

$((m,n) \notin \mathscr{A}, (m,n) \notin \mathscr{B})$ の場合

式 (3) と同じ計算を行う．

$((m,n) \in \mathscr{A})$ の場合

$$\delta r_{mn} = \prod_{n' \in \mathcal{N}(m) \backslash \{n, \beta(n)\}} (q^0_{mn'} - q^1_{mn'}) \tag{13}$$

$((m,n) \in \mathscr{B})$ については計算しない（利用しない）．

Vertical step.

$(n \notin \mathscr{L}_A, n \notin \mathscr{L}_B)$ の場合

式 (4) と同じ計算を行う．

$(n \in \mathscr{L}_A \vee n \in \mathscr{L}_B)$ の場合

$$\begin{cases} Q_{00} = \prod_{m' \in \mathcal{M}(n) \backslash \{\gamma(n), \zeta(n)\}} r^0_{m'n} \\ \quad \times \prod_{m'' \in \mathcal{M}(\beta(n)) \backslash \{\gamma(n), \zeta(n)\}} r^0_{m''n} \\ \quad \times r^0_{\gamma(n)n} r^0_{\zeta(n)n} f^0_n f^0_{\beta(n)} \\ Q_{01} = \prod_{m' \in \mathcal{M}(n) \backslash \{\gamma(n), \zeta(n)\}} r^0_{m'n} \\ \quad \times \prod_{m'' \in \mathcal{M}(\beta(n)) \backslash \{\gamma(n), \zeta(n)\}} r^1_{m''n} \\ \quad \times r^1_{\gamma(n)n} r^1_{\zeta(n)n} f^0_n f^1_{\beta(n)} \\ Q_{10} = \prod_{m' \in \mathcal{M}(n) \backslash \{\gamma(n), \zeta(n)\}} r^1_{m'n} \\ \quad \times \prod_{m'' \in \mathcal{M}(\beta(n)) \backslash \{\gamma(n), \zeta(n)\}} r^0_{m''n} \\ \quad \times r^1_{\gamma(n)n} r^1_{\zeta(n)n} f^1_n f^0_{\beta(n)} \\ Q_{11} = \prod_{m' \in \mathcal{M}(n) \backslash \{\gamma(n), \zeta(n)\}} r^1_{m'n} \\ \quad \times \prod_{m'' \in \mathcal{M}(\beta(n)) \backslash \{\gamma(n), \zeta(n)\}} r^1_{m''n} \\ \quad \times r^0_{\gamma(n)n} r^0_{\zeta(n)n} f^1_n f^1_{\beta(n)} \end{cases} \tag{14}$$

を計算した後，

$(n \in \mathscr{L}_A, (n,m) \notin \{\mathscr{L}_A, \mathscr{L}_B\})$ ならば

$$\begin{cases} q^0_{mn} = \alpha((Q_{00} + Q_{01})/r^0_{mn}) \\ q^1_{mn} = \alpha((Q_{10} + Q_{11})/r^1_{mn}) \end{cases} \tag{15}$$

$(n \in \mathscr{L}_B, (n,m) \notin \{\mathscr{L}_A, \mathscr{L}_B\})$ ならば

$$\begin{cases} q^0_{mn} = \alpha((Q_{00} + Q_{10})/r^0_{mn}) \\ q^1_{mn} = \alpha((Q_{01} + Q_{11})/r^1_{mn}) \end{cases} \tag{16}$$

$(n \in \mathscr{L}_A, (n,m) \in \{\mathscr{L}_A, \mathscr{L}_B\})$ ならば

$$\begin{cases} q^0_{mn} = \alpha((Q_{00} + Q_{11})/r^0_{mn}) \\ q^1_{mn} = \alpha((Q_{01} + Q_{10})/r^1_{mn}) \end{cases} \tag{17}$$

を計算する．$(n \in \mathscr{L}_B, (n,m) \in \{\mathscr{L}_A, \mathscr{L}_B\})$ の場合は計算しない．

Pseudoposterior probabilities.

$(n \notin \mathscr{L}_A, n \notin \mathscr{L}_B)$ の場合

式 (5) と同じ計算を行う．

$(n \in \mathscr{L}_A)$ の場合

$$\begin{cases} q^0_n = \alpha(Q_{00} + Q_{01}) \\ q^1_n = \alpha(Q_{10} + Q_{11}) \end{cases} \tag{18}$$

$(n \in \mathscr{L}_B)$ の場合

$$\begin{cases} q^0_n = \alpha(Q_{00} + Q_{10}) \\ q^1_n = \alpha(Q_{01} + Q_{11}) \end{cases} \tag{19}$$

Decoding. SPA と同様に行う．

このアルゴリズムは Small-Loop 内の変数ノードへのメッセージの伝播を，1 つのノード $(n \in \mathscr{L}_A)$ に代表させて行っている．また，式 (14) は同一の Small-Loop においてのメッセージに対しては同じ計算となるので，1 つの Small-Loop に対して式 (14) は 1 回計算すればよい．

