Vol.2015-AL-151 No.1
2015/1/13

順序保存符号化 n-gram の高速な出現頻度計算手法

佐藤 雄介[†1,a)]　成澤 和志[†1,b)]　篠原 歩[†1,c)]

概要：順序保存照合問題とは，与えられた系列に対し順序関係を保存する符号化を施し，等しい順序関係をもつ部分系列を照合する問題である．本論文では，順序保存符号化 n-gram の出現頻度を高速に計算するアルゴリズムを提案する．また，系列や n-gram の長さに対する依存が少ない文字オラクルとして，ウェーブレット木を用いた $O(\log\sigma)$ 時間文字オラクルを提案する．さらに計算機実験によって提案アルゴリズムの優位性を確認する．

1. 導入

近年，センサ技術の発達により時系列データの取得が容易になり，様々な解析手法が研究されている．時系列データでの解析において，各値の違いや系列の相対的な位置などに依存しない，形状としての類似性を考慮することは重要である．このような形状の類似性を考慮したパターン照合を行う技術として，**順序保存照合** (order preserving matching) と呼ばれる照合方法がある [3], [6], [8], [9].

順序保存照合問題とは，テキスト T とパターン P の 2 つの系列が与えられたとき，P と等しい順序関係をもつ T の部分系列を照合する問題である．ここで順序関係とは，系列および部分系列中の各値の相対的な大小関係をいう．たとえば系列 $T = (10, 59, 13, 78, 93)$ において，長さ 3 の部分系列 $(10, 59, 13), (59, 13, 78), (13, 78, 93)$ の順序関係は，それぞれ $(1, 3, 2), (2, 1, 3), (1, 2, 3)$ となる．このとき，$P = (31, 95, 39)$ の順序関係は $(1, 3, 2)$ であるため，T の部分系列 $(10, 59, 13)$ と順序保存照合する．

系列の個々の値ではなく値の順序関係あるいは系列の形状に着目する照合問題は，一般的な文字列照合問題の自然な拡張と考えることができる．系列を順序関係で表現することを本論文では**順序保存符号化**，もしくは単に**符号化**と呼ぶ．符号の表現方法は，目的に合わせて様々な方法が提案されてきた [3], [6]．表現方法に関しては 3 節において詳しく説明する．

一方，文字列をはじめとする系列の分類や解析における手法に n-gram カーネル [10] を用いた手法があり，SVM や主成分分析など様々な手法において高い精度を出している．n-gram カーネルは，長さ n の部分文字列を次元とする特徴空間における 2 つの文字列の類似度である．n-gram カーネルを計算するためには入力された文字列に含まれる長さ n の部分文字列の出現頻度を数える必要がある．通常の文字列における部分文字列の出現頻度の計算では，接尾辞木 や接尾辞配列 などのデータ構造を用いることで，それぞれの長さが x, y の 2 つの文字列 X, Y に対して $O(x + y)$ 時間・領域で 2 つの文字列に共通する部分文字列の出現頻度を計算できることが知られている．しかしながら，実験的には，連想配列を用いて素朴に長さ n の部分文字列を数えていく手法の方が高速である．

n-gram カーネルにおいて，特徴空間の次元を順序保存符号化した系列とすることで，**順序保存符号化 n-gram カーネル**に自然に拡張することができ，時系列データに対して有用な解析手法となりえる．ここで問題となるのは，長さ n の部分系列を符号化した系列の出現頻度の計算方法である．一般に，長さ N の系列 $T = (t_1, \ldots, t_N)$ を符号化して得られる系列 $T' = (t'_1, \ldots, t'_N)$ の部分系列 $(t'_i, \ldots, t'_{i+n-1})$ と，系列 T の部分系列 $(t_i, \ldots, t_{i+n-1})$ を符号化した系列 $(t''_1, \ldots, t''_n)$ は異なる．そのため，部分系列 $(t_i, \ldots, t_{i+n-1})$ に対して $N - n + 1$ 回の符号化ををを行わなければならず，長さ n の部分系列に対して高速に符号化を行う手法が必要となる．このような符号化されたすべての符号化部分系列の出現頻度を数える問題を**順序保存符号化 n-gram 出現頻度計算問題**という．

問題 1.1 (順序保存符号化 n-gram 出現頻度計算問題)**.** $\Sigma = \{1, 2, \ldots, \sigma\}$ 上の系列 T が与えられたとき，長さ n のすべての部分系列の符号化系列の出現頻度を求める問題を順序保存符号化 n-gram 出現頻度計算問題という．

本論文では，時系列データに対する順序保存符号化 n-gram の出現頻度問題を高速に計算するアルゴリズムを提

†1 現在，東北大学大学院情報科学研究科
a) yusuke_sato@shino.ecei.tohoku.ac.jp
b) narisawa@ecei.tohoku.ac.jp
c) ayumi@ecei.tohoku.ac.jp

案する．また，任意の系列 T に対する符号化系列 T' の任意の位置 i の値 t'_i を求める機構を**文字オラクル**と呼ぶ．文字オラクルは，本論文で提案するアルゴリズムはもちろん，一般に順序保存照合問題において重要な役割をもつ．本論文では，入力系列の長さや n への依存が少ない文字オラクルとして，ウェーブレット木 [7] を用いた $O(\log\sigma)$ 時間の文字オラクルを提案する．さらに，計算機実験によって提案アルゴリズムが優れていることを示す．

