Vol.2015-AL-154 No.1
2015/9/28

実用的な圧縮 Rank/Select 辞書

金田 悠作[1,a)]

概要：Rank/Select 辞書は，入力ビット列 $B[0..n-1]$ 上の $\mathtt{rank}_b(x,B)$ ($B[0..x]$ 中の $b \in \{0,1\}$ の個数を求める操作) と $\mathtt{select}_b(i,B)$ (B の先頭から i 番目の $b \in \{0,1\}$ の位置を求める操作) を実現するデータ構造である．Raman ら (ACM Transactions on Algorithms, 2007) は，入力ビット列を圧縮したまま上述の操作を定数時間で実現するデータ構造を与えている．このデータ構造は，長さ n の入力ビット列を固定長ブロックに分割し，各ブロックを重み（ブロック中の 1 の個数を表す固定長整数）と順位（同じ重みのブロック群に一意に割り振られた可変長整数）の組として符号化する．このブロック符号化とその逆操作であるブロック復号化は，理論的には $o(n)$ ビットの補助領域を用いて定数時間で実現できる．一方，実用的にはこの $o(n)$ ビットの補助領域は無視できないほど大きい．したがって，効率良いブロック符号化・ブロック復号化の実現は，上述のデータ構造を実装する上で重要な課題である．本稿では，効率良いブロック符号化・ブロック復号化アルゴリズムを与え，既存アルゴリズムとの比較実験によりその実用性を示す．

1. はじめに

高速な主記憶上での大規模データ処理を目的に，データを情報理論的下限に近いサイズで表現したまま高速な操作を実現する簡潔データ構造，および，データをエントロピーに近いサイズに圧縮したまま高速な操作を実現する圧縮データ構造が，理論・実用の両面から盛んに研究されている [4], [5], [8], [18]．例えば，圧縮ウェーブレット木 [7], [10] は文字列に対する圧縮データ構造であり，圧縮全文索引を構成する上で重要な役割を果たす [11]．

Rank/Select 辞書は，長さ n のビット列 $B[0..n-1] \in \{0,1\}^n$ と各ビット $b \in \{0,1\}$ に対し，以下の操作を実現するデータ構造である：

- $\mathtt{rank}_b(x,B)$: $B[0..x]$ 中の b の個数を返す．
- $\mathtt{select}_b(i,B)$: $B[0..n-1]$ 中の i 番目の b の位置を返す．

ここで，引数 x と i は，$0 \le x < n$ と $0 \le i \le \mathtt{rank}_b(n-1,B)$ を満たす整数である．上述の操作は様々な簡潔データ構造・圧縮データ構造の内部で利用されるため，高速かつ省領域な Rank/Select 辞書の実現は，簡潔データ構造・圧縮データ構造で現実のデータを扱う上で重要な課題である．

Raman, Raman, Rao [15] は，Rank/Select 辞書に対する圧縮データ構造（以降，RRR と表記）を与えている．RRR は，入力ビット列 B を $\mathcal{B}(n,m)+O(n\log\log n/\log n)$ ビットに圧縮したまま，Rank/Select 操作を定数時間で実現する．ここで，n は B の長さを，m は B 中の 1 の個数を表す．また，$\mathcal{B}(n,m) = \left\lceil \log \binom{n}{m} \right\rceil$ は，B を格納するデータ構造のサイズの情報理論的下限である．RRR は，**ブロック圧縮** [1] によるビット列の圧縮表現，および，高速な Rank/Select 操作を実現するための**簡潔索引**から構成される．

RRR の実用的な実装として，Claude と Navarro の実装 [3] (以降，RRR-CN と表記)，および，Navarro と Providel の実装 [12] (以降，RRR-NP と表記) が提案されている．Navarro と Providel [12] の報告にあるように，RRR の実性能はブロック圧縮の実現方法に大きく依存する．そのため，以降ではブロック圧縮の概要を説明した後，既存実装 [3], [12] におけるブロック圧縮の実現方法を説明する．

ブロック圧縮 [1] は，長さ n のビット列 B を長さ u のブロック B_i ($0 \le i < p = \lceil n/u \rceil$) に分割し，各 B_i を以下で定義される**重み**と**順位**の組として表す．ここで，B_i 中の 1 の個数を $m_i \in [0..u]$ としたとき，B_i の重みは m_i を表す $\lceil \log u \rceil$ ビット整数であり，B_i の順位は $\binom{u}{m_i}$ 個ある重み m_i のブロックに一意に割り振られた $\mathcal{B}(u,m_i) = \left\lceil \log \binom{u}{m_i} \right\rceil$ ビット整数である．このとき，全ブロックの重みと順位は，$\mathcal{B}(n,m)+O(n\log\log n/\log n)$ ビットで表現できる [1], [14]．