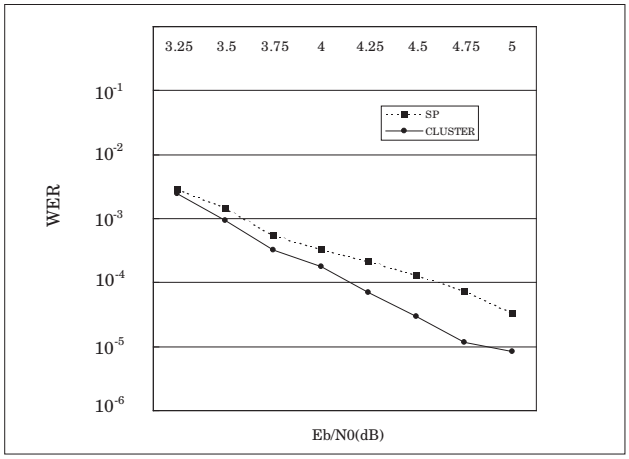

図 3: 実験 1 結果

3.3 計算量評価

SPA と提案アルゴリズムについて演算回数を簡単に比較する．符号長 N，符号化率 0.5，H の列重みを 3，行重みを 6，Small-Loop の個数が L の符号を考える．演算は和，積それぞれ 1 回とカウントし，正規化は考えないこととする．比較を表 1 に示す.

表 1: 演算回数の比較

SPA		提案アルゴリズム (CLUSTER)	
式 (3)	$12N$	式 (3) +式 (13)	$12N-10L$
式 (4)	$12N$	式 (4)+式 (14)〜(17)	$12N+12L$

4 シミュレーション

4.1 実験条件

符号は符号長 200，符号化率 0.5，H の列重みを 3，行重みを 6 の LDPC 符号のクラス，通信路は白色ガウス通信路を用いる．但し，式 (8) のようなシンボル 1 が 2 つ重複するような箇所が，3 行又は 3 列以上存在するような符号は除外した．SPA，提案アルゴリズム共に最大反復回数は 400 回とする．

4.2 実験 1

上記条件で符号をランダムに 5 つ生成し (Small-Loop の個数は 20 個前後)，それぞれについて SPA，提案アルゴリズムを用いて復号する．図 4.2 に SNR3.25〜5.00 のブロック誤り率のグラフを示す (各 SNR, 符号に対しての実験回数は 600000 回).

4.3 実験 2

上記条件で符号をランダムに 1 つ生成し，ある 1 つの Small-Loop に含まれる 2 つの符号語ビットを意図的に送信誤りとし復号を行う（送信時にビットを反転させる）．その他の符号語ビットは通常通り送信する．表 2 に，各 SNR において 5000 回実験した内の，アルゴリズムの復号誤りパターンの個数を示した.

4.4 考察

実験 2 より Small-Loop 構造のビット部分に誤りが発生した場合，SPA では復号誤りが発生しやすくなるが，Small-Loop をクラスタリングすると，その誤りの発生を抑えることができることがわかる．このことにより，

表 2: 実験 2 結果

SPA	CLUSTER	SNR					
		5.00	4.50	4.00	3.50	3.00	2.50
ERROR (UNDETECT)	RIGHT	180 (138)	263 (190)	377 (222)	423 (208)	512 (197)	544 (171)
ERROR	ERROR	67	126	175	299	575	1068
RIGHT	ERROR (UNDETECT)	2 (2)	2 (1)	6 (2)	18 (8)	35 (9)	85 (6)

※ ERROR は復号誤り，(UNDETECT) は未検出誤りを表す
符号語 2 ビットに意図的にノイズを加えているため SNR は厳密な値ではない

平均的な復号誤り率も Small-Loop をクラスタリングした方がよくなると考えられるが，実験 1 の結果よりこれを確認できた.

5 まとめ

本稿では，SPA でのメッセージ伝播に必要なメモリ量を増加させることなく Small-Loop をクラスタリングするアルゴリズムを提案した．このアルゴリズムは，Small-Loop による SPA の近似性能が悪くなる影響を抑えることができるため，Small-Loop が存在する LDPC のクラスを，実用することのできる符号のクラスとして用いることができるようになった.

今後の課題として，Small-Loop 以外のループ構造をクラスタリングした場合の，その計算量やモデル構築方法の検討などが挙げられる．また，Small-Loop を含む符号のクラスと，Small-Loop を取り除いた符号のクラスの符号性能の差を明確にする必要もある.

謝辞

本研究を行うにあたり，数多くの御助言，御支援を賜りました松嶋研究室，平澤研究室の各氏に感謝致します．なお，本研究の一部は日本学術振興会科学研究費基盤 (C) 一般 (No.15560338) の援助による.

参考文献

[1] D.J.C.Mackay,“Good Error-Correcting Codes Based on Very Sparse Matrices,” *IEEE Trans.Inf. Theory* Vol.45,pp.399-431,Mar,1999.

[2] T.Sato,“Cluster BP and cluster CCCP: Two simple methods for computing Kikuchi approximations” ,Tech. Rep. TR03-0001, Dept. of Computer Science, Tokyo Institute of Technology ,2003.

[3] T.Matsushima,S.Hirasawa,“A formalization of generalized probabilistic reasoning and its procedures on Junction trees”, submitted to *IEEE Trans.Inf. Theory.*

[4] B.J. Frey, F.R. Kschischang, and H.-A. Loeliger,“Factor graphs and the sum-product algorithm” ,*IEEE Trans.Inf. Theory,*Vol.47,pp. 498-519,Feb,2001

[5] 和田山 正,”低密度パリティ検査符号とその復号法”, トリケップス社,2002.