2. 準備

整数の集合 $\Sigma = \{1, 2, \ldots \sigma\}$ をアルファベットと呼び，Σ^* の要素を Σ 上の系列と呼ぶ．系列 T の長さを $|T|$ で表す．長さ 0 の系列を空系列と呼び ε で表す．$1 \le i \le |T|$ に対して，$T[i]$ は T の i 番目の値を表し，$1 \le i \le j \le |T|$ に対して，$T[i:j]$ は i 番目から j 番目までの部分系列 $(T[i], T[i+1], \ldots, T[j])$ を表す．また，$T_{(i,n)}$ は i 番目から長さ n の部分系列 $(T[i], T[i+1], \ldots, T[i+n-1])$ を表す．系列 $X, Y \in \Sigma^*$ および値 $v \in \Sigma$ に対して，Xv は系列 X の末尾に値 v を付け加えた系列 $(X[1], \ldots, X[|X|], v)$ を表し，XY は X と Y の連接 $(X[1], \ldots, X[|X|], Y[1], \ldots, Y[|Y|])$ を表す．また，系列 $T = XYZ$ に対し，系列 X, Y, Z をそれぞれ T の接頭辞系列，部分系列，接尾辞系列と呼ぶ．系列 T における値 t の順位を $r_T(t) = 1 + |\{i \mid T[i] < t \text{ for } 1 \le i \le |T|\}|$ とする．系列 T に対して，順序保存符号化を施すことで得られる符号化系列を $Code(T)$ と表す．特に，系列 T の i 番目の値を開始点にもつ長さ n の部分系列 $T_{(i,n)}$ に対する符号化系列を s_i で表す．また，0, 1 の 2 値からなる系列をビット列と呼ぶ．ビット列に対する記号の表記は整数系列と同様のものとする．

3. 順序保存符号化

3.1 符号化の表現方法

順序保存照合における符号化の表現方法は，用途に応じて様々な方法が提案されてきた．ここでは，これらの表現方法について紹介する．なお，ここでは共通して，T をアルファベット Σ 上の長さ N の系列とする．

自然表現

系列 T に対する自然表現 (natural representation) [8] とは，各値を順位で表現した符号化である．つまり，

$$Code_{nr}(T) = (r_T(T[1]), r_T(T[2]), \ldots, r_T(T[N]))$$

である．たとえば，系列 $T = (20, 12, 31, 40, 9)$ に対する自然表現は，$Code_{nr}(T) = (3, 2, 4, 5, 1)$ となる．自然表現による符号化を行うには与えられた系列に対してソートを行い，ソート後の系列を与えるような添字の列を考えればよい．Kim ら [8] は自然表現による符号化を用いて順序保存照合問題を定義するとともに，$O(Nm\log m)$ 時間の素朴な逐次パターン照合アルゴリズムを提案した．ここで，m はパターン系列の長さである．

接頭辞表現

系列 T に対する接頭辞表現 (prefix representation) [8] とは，i 番目の値 $T[i]$ を接頭辞系列 $T[1:i]$ における順位で表現した符号化である．つまり，

$$\begin{aligned} &Code_{pr}(T) \\ &= (r_{T[1:1]}(T[1]), r_{T[1:2]}(T[2]), \ldots, r_{T[1:N]}(T[N])) \end{aligned}$$

である．ただし，この表現方法では，系列の値はすべて異なるものでなければならない．たとえば，系列 $T = (20, 12, 31, 40, 9)$ に対する接頭辞表現は，$Code_{pr}(T) = (1, 2, 3, 4, 1)$ となる．接頭辞表現による符号化を行うには，順序統計木 (order statistic tree) [5] と呼ばれるデータ構造を用いた効率的なアルゴリズムが提案されている．接頭辞表現による符号化を用いることで，Knuth−Morris−Platt (KMP) アルゴリズムを自然に拡張した $O(N\log m)$ 時間の逐次パターン照合アルゴリズムが Kim らよって提案されている [8].

近傍表現

系列 T および任意の $1 \le i \le N$ に対して，

$$v_T^p(i) = ltind, \quad v_T^n(i) = gtind$$

とする．ここで，

$$\begin{aligned} ltind &= \operatorname*{argmax}_{1\le k<i}\{T[k] \mid T[k] < T[i]\}, \\ gtind &= \operatorname*{argmin}_{1\le k<i}\{T[k] \mid T[k] > T[i]\} \end{aligned}$$

であり，そのような $ltind, gtind$ が存在しないときは $v_T^p(i) = -\infty, v_T^n(i) = \infty$ とする．このとき，系列 T に対する近傍表現 (nearest neighbor representation) [8] とは，二項組 $(v_T^p(i), v_T^n(i))$ で表現した符号化である．つまり，

$$\begin{aligned} &Code_{nnr}(T) \\ &= ((v_T^p(1), v_T^n(1)), (v_T^p(2), v_T^n(2)), \ldots, (v_T^p(N), v_T^n(N))) \end{aligned}$$

である．たとえば，系列 $T = (20, 12, 31, 40, 9)$ に対する近傍表現は，$Code_{nnr}(T) = ((-\infty, \infty), (-\infty, 1), (1, \infty), (3, \infty), (-\infty, 2))$ である．近傍表現による符号化に対しては，接頭辞表現と同様に順序統計木を用いたアルゴリズムが提案されている．また，近傍表現を用いた場合には，$T[v_T^p(i)] < T[i] < T[v_T^n(i)]$ の真偽を確認するだけで接頭辞系列 $T[1:i]$ における $T[i]$ の順位を調べることができる．この性質を用いて系列の照合を効率化することで，KMP アルゴリズムに基づく $O(N + m\log m)$ 時間の逐次パターン照合アルゴリズムが提案されている [8].

表 1 各表現において，アルファベットサイズ σ である長さ N の系列 T の，長さ n の部分系列 $T_{(i,n)}$ を符号化するための前処理と符号化時間.