[1] 楽天株式会社，楽天技術研究所
Rakuten Crimson House 1–14–1 Tamagawa, Setagaya-ku, Tokyo 158–0094
[a)] yusaku.kaneta@rakuten.com

ⓒ 2015 Information Processing Society of Japan

以降では，ビット列から対応する重みと順位を求める手続きをブロック符号化（あるいは単に符号化）と呼び，重みと順位から対応するビット列を求める手続きをブロック復号化（あるいは単に復号化）と呼ぶ．

Claude と Navarro [3] は，RRR の最初の実用的な実装 RRR-CN を与えた．RRR-CN は，前計算した $O(u2^u)$ ビットの表を用いて，符号化と復号化を共に定数時間で実現する．一方，この実装は，実用的には高い圧縮率を達成できない．これは，$O(u2^u)$ ビットの表によりブロック長 u を大きく設定できず，重みの合計サイズが無視できなくなるためである．例えば，Claude と Navarro [3] は $u = 15$ と設定しているが，その場合の RRR-CN の重みの合計サイズは，入力サイズに対して $4/15 \approx 27\%$ 程度を占める．

Navarro と Providel [12] は，上述の実装 [3] の圧縮率を改善する実装 RRR-NP を与えた．RRR-NP は，前計算した $u^3 + o(u^3)$ ビットの表を用いて符号化と復号化を共に $O(u)$ 時間で実現する．また，ブロック長 u を大きく設定できるため，実用的にも高い圧縮率を達成できる．例えば，Navarro と Providel [12] は $u = 63$ と設定しているが，その場合の RRR-NP の重みの合計サイズは，入力サイズに対して $6/63 \approx 9.5\%$ 程度を占める．

RRR-NP のブロック長に対して線形時間の符号化と復号化は，疎な（0 と 1 の個数に偏りのある）ビット列を扱う場合，ほとんど問題にならない．これは，RRR-NP は，疎なビット列に多く出現する 0，あるいは，1 だけを含むブロックを定数時間で符号化・復号化できるためである．一方，RRR-NP の線形時間の符号化と復号化は，密な（0 と 1 の個数に偏りのない）ビット列を扱う場合，実性能を大きく低下させる．

本稿は，RRR の実用的な実装のために，ブロック圧縮 [1] の既存実装における上述の課題の解決を目的とする．すなわち，RRR-CN [3] のように高速で，RRR-NP [12] のように省領域な符号化法と復号化法を与え，密なビット列に対しても高速に動作する RRR 実装を提案する．

本稿で扱うブロック圧縮問題を以下のように定義する．

問題 (ブロック圧縮問題) ブロック圧縮問題とは，長さ u のビット列 $x \in \{0,1\}^u$, および, 整数 $w \in [0..u]$ と $o \in [0..\binom{u}{w})$ に対して，以下の手続きを実現する問題である．ここで，$|x|_1$ で x 中の 1 の個数を表す：

ブロック符号化: $\mathtt{encode}(x) = (wgt(x), ord_{|x|_1}(x))$
ブロック復号化: $\mathtt{decode}(w, o) = ord_w^{-1}(o)$

ここで，$wgt(x)$ は x からその重みへの写像を，$ord_w(x)$ は x からその順位への全単射を表す．以降では混乱のない限り，$ord_{|x|_1}(x)$ の下添字を省略して，$ord(x)$ と表記する．

1.1 主結果

本稿では，RRR の実用的な実装のために，ブロック圧縮に対する効率良い符号化法と復号化法を提案する．長さ u のビット列に対し，提案手法は，前計算した $u^3 + o(u^3)$ ビットの表を用いて，符号化を $O(u/\log u)$ 時間，復号化を $O(u \log\log u/\log u)$ 時間で実現する（定理 1）．すなわち，提案手法は，最も省領域な既存手法である RRR-NP [12] と漸近的に一致する計算領域で，符号化と復号化をそれぞれ $O(\log u)$ 倍と $O(\log u/\log\log u)$ 倍に高速化する．ブロック圧縮に対する提案手法と既存手法の比較を表 2 に示す．また，計算機実験により，提案した符号化法と復号化法を用いた RRR 実装の密なビット列上での実用性を示す．

表 1 RRR [15] で利用されるブロック圧縮 [1], [14] に対する符号化法と復号化法の比較．ここで，u はブロック長を表し，補助領域はビット数を表す．

	符号化時間	復号化時間	補助領域
手法 [3]	$O(1)$	$O(1)$	$O(u2^u)$
手法 [12]	$O(u)$	$O(u)$	$u^3 + o(u^3)$
提案手法	$O(u/\log u)$	$O(u \log\log u/\log u)$	$u^3 + o(u^3)$