表現名	系列中の値	前処理を行う系列	前処理時間	符号化時間
自然表現	同じ値を許す	部分系列 $T_{(i,n)}$	$O(n \log n)$	$O(n)$
接頭辞表現	すべて異なる	部分系列 $T_{(i,n)}$	$O(n \log n)$	$O(n \log n)$
近傍表現	同じ値を許す	部分系列 $T_{(i,n)}$	$O(n \log n)$	$O(n \log n)$
$\alpha\beta$ 表現	同じ値を許す	部分系列 $T_{(i,n)}$	$O(n \log\log \sigma)$	$O(n \log\log \sigma)$
計数表現	同じ値を許す	系列 T に対して 1 度だけ	$O(N\sqrt{\log N})$	$O(n \frac{\log N}{\log\log N})$

$\alpha\beta$ 表現

系列 T および任意の $1 \le i \le N$ に対して，

$$\alpha_T(i) = i - leqind, \quad \beta_T(i) = i - geqind$$

とする．ここで，

$$leqind = \operatorname*{argmax}_{1 \le k < i} \{T[k] \mid T[k] \le T[i]\},$$
$$geqind = \operatorname*{argmin}_{1 \le k < i} \{T[k] \mid T[k] \ge T[i]\}$$

であり，そのような $leqind, geqind$ が存在しないときは $\alpha_T(i) = i, \beta_T(i) = i$ とする．このとき，系列 T に対する $\alpha\beta$ 表現 (α-β representation) [6] とは，二項組 $(\alpha_T(i), \beta_T(i))$ で表現した符号化である．つまり，

$$Code_{\alpha\beta r}(T)$$
$$= ((\alpha_T(1), \beta_T(1)), (\alpha_T(2), \beta_T(2)), \ldots, (\alpha_T(N), \beta_T(N)))$$

である．たとえば，系列 $T = (20, 12, 31, 40, 9)$ に対する $\alpha\beta$ 表現は $Code_{\alpha\beta r}(T) = ((1,1), (2,1), (2,3), (1,4), (5,3))$ となる．$\alpha\beta$ 表現に対しては，y-fast 木と呼ばれるデータ構造を用いることで効率的に符号化を行うことができる．$\alpha\beta$ 表現は，順序保存不完全接尾辞木 (order-preserving incomplete suffix tree) と呼ばれる，順序保存を行う索引構造を効率的に構築するために，Chrochemore らによって提案された表現方法である [6]．系列 T に関する順序保存不完全接尾辞木を用いた場合，長さ m のパターン系列 P に対して，T 中における P と順序保存照合するすべての部分系列の位置を $O(m \log\log m + occ)$ 時間で求めることができる．ここで，occ はパターン系列が順序保存照合する部分系列の出現頻度を表す.

計数表現

系列 T および任意の $1 \le i \le N$ に対して，

$$p_T^{<}(i) = |\{k \mid k < i, T[k] < T[i]\}|,$$
$$p_T^{=}(i) = |\{k \mid k < i, T[k] = T[i]\}|$$

とする．このとき，系列 T に対する計数表現 (counting representation) [6] とは，二項組 $(p_T^{<}(i), p_T^{=}(i))$ で表現した符号化である．つまり，

$$Code_{cr}(T)$$
$$= ((p_T^{<}(1), p_T^{=}(1)), (p_T^{<}(2), p_T^{=}(2)), \ldots, (p_T^{<}(N), p_T^{=}(N)))$$

である．たとえば，系列 $T = (20, 12, 31, 40, 9)$ に対する計数表現は，$Code_{cr}(T) = ((0,0), (0,0), (2,0), (3,0), (0,0))$ である．計数表現による符号化は，区間計数問題 (range counting problem) [2] に帰着することができ，入力系列に対する前処理を行うことで，任意の値に対する符号化を高速に求めることができる．計数表現は，順序保存完全接尾辞木 (order-preserving complete suffix tree) と呼ばれる索引構造を効率的に構築するために提案された [6].

表 1 に，Σ 上の長さ N の系列 T が与えられたとき，長さ n の部分系列 $T_{(i,n)}$ を符号化するために必要なデータ構造の構築やソートなどの前処理および符号化時間を表現ごとに示す．また，自然表現を除く 4 つの表現方法について以下の性質が成り立つ.

補題 3.1. 任意の 2 つの系列 S, T において，以下の関係が成り立つ.

$$Code_{pr}(S) = Code_{pr}(ST)[1 : |S|]$$
$$Code_{nnr}(S) = Code_{nnr}(ST)[1 : |S|]$$
$$Code_{\alpha\beta r}(S) = Code_{\alpha\beta r}(ST)[1 : |S|]$$
$$Code_{cr}(S) = Code_{cr}(ST)[1 : |S|]$$

Proof. 接頭辞表現において，定義より，$Code_{pr}(S) = (r_{S[1:1]}(S[1]), r_{S[1:2]}(S[2]), \ldots, r_{S[1:|S|]}(S[|S|]))$ である．また，同様に定義より，$Code_{pr}(ST) = (r_{ST[1:1]}(ST[1]),\ r_{ST[1:2]}(ST[2]), \ldots,\ r_{ST[1:|S|]}(ST[|S|]), r_{ST[1:|S|+1]}(ST[|S|+1]), \ldots, r_{ST[1:|ST|]}(ST[|ST|]))$ である．ここで，任意の $1 \le i \le |S|$ に対して，$S[1:i] = ST[1:i]$ および $S[i] = ST[i]$ であるため，$r_{S[1:i]}(S[i]) = r_{ST[1:i]}(ST[i])$ である．よって，$Code_{pr}(S) = Code_{pr}(ST)[1 : |S|]$ である．同様に，他の表現についても定義より成り立つ． □

4. 素朴な手法

ここでは，順序保存符号化された n-gram の出現頻度を求める素朴な方法を紹介する．Algorithm 1 に，素朴なアルゴリズムを示す．このアルゴリズムでは，与えられた系列 T の長さ n の部分系列を符号化し，連想配列に格納していくことで，出現頻度を計算している．このとき，符号化の各表現方法によって 3 行目の符号化の計算時間が異なる．表 1 より，自然表現，接頭辞表現，近傍表現，$\alpha\beta$ 表現の 4 つの表現方法では，長さ n の部分系列それぞれに対し

Algorithm 1: 順序保存符号化 n-gram の出現頻度計算に対する素朴なアルゴリズム

Input: 長さ N の系列 T
Output: T のすべての順序保存符号化 n-gram とその出現頻度

```
1 連想配列 F;
2 for i = 1 to N − n + 1 do
3     s = Code(T_(i,n));
4     if s ∈ F then F[s]++ ;
5     else F[s] := 1 ;
6 output F;
```