1.2 関連研究

Rank/Select 辞書に対する簡潔データ構造・圧縮データ構造は，理論・実用の両面から盛んに研究されている：Jacobson [9] は Rank 操作を定数時間で実現する簡潔データ構造を，Clark [2] は Select 操作を定数時間で実現する簡潔データ構造を与えた．González ら [6] は上述のデータ構造を実装し，実験的に評価した．Vigna [16] は，計算機のワード内並列性を利用した Rank/Select 辞書の実装を与えた．Zhou ら [17] は，キャッシュ効率の高い Rank/Select 辞書の実装を与えた．

Raman ら [15] は，Rank/Select 辞書に対する圧縮データ構造 RRR を与えた．このデータ構造は入力ビット列を圧縮したまま Rank/Select 操作を定数時間で実現する．Claude と Navarro [3] は，このデータ構造の最初の実用的な実装 RRR-CN を与えた．この実装は，ブロック長を $u = 15$ に設定し，$O(u2^u)$ ビット領域・定数時間で動作するブロックの符号化法・復号化法を用いた．Navarro と Providel [12] は，$u^3 + o(u^3)$ ビット領域・$O(u)$ 時間で動作するブロックの符号化法・復号化法を与え，ブロック長を $u = 31$ と $u = 63$ に設定することで圧縮率を改善した実装 RRR-NP を与えた．Gog と Petri [5] は，RRR-NP を高速化のための実装技法を与えた．Okanohara と Sadakane [13] は，Rank/Select 辞書に対する実用的な圧縮データ構造を与えた．

2. 準備

2.1 基本的な定義

自然数全体の集合を $\mathbb{N} = \{0, 1, \ldots\}$ で表す．整数 $i, j \in \mathbb{N}$ $(i \le j)$ に対し，その区間を $[i..j] = \{i, i+1, \ldots, j\}$ で表す．また，$[i..j) = [i, j-1]$ とする．

ⓒ 2015 Information Processing Society of Japan

文字の有限集合を Σ で表す．Σ 上の長さ n の文字列を $S = s_0 \cdots s_{n-1}$ で表し，その i 番目の文字を $S[i] = s_i \in \Sigma$ $(0 \le i < n)$ で表す．文字列 S の長さを $|S| = n$ で表し，S 中の文字 $\alpha \in \Sigma$ の個数を $|S|_\alpha \in [0..n]$ で表す．長さ 0 の空文字列を ϵ で表す．長さ n の全ての文字列からなる集合を Σ^n で表す．任意の $0 \le i < n$ に対し，S の接頭辞を $S[0..i] = s_0 \cdots s_i$ で表し，接尾辞を $S[i..n-1] = s_i \cdots s_{n-1}$ で表す．したがって，$S = S[0..n-1]$ である．任意の $i < 0$ に対して，$S[0..i] = \epsilon$ とする．長さ n のビット列 $B[0..n-1]$ は $\{0,1\}$ 上の文字列である．また，$Bits_n(m) = \{B \in \{0,1\}^n \mid |B|_1 = m\} \subseteq \{0,1\}^n$ で，長さ n で m 個の 1 を含むビット列の集合を表す．ここで，$|Bits_n(m)| = \binom{n}{m}$ である．

単調増加列を格納した整数配列 $A[0..n-1]$ に対し，$x \in \mathbb{N}$ より小さい最大の要素を $pred(x, A) = \max\{i \in [0..n) \mid A[i] \le x\}$ で表す．ここで，全ての $1 \le i < n$ に対して $A[i-1] < A[i]$ である．

2.2 計算モデル

本稿では，計算モデルとして語長 $\log n$ ビットの *word RAM* を仮定する．このモデルでは，$\log n$ ビット整数に対するビット演算，および，乗算と除算を含む算術演算を定数時間で計算できる．また，メモリの連続する $\log n$ ビットを定数時間で読み書きできる．

3. 提案手法

本節では，RRR [15] の実用的な実装のために，ブロック圧縮 [1] に対する高速かつ省領域な符号化法と復号化法を与える．以降ではブロック長を u で表す．

3.1 ブロック符号化・ブロック復号化の概要

ブロック圧縮の符号化と復号化に対する提案手続きは，長さ u のブロック $x \in \{0,1\}^u$ を長さ k $(0 \le k \le u)$ の局所ブロック（あるいは小ブロック）の列 $x_0, \ldots, x_{p-1}$ に分割し，局所ブロック単位で処理を行う．ここで，$p = \lceil u/k \rceil$ である．以降では，一般性を失うことなくブロック長 u は局所ブロック長 k で割り切れると仮定する．

局所ブロック単位の効率良い処理で重要なのは，効率良く実装可能な順序の計算である．このために，以下で定義される局所ブロックの重みと順位を導入する：

定義 1 (局所ブロックの重みと順位) 局所ブロック $x \in \{0,1\}^k$ の重みを $\mathit{local\text{-}wgt}(x) = |x|_1$ と定義する．また，局所ブロック上の辞書式順序 $>$ に対し，重み $w \in [0..k]$ の局所ブロック $x \in Bits_k(w)$ の順位を $\mathit{local\text{-}ord}_{|x|_1}(x) = |\{y \in Bits_k(w) \mid x > y\}|$ と定義する．

以降では混乱のない限り，$\mathit{local\text{-}ord}_{|x|_1}(x)$ の下添字を省略して，$\mathit{local\text{-}ord}(x)$ と表記する．

次に，局所ブロックの重みと順位から以下の局所ブロック上の全順序を定義する：

定義 2 (局所ブロックの全順序) 任意の長さ $k \le u$ の局所ブロック $x, y \in \{0,1\}^k$ に対する全順序 $x >_k y$ を (i) $\mathit{local\text{-}wgt}(x) > \mathit{local\text{-}wgt}(y)$，または，(ii) $\mathit{local\text{-}wgt}(x) = \mathit{local\text{-}wgt}(y) \wedge \mathit{local\text{-}ord}(x) > \mathit{local\text{-}ord}(y)$ と定義する．

また，任意の長さの局所ブロック列 x, y に対して，全順序 $x >_k y$ を長さ k の局所ブロック上の全順序 $>_k$ から構成される辞書式順序と定義する．このとき，ブロックの順位を以下のように定義する：

定義 3 (ブロックの順位) 長さ u のブロック x 上の順位関数を $\mathit{ord}(x) = |\{y \in \{0,1\}^u \mid x >_k y\}|$ と定義する．

以降では，定義 3 の順序の効率良い実現方法を説明する．

3.2 前処理

RRR-NP [12] におけるブロックの復号化法・復号化法は，二項係数の値を格納した $u^3 + o(u^3)$ ビットの単一の表を用いる．これに対し，提案手法は，局所ブロック単位で処理するために以下の複数の表を用いる．

定義 4 (符号化・復号化のための表) 提案ブロック符号化法・ブロック復号化法で用いる表を以下のように定義する：

疎な二項係数表 $\texttt{Binom}[0..p-1][0..u]$:
整数 $x \in [0..p)$ と $y \in [0..u]$ の全ての可能な組み合わせに対して，$\texttt{Binom}[x][y] = \binom{kx}{y}$ と定義する．

局所重み表 $\texttt{WgtL}[0..2^k-1]$:
全ての可能な局所ブロック $x \in \{0,1\}^k$ に対して，$\texttt{WgtL}[x] = \mathit{local\text{-}wgt}(x) \in [0..k]$ と定義する．

局所重み計数表 $\texttt{WgtCntL}[0..k]$:
全ての可能な局所ブロックの重み $x \in [0..k]$ に対して，$\texttt{WgtCntL}[x] = |\{y \in \{0,1\}^k \mid \mathit{local\text{-}wgt}(y) < x\}|$ と定義する．

局所順位表 $\texttt{OrdL}[0..2^k-1]$:
全ての可能な局所ブロック $x \in \{0,1\}^k$ に対して，$\texttt{OrdL}[x] = \mathit{local\text{-}ord}(x) \in [0..\binom{k}{|x|_1})$ と定義する．

局所ブロック表 $\texttt{BlkL}[0..2^k-1]$:
全ての可能な局所ブロック $x \in \{0,1\}^k$ に対して，$\texttt{WgtL}[\texttt{WgtCntL}[\texttt{WgtL}[x]] + \texttt{OrdL}[x]] = x$ と定義する．

順位計数表 $\texttt{OrdCnt}[0..p-1][0..u][0..k]$:
整数 $x \in [0..p-1)$ と，$y \in [0..u]$，$z \in [0..k]$ の全ての可能な組み合わせに対して，$\texttt{OrdCnt}[x][y][z] = \sum_{w=0}^{z-1} \binom{k}{w}\binom{kx-k}{y-w}$ と定義する．