て，データ構造の構築やソートなどの前処理が必要であるが，計数表現は入力された系列 T に対して，1 度だけ前処理を行えばよい．

5. 提案手法

5.1 更新による符号化アルゴリズム

Algorithm 1 では，長さ n の部分系列を符号化するとき，毎回独立した計算による符号化を行っていた．ここでは，$i-1$ 番目と i 番目の部分系列がほとんど同じ系列であることを利用し，$T_{(i-2,n)}$ の符号化系列 s_{i-1} を更新することで $T_{(i,n)}$ の符号化系列 s_i を求めるアルゴリズムを提案する．

s_{i-1} を更新することにより s_i を求めるアルゴリズムを，Algorithm 2 に示す．5〜6 行目に示すループでは，$i-1$ 番目の符号化部分系列の先頭の値を 1 つ削った $s_{i-1}[2:n]$ を，i 番目の符号化部分系列の最後の値を 1 つ削った $s_i[1:n-1]$ へと更新している．この更新方法は表現方法によって異なる．

自然表現では，s_{i-1} の先頭の値を 1 つ削った場合，残りの系列に対して再度ソートしなおす必要がある．近傍表現および $\alpha\beta$ 表現では，s_{i-1} の先頭を削った場合，削った値が $s_i[j]$ の対象範囲で最大値であるか最小値であるかによって，更新しなければならない．そのため，s_{i-1} の値を利用せずに s_i を計算する場合と計算時間は変わらない．

一方，接頭辞表現と計数表現では，系列の先頭 $T[i-1]$ を削った場合，先頭以降の符号化系列の値 $Code(T_{(i,n-1)})[j]$ は，$T[i-1]$ と $T[i+j-1]$ の大小関係によって 1 減少するかどうかという変化しかしない．そのため，以下の更新式を用いることで符号化系列の値 $s_i[j]$（ただし，$1 \le j \le n-1$）を定数時間で更新することができる．

接頭辞表現の場合：

$$
\begin{aligned}
& s_i[j] \\
=\ & r_{T_{(i,n)}}(T[i+j-1]) \\
=\ & update_T(i,j,s_{i-1}) \\
=\ & \begin{cases} r_{T_{(i-1,n)}}(T[i+j-1]) - 1 \\ \qquad (T[i-1] < T[i+j-1] \text{ のとき}), \\ r_{T_{(i-1,n)}}(T[i+j-1]) \qquad (\text{その他}) \end{cases}
\end{aligned}
$$

Algorithm 2: 更新による高速な符号化を実現した提案アルゴリズム

Input: 長さ N の文字列 T
Output: T のすべての順序保存符号化 n-gram とその出現頻度

```
1  連想配列 F;
2  s_1 := Code(T[1 : n]);
3  F[s_1] := 1;
4  for i := 2 to N − n + 1 do
5      for j := 1 to n − 1 do
6          s_i[j] := update_T(i, j, s_{i−1});
7      s_i[n] := Code(T_(i,n))[n];
8      if s_i ∈ F then F[s_i]++ ;
9      else F[s_i] := 1 ;
10 output F;
```

計数表現の場合：

$$
\begin{aligned}
& s_i[j] \\
=\ & (p^{<}_{T_{(i,n)}}(j), p^{=}_{T_{(i,n)}}(j)) \\
=\ & update_T(i,j,s_{i-1}) \\
=\ & \begin{cases} (p^{<}_{T_{(i-1,n)}}(j+1) - 1,\ p^{=}_{T_{(i-1,n)}}(j+1)) \\ \qquad (T[i-1] < T[i+j-1] \text{ のとき}), \\ (p^{<}_{T_{(i-1,n)}}(j+1),\ p^{=}_{T_{(i-1,n)}}(j+1) - 1) \\ \qquad (T[i-1] = T[i+j-1] \text{ のとき}), \\ (p^{<}_{T_{(i-1,n)}}(j+1),\ p^{=}_{T_{(i-1,n)}}(j+1)) \qquad (\text{その他}) \end{cases}
\end{aligned}
$$

これにより，長さ n の部分系列 $T_{(i,n)}$ のうち，実際に符号化を行う必要があるのは部分系列の最後の $T[i+n-1]$ だけである．この計算は接頭辞表現の場合，$O(\log n)$ 時間でデータ構造を更新した後，$O(\log n)$ 時間で計算できる．計数表現の場合は，データ構造を再構築することなく，$O(\frac{\log N}{\log\log N})$ 時間で計算することができる．ここで，補題 3.1 により，接頭辞表現および計数表現では任意の符号化系列の後ろに符号化された値を連接できるため，$T[i+n-1]$ は独立して符号化しても問題ない．

これらの議論により，以下の定理が成り立つ．

定理 5.1. 長さ N のすべて異なる値の系列 T が与えられたとき，Algorihtm 2 は，$O(Nn\log n)$ 時間ですべての接頭辞表現符号化 n-gram とその出現頻度を，$O(Nn\frac{\log N}{\log\log N})$ 時間ですべての計数表現符号化 n-gram とその出現頻度を求めることができる．

表 1 に示したように，接頭辞表現は系列中のすべての値が異なることを前提としているため実用的ではない．一方，計数表現では任意の系列を扱うことができ，7 行目の符号化においても，素朴な方法では $O(n)$ 時間，データ構造を使えば $O(\frac{\log N}{\log\log N})$ 時間で計算できる．しかし，これらの方法は N や n に依存しており，センサデータなど長時間の計測による系列では大きな問題となる．そこで，本論文では系列の長さへの依存が少ない符号化方法を提案する．

表 2 ウェーブレット木で実現する操作の一部

操作	内容	時間計算量
$access(T,i)$	文字 $T[i]$ を復元する	$O(\log\sigma)$
$rank_c(T,i)$	文字列 $T[1:i-1]$ の文字 c の出現頻度を返す	$O(\log\sigma)$
$select_c(T,i)$	文字列 T 中の $i+1$ 番目の文字 c の出現位置を返す	$O(\log\sigma)$
$rangefreq(T,b,e,x,y)$	文字列 $T[b:e-1]$ 中に出現する $x \le c < y$ を満たす文字 c の合計出現数を返す	$O(\log\sigma)$
$nextvalue(T,b,e,x,y)$	文字列 $T[b:e-1]$ の中で $x \le c < y$ を満たす最大の c を返す	$O(\log\sigma)$
$prevvalue(T,b,e,x,y)$	文字列 $T[b:e-1]$ の中で $x \le c < y$ を満たす最小の c を返す	$O(\log\sigma)$