Algorithm 1 に，定義 4 の表の初期化処理を示す．前処理の計算量に関して以下の補題を得る：

Algorithm 1 表の初期化手続き

```
1: procedure initialize
2:     for x = 0 to 2^k − 1 do
3:         WgtL[x] ← |x|_1
4:     for x = 0 to 2^k − 1 do
5:         OrdL[x] ← WgtCntL[WgtL[x]]
6:         WgtCntL[x] ← WgtCntL[x] + 1
7:     s ← 0
8:     for x = 0 to 2^k − 1 do
9:         WgtCntL[x], s ← s, WgtCntL[x] + s
10:    for x = 0 to 2^k − 1 do
11:        BlkL[WgtCntL[WgtL[x]] + OrdL[x]] ← x
12:    Prev[0..u], Cur[0..u]
13:    for x = 0 to u do
14:        for y = 0 to x do
15:            if x = y ∨ y = 0 then
16:                Cur[y] ← 1
17:            else
18:                Cur[y] ← Prev[y − 1] + Prev[y]
19:        if x mod k = 0 then
20:            Binom[x/k][0..u] ← Cur[0..u]
21:        Prev[0..u], Cur[0..u] ← Cur[0..u], Prev[0..u]
22:    for x = 1 to p−1 do
23:        for y = 0 to u do
24:            for z = 0 to k do
25:                r ← y − z
26:                if 0 ≤ r ≤ kx then
27:                    OrdCnt[x][y][z] ← Binom[1][z]×Binom[x−1][r]
28:            s ← 0
29:            for z = 0 to k do
30:                OrdCnt[x][y][z], s ← s, OrdCnt[x][y][z] + s
```

補題 1 定義 4 の表は $(1+2/k)u^3+ku^2+3k2^k+o(u^3+k2^k)$ ビットかつ $O(u^2+k2^k)$ 時間で計算可能である．

証明． 定義 4 の表のサイズは，疎な二項係数表と順位計数表の各要素を u ビットで，その他の表の各要素を k ビットで表現できることから明らかである．また，任意の局所ブロック $x \in \{0,1\}^k$ に対して，$|x|_1$ を $O(k)$ 時間で計算できることに注意すると，表の初期化時間も明らかである． □

局所ブロック長を $k = \log u$ とすると，表サイズは合計 $u^3+o(u^3)$ ビットになり，RRR-NP の表サイズと一致する．

3.3 ブロック符号化処理

ブロック符号化 encode に対する提案手続きを Algorithm 2 に示す．ブロック符号化は，長さ u のブロック $x \in \{0,1\}^u$ が与えられたときに，その重み $wgt(x)$ と順位 $ord(x)$ を返す手続きである．

補題 2 提案手続き encode は，定義 4 の表集合を用いてブロック復号化を $O(u \log k/k)$ 時間で計算する．

証明． 提案手続きの i 回目の繰り返しの開始において，$w = |x_0 \cdots x_{i-1}|_1$ と $o = |\{y \in Bits_u(|x|_1) \mid x_0 \ldots x_{i-1} >_k y_0 \cdots y_{i-1}\}|$ が成り立つことを i に関する帰納法で示す．

Algorithm 2 提案ブロック符号化手続き

```
1: procedure encode(x = x_0 ··· x_{p−1})
2:     w, o ← 0, 0
3:     for i = 0 to p−1 do
4:         w ← w + WgtL[x_i]
5:     w' ← w
6:     for i = 0 to p−1 do
7:         v ← WgtL[x_i]
8:         r ← OrdL[x_i]
9:         o ← o + OrdCnt[p−i][w'][v]
10:        o ← o + r × Binom[p−i−1][w'−v]
11:        w' ← w' − v
12:    return (w, o)
```

重み表 WgtL の定義から，前半の繰り返し処理の開始において $w = |x_0 \cdots x_{i-1}|$ が成り立つ．また，後半の繰り返し処理の開始において $w' = |x_i \cdots x_{p-1}|$ が成り立つ．

提案手続きの後半の繰り返し処理において，i 回目の繰り返し処理の開始に $o = |\{y \in Bits_u(|x|_1) \mid x_0 \ldots x_{i-1} >_k y_0 \cdots y_{i-1}\}|$ が成り立つことを示す．$i = 0$ のとき，順位変数 $o = 0$ に初期化されているため成り立つ．i 回目の繰り返し処理の開始において成り立つと仮定する．提案手続きは，繰り返し処理において全ての $0 \le j < i$ について $x_j = y_j$ であり，かつ，$x_i >_k y_i$ であるブロックの個数を求める．$x_i >_k y_i$ となるのは，(i) $\mathit{local\text{-}wgt}(x_i) > \mathit{local\text{-}wgt}(y_i)$，または，(ii) $\mathit{local\text{-}wgt}(x_i) = \mathit{local\text{-}wgt}(y_i) \wedge \mathit{local\text{-}ord}(x_i) = \mathit{local\text{-}ord}(y_i)$ の場合であり，(i) を満たすブロックの個数は，順位計数表 $\mathtt{OrdCnt}[p-i][w][v]$ から定数時間で求まる．また，(ii) を満たす y_i あたり，$\binom{k(p-i-1)}{w'-v}$ 個のブロックが存在するので，(ii) を満たすブロックの個数は，$\mathit{local\text{-}ord}(x_i) \times \binom{k(p-i-1)}{w'-v}$ から定数時間で求まる．(i) と (ii) を満たすブロックの個数を o に加算することで $o = |\{y \in Bits_u(|x|_1) \mid x_0 \cdots x_i >_k y_0 \cdots y_i\}|$ が成り立つ．提案ブロック符号化法の重み計算と順位計算は共に $O(u/k)$ 回の繰り返しからなる．また，各繰り返しは定数個の表引きと算術演算からなる．したがって，定数時間で計算可能である． □

3.4 ブロック復号化処理

ブロック復号化 decode に対する提案手続きを Algorithm 3 に示す．ブロック復号化は，長さ u のブロックの重み $w \in [0..u]$ と順位 $o \in [0..\binom{u}{w})$ が与えられたときに，対応するブロック $x \in \{0,1\}^u$ を返す手続きである．

補題 3 提案手続き decode は，定義 4 の表集合を用いてブロック復号化を $O(u \log k/k)$ 時間で計算する．

証明． 復号化されるブロックを $x = x_0 \cdots x_{p-1}$ $(x_i \in \{0,1\}^k)$ とする．提案手続きの i 回目の繰り返しの開始において，$w = |x_i \cdots x_{p-1}|_1$ と $x' = x_0 \cdots x_{i-1}$ が成り立つこと，および，o が $x_0 \cdots x_{i-1}$ を接頭辞としてもつブロッ

ⓒ 2015 Information Processing Society of Japan

Algorithm 3 提案ブロック復号化手続き

```
1: procedure decode(w ∈ [0..u], o ∈ [0..(u choose w)))
2:     x' ← ε
3:     for i = 0 to p−1 do
4:         v ← pred(o, OrdCnt[p−i][w])
5:         r ← (o − OrdCnt[p−i][w][v])/Binom[p−i−1][w−v]
6:         o ← o − OrdCnt[p−i][w][v]
7:         o ← o − r × Binom[p−i−1][w−v]
8:         w ← w − v
9:         x' ← x' · BlkL[WgtCntL[v] + r]
10:    return x'
```

ク中の x の相対順位を表すことを i に関する帰納法で示す．$i = 0$ のとき，$x = \epsilon$ と，$w = wgt(x)$，$o = ord(x)$ に初期化されているため成り立つ．以降，i 回目の繰り返し処理の開始において成り立つと仮定する．

初めに，i 回目の繰り返し処理において $w = |x_{i+1} \cdots x_{p-1}|$ となることを示す：i 回目の繰り返し処理の開始において $w|x_i \cdots x_{p-1}|_1$ である．順位計数表 OrdCnt の定義より，変数 v は $\texttt{OrdCnt}[p-i][w][v-1] \le o < \texttt{OrdCnt}[p-i][w][v]$ を満たし，x_i に対して $\mathit{local\text{-}wgt}(x_i) = |x_i|_1 = v$ が成り立つ．したがって，8 行目で $w = |x_i \cdots x_{p-1}| - |x_i| = |x_{i+1} \cdots x_{p-1}|$ が成り立つ．

次に，i 回目の繰り返し処理において o は，$x_0 \cdots x_i$ を接頭辞としてもつブロック中の x の相対順位を表すことを示す．o を $x_0 \cdots x_i$ を接頭辞としてもつブロック中の x の相対順位に更新するためには，$x_i > y$ を満たす局所ブロック $y \in \{0,1\}^k$ を接頭辞としてもつ $Bits_{k(n-i)}(w)$ の要素の個数を現在の o から引けば良い．$x_i > y$ となるのは，(i) $\mathit{local\text{-}wgt}(x_i) > \mathit{local\text{-}wgt}(y)$，または，(ii) $\mathit{local\text{-}wgt}(x_i) = \mathit{local\text{-}wgt}(y) \wedge \mathit{local\text{-}ord}(x_i) = \mathit{local\text{-}ord}(y)$ の場合である．(i) を満たすブロックの個数は順位計数表 $\texttt{OrdCnt}[p-i][w][v]$ から定数時間で求まる．また，x_i の順位 r は，$(o - \texttt{OrdCnt}[p-i][w][v])/\texttt{Binom}[p-i-1][w-v])$ で求まるから，(ii) を満たすブロックの個数は $r \times \texttt{Binom}[p-i-1][w-v]$ を使って求まる．したがって，6–7 行目で o は $x_0 \cdots x_i$ を接頭辞としてもつブロック中の x の相対順位を表す．