5.2 ウェーブレット木による符号化

Algorithm 2 において，最も計算時間を必要とするのは，7 行目に示す符号化の計算である．計数表現では，素朴な方法で計算した場合でも $O(n)$ 時間，また，Chan らの提案したデータ構造 [2] を用いた場合でも $O(\frac{\log N}{\log\log N})$ 時間を要する．ここでは，入力系列や n への依存が少ない符号化の方法として，ウェーブレット木を用いた $O(\log\sigma)$ 時間の符号化手法を提案する．

ウェーブレット木は，Grossi ら [7] によって提案された文字列に対する様々な操作を高速に実現するデータ構造である [1]．ウェーブレット木はアルファベット木と呼ばれる二分木の各節点にビット列を保持させた木構造であり，葉にはアルファベットに含まれる各値が小さい順に置かれ，各節点にはその子孫の葉に置かれた値の和集合が対応する．ウェーブレット木で実現する操作の一部と計算時間について表 2 に示す．

はじめに，長さ N の系列 T に対するウェーブレット木の構築を考える．アルファベット木の根に系列 T 全体を対応させ，根のビット列を B_* で表す．系列の各値 $T[i]$ について，$T[i]$ が左の子の子孫に存在すれば $B_*[i]=0$，右の子の子孫にあれば $B_*[i]=1$ とし，$T[i]$ を該当する子へと振り分ける．これによりビット列 B_* が完成する．同様の操作を葉に至るまで各節点で再帰的に行うことでウェーブレット木を構築することができる．σ をアルファベットサイズとすると，根から葉までは $O(\log\sigma)$ 時間でたどれるためウェーブレット木は $O(N\log\sigma)$ 時間で構築できる．

ウェーブレット木を用いた計数表現による符号化方法を示す．系列 T における i 番目の長さ n の部分系列の計数表現による符号化系列 $Code_{cr}(T_{(i,n)})$ の j 番目の値 $Code_{cr}(T_{(i,n)})[j] = (p^{<}_{T_{(i,n)}}(j), p^{=}_{T_{(i,n)}}(j))$ は以下のように求めることができる．

$$p^{<}_{T_{(i,n)}}(j) = rangefreq(T,i,i+j,1,T[i+j]),$$
$$p^{=}_{T_{(i,n)}}(j) = rangefreq(T,i,i+j,1,T[i+j+1]) - p^{<}_{T_{(i,n)}}(j)$$

既存の操作である *rangefreq* を用いた場合，2 回の操作が必要である．そこで，本論文では 1 回の操作で $p^{<}_{T_{(i,n)}}(j)$ および $p^{=}_{T_{(i,n)}}(j)$ を同時に求める操作 *rangelesseq* を提案する．

Algorithm 3 に，*rangelesseq* 操作のアルゴリズムを示す．*rangelesseq* 操作は $rangelesseq(T,b,e,x,y)$ の形で実行され，$T[b:e-1]$ 中に出現する $x \le c_1 < y$ を満たす文字 c_1 の出現数 lt と，$c_2 = y$ を満たす文字 c_2 の出現数 eq を返す

Algorithm 3: $rangelesseq(T,b,e,x,y)$: $T[b:e-1]$ 中に出現する $x \le c_1 < y$ を満たす文字 c_1 の出現数 lt と $c_2 = y$ を満たす文字 c_2 の出現数 eq を返す

```
1  Function rangelesseq( T, b, e, x, y)
2      t := WT.root;
3      lt := 0;
4      while t が葉ノードでない do
5          ob := B_t[1 : b − 1] 中の 1 の合計出現数;
6          oe := B_t[1 : e − 1] 中の 1 の合計出現数;
7          if y に対応する子孫が t の左の子に存在する then
8              b := b − ob;
9              e := e − oe;
10             t を左の子へ更新;
11         else
12             lt := lt + (e − b) − (ob − oe);
13             b := ob;
14             e := oe;
15             t を右の子へ更新;
16     eq := e − b;
17     return (lt, eq);
```

す文字 c_1 の出現数 lt と，$c_2 = y$ を満たす文字 c_2 の出現数 eq を報告する．すなわち，$p^{<}_{T_{(i,n)}}(j)$ と $p^{=}_{T_{(i,n)}}(j)$ は $rangelesseq(T,i,i+j,1,T[i+j])$ によって得られる 2 つの値 lt および eq と一致している．

rangelesseq 操作は各節点において t と b,e の更新，および左の子の子孫に対応する値の数え上げを行いながら，根から y に対応する葉までたどることで完了する．各節点において，左の子をたどる場合には b,e はそれぞれ $B_t[1:b-1], B_t[1:e-1]$ に出現する 0 の個数に更新する．右の子をたどる場合には，b,e をそれぞれ $B_t[1:b-1], B_t[1:e-1]$ 中の 1 の合計出現数へと更新するとともに，加えて $B_t[b:e-1]$ に出現する 0 を数え上げる．なぜなら，このときに左の子へと移動する値はどれも y よりも小さいからである．葉に至るまでに数え上げた 0 の総数は，$T[b:e-1]$ における y 未満の値の合計出現数である．また，葉に到達した際の b と e の差 $|b-e|$ は $T[b:e-1]$ における y と等しい値の合計出現数である．*rangelesseq* 操作では根から特定の葉までをたどり，各節点においてビット列 B_t における 1 の数え上げを行う．各ビット列を

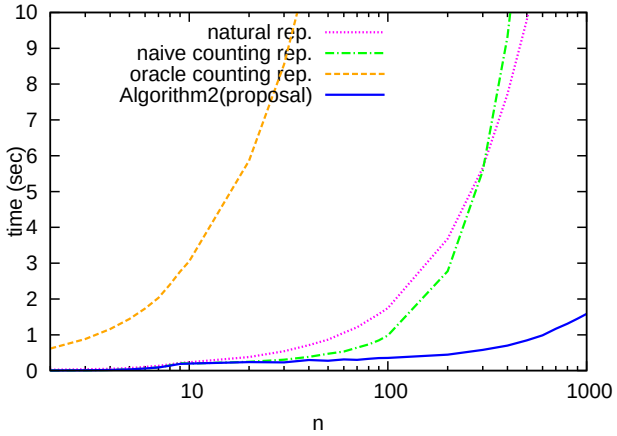


図 1 Algorithm 2（proposal）と，ソートによる自然表現での符号化を行う Algorithm 1（natural rep.），計数表現を用いた Algorithm 1 において素朴な符号化を行う場合（naive counting rep.）と，データ構造を用いた符号化を行う場合（oracle counting rep.）による順序保存符号化 n-gram 出現頻度の計算時間（秒）の比較（長さ $|T| = 100000$，アルファベットサイズ $\sigma = 1000$）

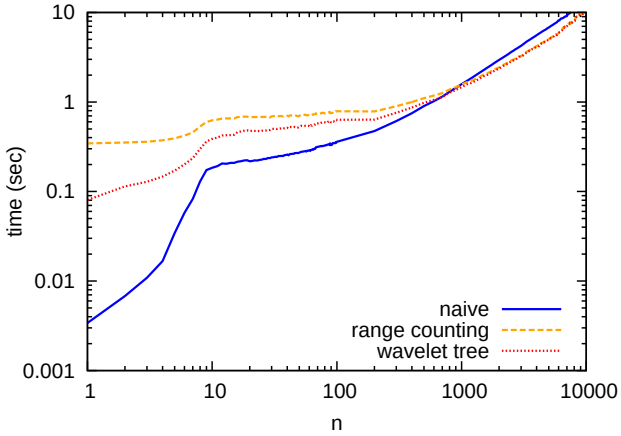


図 2 Algorithm 2 におけるウェーブレット木による符号化（wavelet tree）とデータ構造を用いた符号化（range counting），素朴な符号化（naive）での順序保存符号化 n-gram 出現頻度の計算時間（秒）の比較（長さ $|T| = 100000$，アルファベットサイズ $\sigma = 1000$）

完備辞書 [11] を用いて保持することで 1 または 0 の数え上げを $O(1)$ 時間で行える．したがって，*rangelesseq* 操作は $O(\log \sigma)$ 時間で完了する．

6. 実験

ここでは，順序保存符号化 n-gram 出現頻度計算問題において，提案アルゴリズムを用いた 2 つの計算機実験を行う．1 つ目は，他の表現方法を含めた，素朴なアルゴリズムとの速度比較実験である．2 つ目は，入力系列の長さ N および n，σ を変化させたときの符号化の方法による提案アルゴリズムの速度の違いを観察する．

計測環境は CPU:Intel(R) Xeon(R) Processor E5-2609 (2.40GHz)，メモリ:256GB，OS:Debian wheezy，実装はすべて C++で行う．また，データはすべてパラメータを変化させながらランダムに生成したデータとする．

6.1 アルゴリズムの速度比較

まず，提案アルゴリズムである Algorithm 2 と，素朴なアルゴリズムである Algorithm 1 を比較する．

Algorithm 2 の表現方法は計数表現とし，7 行目における符号化の方法としては，先頭から順に数えていく素朴な方法による符号化を用いる．Algorithm 1 の表現方法および符号化としては次の 3 つを用いる．

- 自然表現によるソートでの符号化 (natural rep.),
- 計数表現による素朴な符号化 (naive counting rep.),
- 計数表現によるデータ構造を用いた符号化 (oracle counting rep.)

自然表現による符号化のソートでは，C++の sort() 関数を使用している．計数表現による素朴な符号化では，各位置に対して先頭から順に数えていくことで符号化を行っている．データ構造を用いた符号化では，Chan らのデータ構造 [2] を用いている．

これら 4 種類の手法について，部分系列の長さ n を変化させながら計算時間を測定する．系列の長さは $N = 100000$，アルファベットサイズは $\sigma = 1000$ とする．各手法の計算時間は 100 回の実行時間の平均を用いるものとし，データ構造を用いる符号化手法は前処理時間も計算時間に含める．

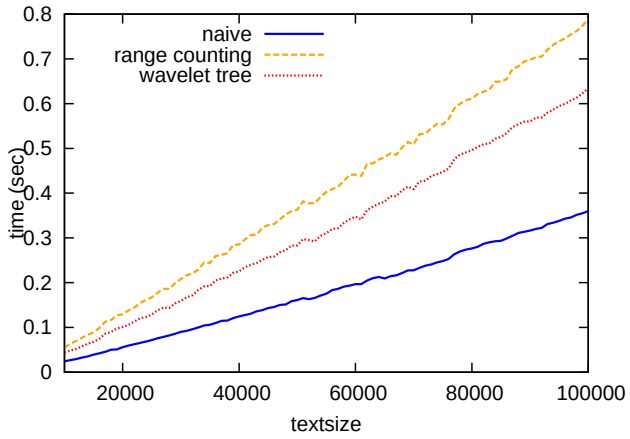
図 1 に結果を示す．図 1 より，任意の n において提案する Algorithm 2 が最も良い結果を示している．Algorithm 2 では，値の符号化は naive counting rep. と同様に行っているものの，部分系列 $T_{(i,n)}$ の符号化に前の符号化系列 s_{i-1} を活用することで値の符号化回数を削減している．

6.2 符号化の方法による違い

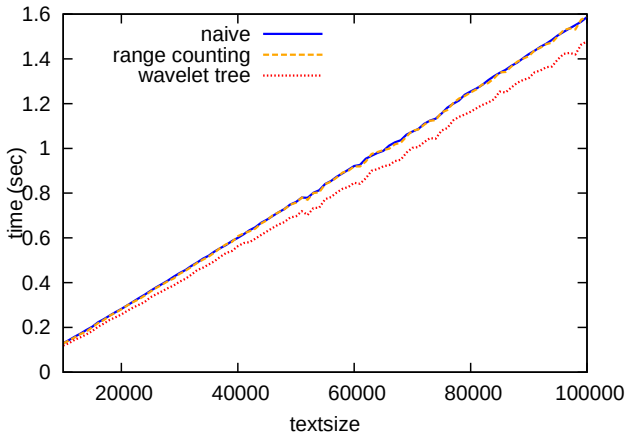
次に，Algorithm 2 において，符号化の方法による速度の違いを実験的に確認する．ここでは，計数表現に対する以下の 3 つの符号化方法について実験を行う．