最後に，i 回目の繰り返し処理において $x' = x_0 \cdots x_i$ となることを示す：これは，v と r を用いて局所ブロック表 BlkL から x_i を復元できることから成り立つ．

手続き decode は $O(u/k)$ 回の繰り返しからなり，4 行目を除く処理は，定数個の表引きと算術演算で計算できる．配列 $\texttt{OrdL}[p-i-1][w]$ は $k+1$ 個の整数からなる単調増加列を格納しており，4 行目の処理は二分探索により $O(\log k)$ 時間で計算可能である．したがって，decode は $O(u \log k/k)$ 時間で計算可能である． □

3.5 主定理

補題 1–3 に $k = \log u$ を代入することで，ブロック圧縮に関する以下の定理を得る：

定理 1 (ブロック圧縮の計算量) 長さ u のビット列に対する符号化と復号化は，$O(u^2)$ 時間で前計算可能な $u^3 + o(n^3)$ ビットの表を用いて，それぞれ $O(u/\log u)$ 時間と $O(u \log\log u/\log u)$ 時間で計算可能である．

また，提案手法は，RRR-NP の符号化と復号化と同様に，ブロックの任意の接頭辞のみを部分的に復号化できる．Gog と Petri [5] は，この部分的な復号化により Rank/Select 操作の実性能を大きく改善できることを報告している．

系 1 (ブロックの接頭辞の部分復号化) 提案手続きで符号化された長さ u のビット列の任意の長さ $\ell \in [0..u]$ の接頭辞は，$O(u^2)$ 時間で前計算可能な $u^3 + o(u^3)$ ビットの表を用いて，$O(\ell \log\log u/\log u)$ 時間で復号化可能である．

証明. 提案復号化法を ℓ/k 番目の局所ブロックを復号化した後に打ち切ることにより示される． □

4. 評価実験

本節では，提案した符号化法と復号化法の評価実験の結果を示す．

4.1 実験環境

提案した符号化手続きと復号化手続きの実性能を評価するために， 現在よく利用されている RRR 実装の一つである RRR-NP [12] を Go で実装し，Go 1.5 で実行可能ファイルを生成した．ただし，実験で用いた RRR 実装は，ビット追記で動的に変更可能である．また，実装の簡単化のために，ブロック長を $u = 64$ に固定し，重みを 8 ビット整数に格納した．高速な Rank/Select 操作のために 1 の個数を $2,048$ ビットごとに，0 と 1 の位置を $2,048$ 個ごとに格納した．実験で用いた RRR 実装の Rank 操作は RRR-NP と同様の処理を実装し，Select 操作は統合サンプリング技法 [12] による処理を実装した．RRR-NP の実装の詳細に関しては，文献 [12] を参照されたい．

実験では，RRR のブロック符号化とブロック復号化を以下の三つの手法で実装した：

RRR-NP: RRR-NP のビット単位の符号化法と復号化法．
RRR-K8: 3 節で提案した符号化法と復号化法 ($k = 8$)．
RRR-K16: 3 節で提案した符号化法と復号化法 ($k = 16$)．

RRR-K8 と RRR-K16 の復号化で用いる手続き *pred* は，実装の簡単化のために線形探索で実装した．すなわち，全ての符号化と復号化は $O(u)$ 時間で動作する．

実験環境には，Intel Core i7 (2.5GHz) と 16GB のメモリを搭載した MacBook Pro (Retina, 15-inch Mid 2014) を用いた．入力ビット列として，長さ $n = 2^{28}$ の密なビット列 D ($\rho(D) = |D|_1/|D| = 50\%$) と疎なビット列 S ($\rho(S) = |S|_1/|S| = 1\%$) を一様なビット分布で生成した．

ⓒ 2015 Information Processing Society of Japan

表 2　ブロック圧縮の各実装に対するビット追記 (append) と，Rank ($\texttt{rank}_1$), Select ($\texttt{select}_1$) の計算時間（単位 10^9 秒）．ここで，ビット追記に関しては 2^{28} 回の操作の平均時間であり，Rank と Select に関してはそれぞれ 10^7 回の操作の平均時間である．

入力ビット列	密なビット列 D ($\rho(D) = 50\%$)			疎なビット列 S ($\rho(S) = 1\%$)		
操作	append	$\texttt{rank}_1$	$\texttt{select}_1$	append	$\texttt{rank}_1$	$\texttt{select}_1$
RRR-NP [12]	22.5	355.5	409.7	13.6	126.6	277.9
RRR-K8	19.1	283.7	331.3	13.3	119.9	245.8
RRR-K16	20.0	276.1	322.5	13.6	130.9	260.3