- 素朴な符号化 (naive)
- データ構造を用いた符号化 (range counting)
- ウェーブレット木を用いた符号化 (wavelet tree)

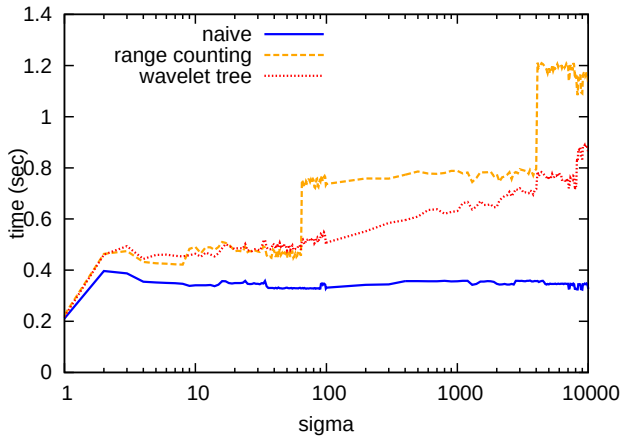
素朴な符号化による結果は，6.1 節の実験における Algorithm 2 と同じである．データ構造を用いた符号化は，6.1 節の実験における oracle counting rep. の符号化方法を Algorithm 2 に適用させたものである．ウェーブレット木を用いた符号化は，5.2 節で提案したウェーブレット木における *rangelesseq* 操作を用いた符号化である．これら 3 つの符号化に対してパラメータを変化させながら計算時間を測定する．6.1 節の実験と同様に，各図において計算時間は 100 回の実行時間の平均を用いて，データ構造およびウェーブレット木の前処理時間も計算時間に含める．



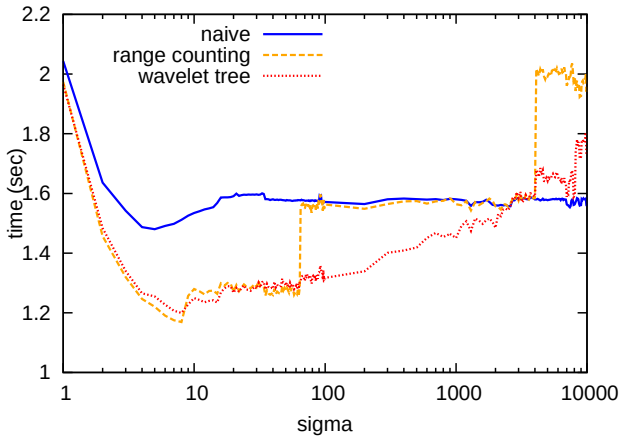
(a) アルファベットサイズ $\sigma = 1000$，部分系列の長さ $n = 100$



(b) アルファベットサイズ $\sigma = 1000$，部分系列の長さ $n = 1000$



(c) 長さ $|T| = 100000$，部分系列の長さ $n = 100$



(d) 長さ $|T| = 100000$，部分系列の長さ $n = 1000$

図 3 Algorithm 2 におけるウェーブレット木による符号化（wavelet tree）とデータ構造を用いた符号化（range counting），素朴な符号化（naive）での順序保存符号化 n-gram 出現頻度の計算時間（秒）の比較

部分系列の長さ n を変化させた様子を図 2 に，系列の長さ N およびアルファベットサイズ σ を変化させた様子を図 3 に示す．図 3(a) と図 3(c) は $n = 100$ としたときの，図 3(b) と図 3(d) は $n = 1000$ としたときの変化の様子である．

図 2 から，$n = 1000$ あたりを境にして，素朴に符号化を行う手法とウェーブレット木あるいはデータ構造を用いて符号化する手法との優劣が入れ替わっている．図 3(a), 3(b) より，系列の長さ N を変化させた場合には，図 2 でも確認したとおり，$n = 100$ の場合には素朴な符号化方法が最も速い．しかしながら $n = 1000$ となると，系列の長さへの依存が少ないウェーブレット木を用いた符号化手法が良い結果を示す．図 3(c), 3(d) より，いずれの結果においても素朴な手法はアルファベットサイズに依存していない．$n = 100$ の場合には，あらゆる σ に対して最も高速であるが，$n = 1000$ の場合では，ウェーブレット木または Chan らのデータ構造を用いた符号化手法が優れた結果を残している．

7. 議論

系列 T の i 番目の接尾辞の j 番目の符号 $Code(T[i:N])[j]$ を返すデータ構造を接尾辞文字オラクル [*1] と呼ぶ [4]．接尾辞文字オラクルとしては，区間木 (range tree) を用いた $\alpha\beta$ 表現に対する接尾辞文字オラクルと，Chan らのデータ構造 [2] を用いた計数表現に対する接尾辞文字オラクルが提案されている [6] が，自然表現や接頭辞表現，近傍表現に対する接尾辞文字オラクルは存在しない．接尾辞文字オラクルは索引構造の構築に用られる．順序保存完全接尾辞木の構築について，$\alpha\beta$ 表現を用いた場合には $O(N \log N)$ 時間の構築アルゴリズムが，計数表現には $O(N \frac{\log N}{\log \log N})$ 時間構築アルゴリズムが提案されている [6]．

一方で，5.2 節で提案したウェーブレット木を用いた符号化方法は，接尾辞系列 $T[i:N]$ に限らず，任意の部分

[*1] 論文 [4] では単に文字オラクルと呼んでいるが，本論文では部分系列に対する文字オラクルと区別するため接尾辞文字オラクルと呼ぶ

系列 $T_{(i,l)}$ に対して，任意の位置 j の符号 $Code(T_{(i,l)})[j]$ を $O(\log\sigma)$ 時間で返す文字オラクルである．したがって，ウェーブレット木を用いた文字オラクルは本論文で提案したアルゴリズムにおいて高速化を実現するだけでなく，接尾辞木などの索引構造の構築にも用いることができる．たとえば，ウェーブレット木による文字オラクルを順序保存完全接尾辞木の構築アルゴリズムに用いることで，$\alpha\beta$ 表現と計数表現のいずれに対しても $O(N\log\sigma)$ 時間の構築が可能となる．