表 2 に実験結果を示す．列 append は各入力ビット列の先頭ビットから順に RRR 実装にビット追記した場合の操作あたりの平均時間を示し，列 $\texttt{rank}_1$ と列 $\texttt{select}_1$ はビット追記操作のあとでそれぞれの操作を $1,000,000$ 回繰り返し実行した際の平均時間を示す．ここで，各操作の引数はあらかじめ生成して主記憶上に格納しておいた．

密なビット列 D に対しては，RRR-K8 と RRR-K16 は，RRR-NP のビット追記を 10% 程度，Rank/Select 操作を 20% 程度高速化している（表 2 の左）．一方，疎なビット列 S に対しては，RRR-NP と同程度の性能であった（表 2 の右）．これは実装した全ての手法において，一様かつ疎な入力ビット列に多く含まれるブロック 0^u と 1^u を定数時間で高速に符号化できるためであると推察できる．

以上の結果より，提案した符号化手続きと復号化手続きを用いた RRR 実装は密なビット列に対する性能を大きく改善しており，また，疎なビット列に対しても既存実装 RRR-NP と同等の性能である．

5. おわりに

ブロック圧縮 [1] の実装方法は，RRR [15] の実性能を大きく左右する．本稿では，入力ビット列長に対して線形時間で動作するビット単位の符号化法・復号化法 [12] を改良し，より高速な小ブロック単位の符号化法・復号化法を提案した．また，提案した符号化法・復号化法を利用した RRR の実装は，既存手法を利用した場合と比較して，密なビット列に対して実用的にも高速に動作することを示した．

謝辞　有益なコメントを頂いた新里圭司氏に感謝する．

参考文献

[1] A. Brodnik, J. I. Munro: Membership in constant time and almost-minimum space. *SIAM J. Comput.*, Volume 28, Number 5, pp. 1627–1640, 1999.
[2] D. Clark: Compact pat trees. Ph.D. thesis, University of Waterloo, Canada, 1996.
[3] F. Claude, G. Navarro: Practical rank/select queries over arbitrary sequences, In *Proc. 15th SPIRE*, pp. 176–187, 2008.
[4] S. Gog, T. Beller, A. Moffat, M. Petri: From theory to practice: plug and play with succinct data structures. In *Proc. 13th SEA*, pp. 326-337, 2014.
[5] S. Gog, M. Petri: Optimized succinct data structures for massive data. *Softw. Pract. Exper.,* Volume 44, Number 11, pp. 1287–1314, 2014.
[6] R. González, S. Grabowski, V. Mäkinen, G. Navarro: Practical implementation of rank and select queries. In *Poster Proc. 4th WEA*, pp. 27–38, 2005.
[7] R. Grossi, A. Gupta, J. S. Vitter: High-order entropy-compressed text indexes. In *Proc. 14th SODA*, pp. 841–850, 2003.
[8] R. Grossi, G. Ottaviano: Design of practical succinct data structures for large data collections. In *Proc. 12th SEA*, pp. 5-17, 2013.
[9] G. Jacobson: Space-efficient static trees and graphs. In *Proc. 30th FOCS*, pp. 549–554, 1989
[10] G. Navarro: Wavelet trees for all. *J. Discrete Algorithms*, Volume 25, pp. 2–20, 2014.
[11] G. Navarro, V. Mäkinen: Compressed full-text indexes. *ACM Comput. Surv.*, Volume 39, Number 1, 2007
[12] G. Navarro, E. Providel: Fast, small, simple rank/select on bitmaps, In *Proc. 11th SEA*, pp. 295–306, 2012.
[13] D. Okanohara, K. Sadakane: Practical entropy-compressed rank/select dictionary. In *Proc. 9th ALENEX*, pp. 60–70, 2007.
[14] R. Pagh: Low redundancy in static dictionaries with constant query time, *SIAM J. Comput.*, Volume 31, Number 2, pp. 353–363, 2001.
[15] R. Raman, V. Raman, S. R. Satti: Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. *ACM Transactions on Algorithms*, Volume 3, Number 4, 2007.
[16] S. Vigna: Broadword implementation of rank/select queries. In *Proc. 7th WEA*, pp. 154–168, 2008.
[17] D. Zhou, D. G. Andersen, M. Kaminsky: Space-efficient, high-performance rank and select structures on uncompressed bit sequences. In *Proc. 12th SEA*, pp. 151–163, 2013.
[18] 岡野原 大輔：高速文字列解析の世界．岩波書店，2012.

ⓒ 2015 Information Processing Society of Japan