さらに，ウェーブレット木が実現する操作を用いることで，計数表現以外の表現においてもウェーブレット木を文字オラクルとして利用することができる．ここでは，各表現に対する符号化方法を示す．

自然表現を用いた符号化系列の値 $Code_{nr}(T_{(i,l)})[j]$ は，*rangelesseq* 操作を用いて以下のように計算できる．

$$Code_{nr}(T_{(i,l)})[j] = lt + eq$$

ただし，$(lt, eq) = rangelesseq(T, i, i+l+1, 1, T[i+j])$ である．

接頭辞表現は計数表現における $p^{<}_{T_{(i,l)}}(j)$ と等価であるため，$rangelesseq(T, i, i+j, 1, T[i+j])$ によって得られる lt が $Code_{pr}(T_{(i,l)})[j]$ である．

近傍表現による符号化を行うには，*prevvalue*，*nextvalue* および *select* の 3 つの操作を用いる．なお，これらの操作は表 2 に示したとおりすべて $O(\log\sigma)$ 時間で実行できる．

$$v^{p}_{T_{(i,l)}}(j) = select_{c_1}(T_{(i,l)}, 1), \quad v^{n}_{T_{(i,l)}}(j) = select_{c_2}(T_{(i,l)}, 1)$$

ここで，

$$c_1 = prevvalue(T, i, i+j, 1, T[i+j]+1),$$
$$c_2 = nextvalue(T, i, i+j, T[i+j], \infty)$$

である．

$\alpha\beta$ 表現による符号化は，近傍表現と同様の操作で求めた c_1, c_2 を用いて符号化することができる．

$$\alpha_{T_{(i,l)}}(j) = j - select_{c_1}(T_{(i,l)}, 1)$$
$$\beta_{T_{(i,l)}}(j) = j - select_{c_2}(T_{(i,l)}, 1)$$

8. まとめと今後の課題

本論文では，計数表現によるすべての順序保存符号化 n-gram の出現頻度を高速に計算するアルゴリズムおよび，ウェーブレット木を用いた $O(\log\sigma)$ 時間文字オラクルを提案した．また，実験的に提案アルゴリズムが高速であることを示した．さらに，本論文で提案したウェーブレット木を用いた文字オラクルは，計数表現だけでなく，自然表現，接頭辞表現，近傍表現，$\alpha\beta$ 表現のすべてに対して $O(\log\sigma)$ 時間で動作する万能な文字オラクルである．

本研究では，順序保存符号化 n-gram の数え上げを行ったが，各 n-gram の出現頻度は連想配列を用いて保持している．連想配列では要素数 n に対して要素の参照に $O(\log n)$ 時間が必要であり，カーネル計算ではこれがボトルネックである．定数時間に近い時間ですべての n-gram の出現頻度を参照する手法を開発することで，カーネル計算のさらなる高速化が可能となる．

謝辞

本研究は，JSPS 科研費 23300051 および 25240003 の助成によるものである．

参考文献

[1] 高速文字列解析の世界: データ圧縮・全文検索・テキストマイニング. 確率と情報の科学 / 甘利俊一, 麻生英樹, 伊庭幸人編. 岩波書店, 2012.

[2] Timothy M. Chan and Mihai Pătraşcu. Counting inversions, offline orthogonal range counting, and related problems. In *Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 161–173, 2010.

[3] Sukhyeun Cho, Joong Chae Na, Kunsoo Park, and Jeong Seop Sim. A fast algorithm for order-preserving pattern matching. *Information Processing Letters*, Vol. 115, No. 2, pp. 397–402, 2015.

[4] Richard Cole and Ramesh Hariharan. Faster suffix tree construction with missing suffix links. *SIAM J. Comput.*, Vol. 33, No. 1, pp. 26–42, January 2004.

[5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

[6] Maxime Crochemore, Costas S. Iliopoulos, Tomasz Kociumaka, Marcin Kubica, Alessio Langiu, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Order-preserving incomplete suffix trees and order-preserving indexes. In *Proceedings of the 20th Symposium on String Processing and Information Retrieval (SPIRE)*, pp. 84–95, 2013.

[7] Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. High-order entropy-compressed text indexes. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 841–850, 2003.

[8] Jinil Kim, Peter Eades, Rudolf Fleischer, Seok-Hee Hong, Costas S. Iliopoulos, Kunsoo Park, Simon J. Puglisi, and Takeshi Tokuyama. Order-preserving matching. *Theoretical Computer Science*, Vol. 525, No. 13, pp. 68–79, 2014.

[9] Marcin Kubica, Tomasz Kulczynski, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. A linear time algorithm for consecutive permutation pattern matching. *Information Processing Letters*, Vol. 113, No. 12, pp. 430–433, 2013.

[10] Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. Text classification using string kernels. *Journal of Machine Learning Research*, Vol. 2, pp. 419–444, 2002.

[11] Rajeev Raman, Venkatesh Raman, and Srinivasa Rao Satti. Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. *ACM Trans. Algorithms*, Vol. 3, No. 4, November 2007.

「順序保存符号化 n-gram の高速な出現頻度計算手法」 正誤表

位置	誤	正
5.1 節，4 行目	$T_{(i-2,n)}$ の符号化系列 s_{i-1} を更新することで	$T_{(i-1,n)}$ の符号化系列 s_{i-1} を更新することで
5.2 節	$p^{=}_{T_{(i,n)}}(j) = rangefreq(T, i, i+j, 1, T[i+j+1]) - p^{<}_{T_{(i,n)}}(j)$	$p^{=}_{T_{(i,n)}}(j) = rangefreq(T, i, i+j, T[i+j], T[i+j]+1)$